



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO, DE CIÊNCIAS EXATAS E EDUCAÇÃO
DEPARTAMENTO DE ENG. DE CONTROLE, AUTOMAÇÃO E COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Danielle da Silva Romano

**Desenvolvimento de um Software Didático para Simulação de Sistemas
Dinâmicos.**

Blumenau
2025

Danielle da Silva Romano

**Desenvolvimento de um Software Didático para Simulação de Sistemas
Dinâmicos.**

Trabalho de Conclusão de Curso de Graduação em Engenharia de Controle e Automação do Centro Tecnológico, de Ciências Exatas e Educação da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Engenheiro de Controle e Automação.

Orientador: Prof. Luiz Antonio Maccari Junior, Dr.

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Romano, Danielle da Silva

Desenvolvimento de um Software Didático para Simulação
de Sistemas Dinâmicos / Danielle da Silva Romano ;
orientador, Luiz Antonio Maccari Junior, 2025.
128 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Campus Blumenau,
Graduação em Engenharia de Controle e Automação, Blumenau,
2025.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Métodos
Numéricos. 3. Interface Gráfica. 4. Simulação. I. Junior,
Luiz Antonio Maccari. II. Universidade Federal de Santa
Catarina. Graduação em Engenharia de Controle e Automação.
III. Título.

Danielle da Silva Romano

Desenvolvimento de um Software Didático para Simulação de Sistemas Dinâmicos.

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “Engenheiro de Controle e Automação” e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação.

Blumenau, dia de mês de 2025.

Banca Examinadora:



Documento assinado digitalmente
Luiz Antonio Maccari Junior
Data: 16/12/2025 14:08:24-0300
CPF: ***.528.750-**
Verifique as assinaturas em <https://v.ufsc.br>

Prof. Luiz Antonio Maccari Junior, Dr.
Universidade Federal de Santa Catarina



Documento assinado digitalmente
Ebrahim Samer El Youssef
Data: 16/12/2025 15:25:55-0300
CPF: ***.252.109-**
Verifique as assinaturas em <https://v.ufsc.br>

Prof. Ebrahim Samer El Youssef, Dr.
Universidade Federal de Santa Catarina



Documento assinado digitalmente
Marcos Vinicius Matsuo
Data: 16/12/2025 14:06:21-0300
CPF: ***.580.029-**
Verifique as assinaturas em <https://v.ufsc.br>

Prof. Marcos Vinicius Matsuo, Dr.
Universidade Federal de Santa Catarina

Dedico este trabalho à todas as pessoas que eu amo:
minha família, meu namorado e meus amigos, que sempre
acreditaram no meu sonho de concluir a minha graduação.

AGRADECIMENTOS

Gostaria de expressar minha profunda gratidão a todos que contribuíram para a conclusão deste projeto e da minha graduação. Foram longos anos de muitas batalhas, principalmente internas, e, por diversas vezes, não acreditei que seria possível chegar até aqui.

Agradeço imensamente à minha família pelo apoio fundamental: à minha mãe, Iracy Romano; ao meu pai, Evanir Romano; e aos meus irmãos, Elaine Romano, Ian Romano e Ivo Romano, as minhas avós Conceição e Benedita e aos demais familiares que sempre me apoiaram, mesmo à distância, e nunca deixaram de acreditar no meu potencial e em minhas conquistas profissionais e educacionais.

Meu especial agradecimento ao meu companheiro, Alexandre Katzjarowski, por me incentivar, inspirar e permanecer ao meu lado nos momentos mais difíceis, não permitindo que, por muitas vezes, eu desistisse dos meus objetivos, isto foi essencial para que eu conseguisse finalizar esta jornada.

Às minhas amizades, tanto as conquistadas ao longo da vida quanto as formadas durante a jornada acadêmica, que me deram suporte incondicional e acreditaram que este sonho era possível.

Agradeço à minha psicóloga Ana Claudia, por todo acolhimento e preocupação e sempre me ensinar a ver a vida com mais leveza e clareza e me apoiar nos momentos difíceis e a minha psiquiatra Bruna, por também sempre estar ao meu lado na busca da estabilidade e ter uma vida melhor.

Ao meu orientador, Luiz Antonio Maccari, dedico minha imensa gratidão pelo apoio, compreensão e confiança no meu sucesso, mesmo diante das dificuldades pessoais. Sua orientação foi primordial para a conclusão deste projeto, e sua humildade e humanidade me mostraram o papel transformador de um docente na vida de um aluno. Agradeço por cada conversa, conselho e por acreditar que eu seria capaz de alcançar meus objetivos.

Meus sinceros agradecimentos à Universidade Federal de Santa Catarina pela formação de excelência que me proporcionou, pelos desafios que me prepararam e pelas experiências que contribuíram para a profissional que me tornei hoje.

RESUMO

Apesar da ampla utilização de ferramentas computacionais como MATLAB no ensino de engenharia, sua natureza proprietária e custos elevados frequentemente limitam o acesso ao conhecimento, especialmente em um cenário de recursos financeiros limitados. Essa problemática motiva a busca por alternativas *open-source* que mantenham as mesmas capacidades de cálculo e visualização, sem as restrições de licenciamento. Este trabalho apresenta a implementação de métodos numéricos clássicos e avançados (Euler, Runge-Kutta) para a solução de sistemas dinâmicos lineares e invariantes no tempo modelados no espaço de estados, integrados a uma interface gráfica *open-source* desenvolvida em GNU Octave. A ferramenta permite ao usuário selecionar o método desejado, inserir os parâmetros das matrizes do sistema e visualizar os resultados de forma intuitiva. O objetivo é fornecer uma interface computacional eficiente para análise comparativa de técnicas numéricas, com aplicações na disciplina de Modelagem e Simulação de Processos do curso de Engenharia de Controle e Automação. Os resultados demonstram a eficácia da ferramenta como recurso didático, facilitando o estudo de sistemas dinâmicos, auxiliando na compreensão teórica e prática dos métodos numéricos aplicados.

Palavras-chave: Métodos Numéricos; Simulação; Interface Gráfica; GNU Octave.

ABSTRACT

Despite the widespread use of computational tools like MATLAB in engineering education, their proprietary nature and high costs often limit access to knowledge, particularly in resource-constrained settings. This challenge motivated the search for open-source alternatives that preserve the same computational and visualization capabilities without licensing restrictions. This work presents the implementation of classical and advanced numerical methods (Euler, Runge-Kutta) for solving linear time-invariant dynamic systems modeled in state-space form, integrated into an open-source graphical interface developed in GNU Octave. The tool allows users to select the desired method, input system matrix parameters, and intuitively visualize results. Its goal is to provide an efficient computational interface for comparative analysis of numerical techniques, with applications in the Modeling and Simulation of Processes course for Control and Automation Engineering. The results demonstrate the tool's effectiveness as a didactic resource, facilitating the study of dynamic systems and enhancing theoretical and practical understanding of applied numerical methods.

Keywords: Numerical Methods; Simulation; Graphical Interface; GNU Octave.

LISTA DE FIGURAS

Figura 1 – Sistema massa-mola-amortecedor.	25
Figura 2 – Diagrama de Corpo Livre.	25
Figura 3 – Descrição gráfica de um método de um passo.	32
Figura 4 – Aproximação numérica da derivada de v com relação a t por diferença finita.	33
Figura 5 – Tela inicial do GNU Octave.	49
Figura 6 – Editor de <i>scripts</i> do GNU Octave.	50
Figura 7 – Tela inicial guiEditor.	51
Figura 8 – guiEditor executando o <i>script</i> do <i>Software Didático Numerical Methods Interface</i>	52
Figura 9 – Estrutura de diretório de um projeto no guiEditor.	53
Figura 10 – Diagrama de Estrutura e Relação entre os Códigos.	59
Figura 11 – Exemplo de exibição da tabela de resultados.	64
Figura 12 – Visualização Gráfica da simulação implementada.	66
Figura 13 – Plotagem dos diagramas de análise dinâmica.	68
Figura 14 – Tela <i>Numerical Methods Interface</i>	76
Figura 15 – Tela <i>Analysis and Simulation</i>	77
Figura 16 – Tela adicional do <i>software</i>	78
Figura 17 – Tela inicial da interface exibida no <i>Software</i> guiEditor.	86
Figura 18 – Integração das funções no <i>Software</i> guiEditor.	86
Figura 19 – Fluxograma do <i>Software Numerical Methods Interface</i>	88
Figura 20 – Instalação do pacote do <i>Software</i>	90
Figura 21 – Carregamento da interface.	90
Figura 22 – Diagrama das telas existentes no <i>Software Numerical Methods Interface</i>	91
Figura 23 – Tela Inicial do <i>Software Numerical Methods Interface</i>	92
Figura 24 – Tela do <i>software Analysis and Simulation</i>	94
Figura 25 – Resultados obtidos na tabela <i>Results Table</i>	94
Figura 26 – Erro na tela <i>Results Table</i>	95
Figura 27 – Tela <i>Other Options</i>	96
Figura 28 – Tela <i>Error Analysis</i>	97
Figura 29 – Tela <i>Pop-up</i> de aviso ao deixar campos dos parâmetros em branco.	99
Figura 30 – Tela <i>Pop-up</i> de aviso ao não selecionar o tipo de entrada da simulação.	99
Figura 31 – Plotagem gráfica da simulação didática do Exemplo 25.5	105
Figura 32 – Plotagem gráfica da simulação do <i>software</i>	105
Figura 33 – Diagrama do circuito RLC em série utilizado como estudo de caso.	106
Figura 34 – Função de transferência obtida na simulação.	107
Figura 35 – Plotagem gráfica da simulação didática.	108

Figura 36 – Plotagem gráfica da simulação do <i>software</i>	108
--	-----

LISTA DE TABELAS

Tabela 3 – Objetivos e aplicações dos diagramas de análise dinâmica.	66
Tabela 4 – Resultados do Método de Heun do Exemplo 25.5.	102
Tabela 5 – Tabela de resultados obtida pelo <i>software</i>	103
Tabela 6 – Tabela de erros obtida pelo <i>software</i>	103
Tabela 7 – Resultados do Método Ponto Intermediário: Evolução Temporal dos Estados Físicos.	106
Tabela 8 – Tabela de resultados obtida pelo <i>software</i>	107
Tabela 9 – Tabela de erros obtida pelo <i>software</i>	109

LISTA DE ABREVIATURAS E SIGLAS

dB	<i>Decibel (unidade de medida)</i>
DCL	<i>Diagrama de Corpo Livre</i>
EDOs	<i>Equações Diferenciais Ordinárias</i>
GPL	<i>General Public License</i>
GUIs	<i>Graphical User Interfaces</i>
LIT	<i>Linear Time-Invariant System</i>
lsim	<i>Linear Simulation</i>
MIMO	<i>Multiple-Input Multiple-Output</i>
NMI	<i>Numerical Methods Interface</i>
pkg	<i>Packages</i>
PVI	<i>Problema de Valor Inicial</i>
RK1	<i>Método de Runge-Kutta de Primeira Ordem</i>
RK2	<i>Método de Runge-Kutta de Segunda Ordem</i>
RK3	<i>Método de Runge-Kutta de Terceira Ordem</i>
RK4	<i>Método de Runge-Kutta de Quarta Ordem</i>
SISO	<i>Single-Input Single-Output</i>

LISTA DE SÍMBOLOS

m	Massa [kg]
k	Constante elástica da mola [$\frac{N}{m}$]
c	Coefficiente de amortecimento viscoso [$\frac{N \cdot s}{m}$]
x	variável de deslocamento [m]
F_k	Força restauradora da mola [N]
F_b	Força viscosa do amortecedor [N]
F_i	Força de D'Alembert [N]
f_{ext}	Forças externas que agem sobre o corpo [N]
t	Tempo [s]
ζ	Fator de amortecimento [<i>adimensional</i>]
ω_n	Frequência natural [$\frac{rad}{s}$]
ω_d	Frequência amortecida [$\frac{rad}{s}$]
g	Aceleração da gravidade [$\frac{m}{s^2}$]
c	Coefficiente de arrasto [$\frac{kg}{s}$]
v	Velocidade [$\frac{m}{s}$]
v_f	Tensão de entrada no sistema [V]
i_l	Corrente no indutor [A]
v_c	Tensão no capacitor [V]
v_r	Tensão no resistor [V]
L	Indutor [H]
C	Capacitor [F]
R	Resistor [Ω]

SUMÁRIO

1	INTRODUÇÃO	15
1.1	OBJETIVOS	21
1.1.1	Objetivo Geral	21
1.1.2	Objetivos Específicos	22
1.2	ESTRUTURA DO TRABALHO	22
2	REFERENCIAL TEÓRICO	24
2.1	MODELAGEM MATEMÁTICA	24
2.1.1	Exemplo: Sistema massa-mola amortecido	24
2.1.1.1	<i>Equações Diferenciais</i>	26
2.1.2	Solução analítica	27
2.2	MODELAGEM NO ESPAÇO DE ESTADOS	28
2.3	MÉTODOS NUMÉRICOS	30
2.3.1	Métodos Numéricos para Resolução de Sistemas Lineares	31
2.4	APROXIMAÇÃO DA DERIVADA	32
2.5	MÉTODO DE EULER EXPLÍCITO	35
2.5.1	Análise dos Erros de Truncamento no Método de Euler	35
2.5.2	Método de Euler Implícito	36
2.6	MÉTODOS DE RUNGE-KUTTA	37
2.6.1	Runge-Kutta de Segunda Ordem	38
2.6.1.1	<i>Método de Heun</i>	38
2.6.1.2	<i>Método do Ponto Médio (Midpoint)</i>	39
2.6.1.3	<i>Método de Ralston</i>	40
2.6.2	Runge-Kutta de Quarta Ordem	40
2.7	MÉTODOS NUMÉRICOS PARA SISTEMAS NO FORMATO ES- PAÇO DE ESTADOS	41
2.7.1	Euler Explícito Matricial	41
2.7.2	Euler Implícito Matricial	42
2.8	MÉTODOS DE RUNGE-KUTTA NO FORMATO ESPAÇO DE ES- TADOS	43
2.9	TABELA DE BUTCHER	45
2.10	GNU OCTAVE	47
2.11	GUIEDITOR	50
3	SOFTWARE DESENVOLVIDO	55
3.1	DESENVOLVIMENTO DO <i>SOFTWARE</i> DIDÁTICO	55
3.2	<i>SOFTWARE</i> PROPOSTO	55
3.2.1	Lista de funcionalidades	56
3.2.2	Lista de Entradas e Saídas	57

3.2.3	Diagrama de funções	58
3.2.4	Função principal <code>main_v9.m</code>	59
3.2.5	Implementação dos Métodos Numéricos	67
3.2.5.1	Implementação do Método de Euler Explícito Matricial	68
3.2.5.2	Implementação do Método de Euler Implícito Matricial	70
3.2.5.3	Implementação dos Métodos de Runge-Kutta através da Tabela de Butcher	71
3.2.5.4	Implementação do Código <code>metodos_v9.m</code>	73
3.2.6	Programação Interface	75
3.2.6.1	Detalhamento da Função <code>mostrar_TelaInicial.m</code>	79
3.2.6.2	Detalhamento da Função <code>mostrar_TelaSecundaria.m</code>	80
3.2.6.3	Detalhamento da Função <code>mostrar_Tela_OtherOptions.m</code>	83
3.2.7	Integração <code>guiEditor</code> e GNU Octave	85
3.3	FLUXOGRAMA DE OPERAÇÃO DA INTERFACE	87
3.4	INSTALAÇÃO DO PACOTE <code>NMINTERFACE.TAR.GZ</code> NO GNU OCTAVE	89
3.5	DIAGRAMA DAS TELAS EXISTENTES NA INTERFACE	91
3.5.1	Tela Inicial <i>Numerical Methods Interface</i>	92
3.5.2	Tela Principal <i>Analysis and Simulation</i>	93
3.5.3	Tela adicional <i>Other Options</i>	95
3.6	INSTRUÇÕES DE USO DO <i>SOFTWARE</i>	98
3.6.1	Etapas futuras	99
4	RESULTADOS	101
4.1	EXEMPLO 1: SISTEMA LIT COM RESPOSTA ANALÍTICA (Chapra; Canale, 2016)	101
4.2	EXEMPLO 2: SISTEMA ELÉTRICO RLC (Maccari Junior, 2025)	104
5	CONCLUSÃO	110
	REFERÊNCIAS	112
	APÊNDICE A – Código <code>main_v9.m</code>	114
	APÊNDICE B – Código <code>rk_v9.m</code>	120
	APÊNDICE C – Código <code>metodos_v9.m</code>	124
	ANEXO A – Tabelas de Butcher	126

1 INTRODUÇÃO

A simulação de sistemas é uma ferramenta indispensável não apenas no campo da Engenharia, mas também em diversas outras áreas importantes. Seja na otimização de processos industriais, na previsão de cenários econômicos ou até mesmo no gerenciamento de crises ambientais e epidemiológicas, a capacidade de modelar sistemas complexos e prever seu comportamento oferece vantagens estratégicas significativas.

A simulação computacional de sistemas dinâmicos, embora aplicável a diversos contextos, baseia-se em um princípio fundamental comum: a resolução numérica de equações diferenciais que modelam a dinâmica temporal de sistemas. Essa metodologia permite a análise do comportamento do sistema em diversos cenários, constituindo-se como ferramenta essencial tanto para estudos teóricos quanto para aplicações práticas em engenharia. (Kulakowski; Gardner; Shearer, 2007).

Ao reproduzir virtualmente condições reais, a simulação permite que projetistas e analistas testem hipóteses, identifiquem falhas potenciais e validem soluções com menor custo e nenhum risco operacional. Essa abordagem é particularmente relevante para a Engenharia de Controle e Automação, onde a integração entre sistemas físicos e computacionais exige precisão e confiabilidade. Além disso, sua aplicação multidisciplinar — da etapa de projeto à implementação — reforça seu papel como um dos pilares da inovação tecnológica.

A chegada dos computadores digitais no final da década de 1940 marcou o início de uma significativa expansão no desenvolvimento e aplicação de métodos numéricos. Contudo, nos primeiros estágios, a utilização dessas técnicas permaneceu limitada devido aos elevados custos do acesso a computadores que apresentavam um grande porte, o que levou muitos engenheiros a manterem suas abordagens analíticas tradicionais para a maioria dos problemas. A situação transformou-se radicalmente com o surgimento dos computadores pessoais acessíveis, que democratizaram o acesso a capacidade computacional avançada (Chapra; Canale, 2016) permitindo a execução de técnicas avançadas de simulação.

Os métodos numéricos destacam-se como ferramentas essenciais para a resolução de problemas complexos na engenharia moderna. Como destacado por Chapra e Canale (2016), essas técnicas permitem abordar equações diferenciais, lineares e não-lineares, complexas que frequentemente surgem na prática profissional, mas que se mostram não tratáveis ou de difícil solução por meio de abordagens puramente analíticas. A habilidade de transformar desafios matemáticos abstratos em soluções computacionais práticas amplia de maneira significativa a capacidade de resolução de problemas na área de controle e automação.

A análise comparativa entre diferentes métodos numéricos constitui etapa fundamental no processo de modelagem computacional. Como destacam Quarteroni, Sacco e Saleri (2010), a existência de múltiplas abordagens numéricas para um mesmo problema -

como os métodos de Runge-Kutta, Euler e Adams-Bashforth para integração de equações diferenciais - demanda avaliação criteriosa de precisão, estabilidade e eficiência computacional em cada aplicação específica. Chapra e Canale (2016) reforçam que a comparação sistemática entre algoritmos alternativos permite selecionar a técnica mais adequada ao problema em questão, balanceando erros de truncamento, esforço computacional e características particulares do sistema dinâmico analisado.

Além de sua aplicação direta na solução de problemas de engenharia, os métodos numéricos servem para o aluno como plataforma valiosa para o desenvolvimento de competências computacionais. Sua implementação prática em linguagens de programação não apenas consolida o aprendizado de algoritmos numéricos, mas também evidencia de maneira concreta tanto o potencial quanto as limitações naturais do processamento computacional.

Do ponto de vista da formação matemática, os métodos numéricos oferecem uma perspectiva singular, decompondo conceitos avançados em operações aritméticas fundamentais. Como observam Chapra e Canale (2016), essa abordagem não apenas torna acessíveis tópicos matemáticos complexos, mas também proporciona uma compreensão mais profunda e intuitiva dos princípios básicos envolvidos. Para o engenheiro de controle e automação, esse entendimento aprimorado traduz-se em maior capacidade para modelar sistemas dinâmicos, desenvolver algoritmos de controle e otimizar processos industriais.

Kulakowski, Gardner e Shearer (2007) complementam que a eficácia desses métodos reside na capacidade de balancear erros de truncamento e estabilidade, garantindo que soluções aproximadas mantenham fidelidade com a dinâmica real do sistema. Na prática, isso é essencial para aplicações como:

- Projeto de controladores com simulação prévia de resposta temporal;
- Análise de estabilidade de sistemas não-lineares por linearização numérica;
- Otimização de parâmetros em modelos de plantas industriais.

Diversos sistemas - sejam mecânicos, elétricos, térmicos ou até mesmo biológicos e econômicos - podem ter sua dinâmica caracterizada em termos de equações diferenciais. Tais equações são obtidas mediante a aplicação de princípios físicos fundamentais, destacando-se as leis de Newton para a modelagem de sistemas mecânicos e as leis de Kirchhoff para a análise de circuitos elétricos. A construção de modelos matemáticos adequados constitui uma etapa crítica em todo o processo de análise de sistemas de controle, pois é a base para o projeto e implementação de estratégias de controle eficazes. (Ogata, 2014).

Como citado anteriormente, os modelos matemáticos podem assumir diferentes formas, sendo a escolha do tipo mais adequado determinada pelas características do sistema em estudo e pelos objetivos da análise. Em problemas de controle ótimo, por exemplo, a representação por espaço de estados mostra-se particularmente vantajosa, enquanto para a

análise da resposta transitória ou em frequência de sistemas lineares invariantes no tempo com entrada e saída únicas, a representação por função de transferência geralmente oferece maior conveniência.

Um aspecto crítico no desenvolvimento de modelos matemáticos, apontado por Ogata (2014), é o equilíbrio necessário entre simplicidade e precisão. Na prática, a obtenção de modelos simplificados frequentemente requer a negligência de certas propriedades físicas do sistema real. Essa aproximação é particularmente evidente na modelagem linear de parâmetros concentrados, onde não-linearidades e efeitos de parâmetros distribuídos são comumente desconsiderados. Quando tais simplificações produzem erros aceitáveis nos resultados, o modelo mantém sua utilidade para fins de análise e projeto.

Um aspecto fundamental no estudo de sistemas de controle é a compreensão dos sistemas lineares. Um sistema é considerado linear quando obedece ao princípio da superposição, que estabelece que a resposta do sistema a múltiplas entradas simultâneas equivale à soma das respostas individuais a cada entrada isoladamente. Esta propriedade é de extrema importância prática, pois permite a análise de sistemas complexos por meio da decomposição em problemas mais simples. A linearidade de um sistema pode ser verificada experimentalmente através da relação de proporcionalidade entre causa e efeito. Quando esta relação proporcional é observada, confirmando a validade do princípio da superposição, o sistema pode ser classificado como linear (Ogata, 2014).

Na modelagem de sistemas dinâmicos, duas representações matemáticas destacam-se como fundamentais: as funções de transferência e a formulação no espaço de estados. Conforme estabelecido na literatura especializada (Ogata, 2014; Esfandiari, 2017), a escolha entre estas representações é determinada pela natureza do sistema analisado e pelos objetivos do projeto de controle. Enquanto as funções de transferência são particularmente eficazes para sistemas lineares invariantes no tempo com entrada e saída únicas (SISO), a representação por espaço de estados mostra-se mais versátil para sistemas multivariáveis (MIMO) e não-lineares. Ambas abordagens, quando adequadamente aplicadas, permitem ao engenheiro de controle analisar a estabilidade, projetar compensadores e prever o comportamento dinâmico de sistemas físicos.

Como fundamentado por Esfandiari (2017), as funções de transferência representam uma poderosa ferramenta matemática para análise de sistemas lineares invariantes no tempo (LIT), permitindo caracterizar completamente a relação entrada-saída no domínio da frequência. Esta representação algébrica, obtida através da transformada de Laplace das equações diferenciais ordinárias que descrevem o sistema (Chapra; Canale, 2016), oferece vantagens computacionais significativas, particularmente na análise da estabilidade e resposta em frequência de sistemas dinâmicos. Ogata (2014) ressalta que a função de transferência sintetiza todas as propriedades essenciais do sistema físico, servindo como base para o projeto de controladores no domínio da frequência, como os métodos de compensação por avanço/atraso de fase.

A utilidade das funções de transferência estende-se a diversas aplicações na Engenharia de Controle e Automação, desde o controle de motores elétricos até sistemas mecatrônicos complexos (Ogata, 2014). Contudo, Chapra e Canale (2016) alerta que esta abordagem apresenta limitações intrínsecas, sendo aplicável apenas a sistemas lineares e invariantes no tempo, o que demanda frequentemente a linearização de sistemas não-lineares em torno de pontos de operação. Outra importante limitação do uso de funções de transferência é o fato de que estas não podem ser diretamente empregadas para utilização em computadores digitais.

Apesar dessas restrições, a simplicidade de manipulação algébrica e a interpretação física direta dos polos e zeros - que determinam completamente o comportamento dinâmico do sistema (Esfandiari, 2017) - estabelecem as funções de transferência como elemento essencial na metodologia de análise de sistemas de controle.

A representação no espaço de estados, conforme Ogata (2014), oferece uma estrutura matemática poderosa para a análise e projeto de sistemas de controle. Diferentemente das funções de transferência, que são limitadas a sistemas lineares e invariantes no tempo com entrada e saída únicas, a abordagem por espaço de estados permite:

- Descrição completa da dinâmica interna do sistema através de variáveis de estado;
- Análise de sistemas multivariáveis (MIMO) e não-lineares;
- Implementação eficiente de técnicas modernas como controle ótimo e filtragem.

Sua formulação matricial, baseada em equações diferenciais de primeira ordem, é particularmente vantajosa para implementação computacional e simulação de sistemas complexos.

A modelagem no espaço de estados e a análise de sistemas dinâmicos ganham eficiência e precisão com o uso de *softwares* especializados, que se tornaram ferramentas indispensáveis no ensino e na prática da Engenharia de Controle e Automação. Plataformas como MATLAB (MathWorks, 2023b), GNU Octave (Octave Community, 2023) e Scilab (Scilab Enterprises, 2023) oferecem ambientes integrados para implementação de modelos matemáticos, simulação de sistemas e projeto de controladores, permitindo que estudantes e profissionais validem teorias complexas por meio de experimentos virtuais.

Esses *softwares* facilitam a manipulação de representações matriciais típicas do espaço de estados, a solução numérica de equações diferenciais e a visualização de respostas temporais e frequenciais. O MATLAB, com sua *Control System Toolbox* (MathWorks, 2023a), e o GNU Octave, com seu *Control Package* (Octave Contributors, 2023), como alternativa *open-source*, são particularmente eficazes para:

- Simular sistemas lineares e não-lineares;
- Projetar controladores por métodos clássicos e modernos (ex.: PID, LQR e

realimentação de estados);

- Analisar estabilidade, observabilidade e controlabilidade.

Além disso, a capacidade dessas ferramentas de integrar-se com *hardware* (como microcontroladores) em laboratórios didáticos reforça seu papel na transição entre teoria e prática, essencial para a formação em Engenharia de Controle e Automação.

No contexto educacional, o ensino de Sistemas de Controle que envolvem cálculos complexos e análise de sistemas dinâmicos, requer o uso de ferramentas computacionais capazes de simular e visualizar o comportamento de sistemas sob diferentes condições e utilizando diferentes métodos numéricos. Tradicionalmente, uma abordagem possível seria o desenvolvimento de aplicações dedicadas em linguagens como Pascal, C++ ou Java. No entanto, essa solução exige conhecimentos avançados em programação, o que pode desviar o foco pedagógico dos cursos de graduação, cujo objetivo principal é o domínio dos conceitos teóricos e sua aplicação prática (Burgos; Adam, 2020).

Nos últimos anos, conforme apontam Burgos e Adam (2020), uma solução amplamente adotada tem sido o uso de ambientes computacionais baseados em linguagens interpretadas de alto nível, como MATLAB, GNU Octave, SciLab e Spyder IDE para Python. Essas plataformas oferecem duas vantagens principais:

- Bibliotecas especializadas que implementam algoritmos prontos para aplicações em engenharia;
- Flexibilidade de uso, permitindo desde operações interativas simples até o desenvolvimento de aplicações mais complexas.

Como destacado no estudo, essas ferramentas funcionam tanto no modo interativo (comando a comando) quanto por meio da execução de *scripts*, tornando-as ideais para fins didáticos (Burgos; Adam, 2020). O GNU Octave, em particular, tem se destacado como alternativa *open-source* ao MATLAB, especialmente após o desenvolvimento de interfaces gráficas amigáveis que facilitam seu uso em sala de aula.

A implementação de interfaces gráficas especializadas tem se mostrado fundamental para viabilizar o ensino de ferramentas avançadas de projeto de controle. Como demonstrado por Burgos e Adam (2020) em sua ferramenta LTITool para GNU Octave, ambientes interativos que combinam visualização intuitiva com capacidades analíticas permitem que estudantes e profissionais explorem conceitos complexos sem a necessidade de domínio profundo de programação. A LTITool, especificamente, oferece funcionalidades comparáveis às do Control System Designer do MATLAB (MathWorks, 2023a), permitindo análise e projeto de controladores no domínio da frequência através de uma interface gráfica unificada.

O desenvolvimento de GUIs (do inglês, *Graphical User Interfaces*) eficazes para aplicações de engenharia deve equilibrar funcionalidade técnica e usabilidade, conforme

destacam Burgos e Adam (2020). Ferramentas como o Control System Designer do MATLAB estabeleceram padrões importantes nessa área, oferecendo:

- Visualização interativa de diagramas de Bode, Nyquist e Nichols;
- Ajuste gráfico de compensadores por arrasto-e-solta;
- Análise integrada de margens de estabilidade.

Esses recursos, quando implementados em plataformas *open-source* como a LTI-TOOL para Octave, ampliam significativamente o acesso a metodologias avançadas de projeto de controle. A experiência prática com essas interfaces não apenas facilita o entendimento conceitual, como também prepara os estudantes para utilizar ferramentas profissionais na indústria, fechando a lacuna entre teoria e prática na formação em Engenharia de Controle e Automação.

Explorando as necessidades didáticas para o ensino da Engenharia notou-se que seria importante o desenvolvimento de uma ferramenta de auxílio para a implementação de técnicas de simulação de sistemas.

A implementação de métodos numéricos, como os métodos de Runge-Kutta, demanda não apenas rigor matemático, mas também eficiência computacional. *Softwares* proprietários como MATLAB, embora poderosos, apresentam custos elevados de licenciamento, tornando-se inacessíveis para muitas instituições de ensino e empresas (Moler, 2010). Essa limitação financeira pode restringir o acesso a ferramentas essenciais para o estudo de sistemas dinâmicos, especialmente em cursos de Engenharia, onde a simulação numérica é fundamental. Além disso, a dependência de *softwares* fechados pode criar barreiras à reprodução e validação de resultados, contrariando os princípios de ciência aberta e colaborativa (Pérez; Granger; Hunter, 2011).

Alternativas *open-source*, como o GNU Octave, surgem como soluções viáveis, oferecendo funcionalidades equivalentes ao MATLAB sem custos de licença (Eaton; Bateman; Hauberg, 2008). No contexto deste trabalho, a escolha pelo Octave justifica-se não apenas pela economia de recursos, mas também pela compatibilidade de sintaxe, que permite a migração de códigos existentes com mínimas adaptações. Essa flexibilidade é crítica para o desenvolvimento de *softwares* didáticos, como o proposto neste TCC, que integra métodos numéricos clássicos e avançados (Euler, Runge-Kutta) em uma interface gráfica acessível. A ferramenta desenvolvida permite que estudantes manipulem matrizes de espaço de estados e comparem visualmente os resultados de diferentes métodos, eliminando a necessidade de plataformas caras.

A importância do *software* desenvolvido reside em sua capacidade de suprir uma lacuna educacional: a carência de recursos didáticos gratuitos que combinem teoria e prática de forma interativa. Segundo Butcher (2008), a compreensão de métodos numéricos é significativamente ampliada quando os alunos podem visualizar erros de truncamento, estabilidade e convergência em tempo real. A interface proposta neste trabalho não ape-

nas reproduz funcionalidades de *software* comerciais, mas também é projetada para fins pedagógicos, com uma arquitetura de fácil interação que guia o usuário passo a passo nas simulações. Sua organização intuitiva permite que estudantes configurem rapidamente os parâmetros das matrizes de estado, selecionem métodos numéricos e visualizem os resultados de forma clara e imediata. Isso é particularmente relevante em disciplinas como Modelagem e Simulação de Processos, onde a curva de aprendizado é íngreme e a abstração matemática pode ser um obstáculo.

Além da acessibilidade, o uso de ferramentas *open-source* promove a reprodutibilidade científica, um pilar da pesquisa moderna (Peng, 2011). O código desenvolvido em Octave pode ser livremente compartilhado, modificado e adaptado para outras aplicações, como o estudo de sistemas não lineares ou a implementação de controladores. Do ponto de vista computacional, a otimização de algoritmos em Octave — como a vetorização de operações e o uso eficiente de estruturas de dados — garante desempenho comparável ao de soluções proprietárias para problemas de pequena e média escala (Eaton; Bateman; Hauberg, 2008). Isso é especialmente válido para métodos de Runge-Kutta de ordem inferior (RK1, RK2, RK3) e superior (RK4), cuja implementação eficiente depende mais da qualidade do algoritmo do que do ambiente de *software*.

O desenvolvimento deste *software* didático em GNU Octave justifica-se por três pilares fundamentais: acessibilidade, capacidade pedagógica e eficiência computacional. Ao substituir ferramentas caras por soluções *open-source*, o trabalho facilita o acesso ao ensino de métodos numéricos, alinhando-se à um cenário de recursos financeiros limitados. Adicionalmente, a interface gráfica e a integração de múltiplos métodos numéricos oferecem um ambiente de aprendizado prático, onde os alunos podem explorar os equilíbrios entre precisão, estabilidade e custo computacional. Como demonstrado nos resultados, a ferramenta não apenas replica funcionalidades de *software* comerciais, mas também as adapta para fins educacionais, reforçando seu valor como recurso didático inovador e inclusivo.

1.1 OBJETIVOS

A seguir, apresentam-se o objetivo geral e os objetivos específicos deste TCC, bem como a estrutura organizacional adotada no trabalho.

1.1.1 Objetivo Geral

O presente trabalho tem como principal objetivo desenvolver uma ferramenta computacional *open-source* em GNU Octave que integre métodos numéricos clássicos e avançados para solução de sistemas dinâmicos lineares e invariantes no tempo (LIT) modelados no espaço de estados, proporcionando uma interface gráfica intuitiva para análise comparativa e didática.

1.1.2 Objetivos Específicos

A implementação de métodos numéricos para sistemas dinâmicos exige não apenas precisão computacional, mas também requer uma interface gráfica que possua um *design* intuitivo para aplicações didáticas. Para atender a esses requisitos, este trabalho estrutura-se nos seguintes objetivos específicos:

1. Implementar os métodos numéricos clássicos em funções computacionais (Euler - também denominado de RK1, Runge-Kutta de 2ª e 4ª ordens – RK2 e RK4), capazes de resolver sistemas dinâmicos modelados no Espaço de Estados;
2. Implementar um método numérico adicional (Runge-Kutta de 3ª ordem – RK3) para ampliar as opções de análise comparativa;
3. Desenvolver uma interface gráfica (GUI) no Octave com as seguintes funcionalidades:
 - Seleção do método numérico via lista suspensa (*drop-down*);
 - Escolha de entradas típicas (degrau, senoidal, impulso, exponencial);
 - Visualização gráfica dos resultados (resposta temporal, estados do sistema).
4. Gerar saídas numéricas e gráficas para avaliação do desempenho dos métodos:
 - Tabelas comparativas de entradas, saídas, estados e erros do sistema;
 - Diagramas de análise dinâmica (Bode, *Nyquist*, Polos e Zeros);
 - Representação da função de transferência equivalente.
5. Validar a ferramenta com exercícios didáticos da literatura para a disciplina de Modelagem e Simulação de Processos, demonstrando:
 - Precisão dos métodos implementados;
 - Facilidade de uso para fins educacionais.

1.2 ESTRUTURA DO TRABALHO

O trabalho encontra-se dividido em 5 capítulos. O primeiro capítulo, de introdução, apresenta a motivação e a problemática do projeto, com ênfase na necessidade de ferramentas computacionais para o estudo de sistemas dinâmicos na engenharia de controle e automação. O segundo capítulo contempla o referencial teórico, abordando os fundamentos matemáticos dos métodos numéricos implementados (Euler, Runge-Kutta de segunda, terceira e quarta ordens). O terceiro capítulo descreve a metodologia empregada no desenvolvimento do *software*, detalhando a implementação da ferramenta computacional em GNU Octave, apresentando o projeto e as funcionalidades da interface gráfica. No quarto capítulo, os resultados obtidos são descritos, demonstrando a aplicação prática da

ferramenta através de testes com sistemas dinâmicos exemplares e análise comparativa dos métodos numéricos implementados. Por fim, o quinto capítulo traz as conclusões do trabalho, evidenciando suas contribuições, apontando limitações identificadas e sugerindo direções para desenvolvimentos futuros.

2 REFERENCIAL TEÓRICO

2.1 MODELAGEM MATEMÁTICA

Segundo a fundamentação teórica apresentada por Chapra e Canale (2016) e Ogata (2014), um modelo matemático consiste em uma representação formal de um sistema físico por meio de equações matemáticas, permitindo analisar seu comportamento e prever sua evolução temporal. Como apresentado por Chapra e Canale (2016), esse processo envolve a identificação das variáveis relevantes e das relações entre elas, conforme a estrutura geral (1):

$$\begin{array}{c} \text{Derivada da} \\ \text{Variável dependente} \end{array} = f \left(\begin{array}{c} \text{Variáveis} \\ \text{independentes,} \end{array} \begin{array}{c} \text{termos} \\ \text{parâmetros,} \end{array} \begin{array}{c} \\ \text{forçantes} \end{array} \right) \quad (1)$$

Onde:

- **Variável dependente:** Grandeza que descreve o estado do sistema (ex.: aceleração, temperatura);
- **Variáveis independentes:** Dimensões como tempo (t) ou espaço (x, y, z);
- **Parâmetros:** Propriedades intrínsecas do sistema (ex.: massa, constante elástica);
- **Termos forçantes:** Influências externas (ex.: forças aplicadas).

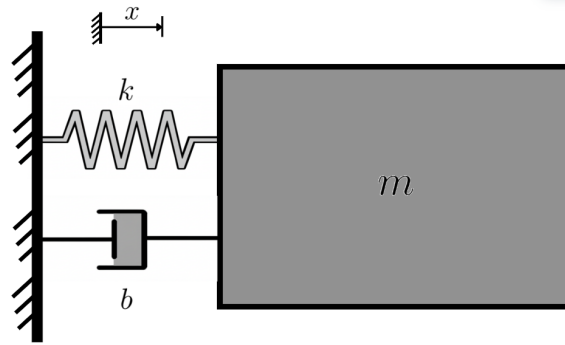
Essa abordagem teórica se manifesta concretamente na análise de sistemas dinâmicos mecânicos, onde as interações entre forças, inércia e propriedades dos materiais podem ser precisamente quantificadas. Particularmente em sistemas vibratórios, a combinação de elementos armazenadores (molas) e dissipadores de energia (amortecedores) com corpos inertes (massas) constitui um exemplo ideal para demonstrar a construção e solução de modelos matemáticos em engenharia. Como caso emblemático, o sistema massa-mola amortecido, ilustrado na Figura 1, engloba todas as componentes da relação fundamental (1), permitindo derivar tanto soluções analíticas quanto numéricas para seu comportamento dinâmico.

2.1.1 Exemplo: Sistema massa-mola amortecido

Considere o sistema mecânico ilustrado na Figura 1, composto por:

- Um corpo de massa m que desliza sobre uma superfície horizontal sem atrito;
- Uma mola linear com constante elástica k ;
- Um amortecedor viscoso com coeficiente c ;
- Um deslocamento com coeficiente x ;

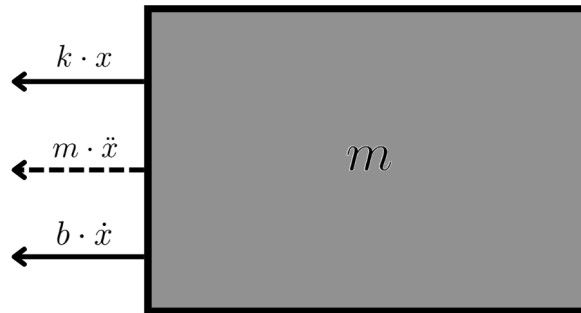
Figura 1 – Sistema massa-mola-amortecedor.



Fonte: Elaborado pela autora (2025).

Antes de derivar as equações do movimento, é essencial analisar as forças atuantes através de um diagrama de corpo livre (DCL), ferramenta fundamental da mecânica newtoniana que isola o corpo de interesse e representa todas as forças externas atuantes sobre ele. Na Figura 2, apresenta-se o DCL para a massa m

Figura 2 – Diagrama de Corpo Livre.



Fonte: Elaborado pela autora (2025).

onde,

- **Força restauradora da mola (Lei de Hooke):** Proporcional ao deslocamento x e oposta ao movimento (2):

$$F_k = -k \cdot x \quad (2)$$

- **Força viscosa do amortecedor ($\dot{x} \equiv \frac{dx}{dt}$):** Proporcional à velocidade v e oposta ao movimento (3):

$$F_b = -b \cdot \dot{x} \quad (3)$$

- **Força de D'Alembert:** Também denominada de força inercial, trata-se de uma força fictícia que tem como objetivo estabelecer o equilíbrio no somatório

das forças externas. Se caracteriza por ser uma força de oposição à aceleração a do corpo (4):

$$F_i = m \cdot \ddot{x} \quad (4)$$

Aplicando o Princípio de D'Alembert, que estende a segunda lei de Newton para sistemas dinâmicos através do conceito de força de inércia ($-m\ddot{x}$), estabelece-se o equilíbrio dinâmico (5) (Close; Frederick; Newell, 2001):

$$\sum_i (f_{ext})_i + \underbrace{(-m\ddot{x})}_{\text{Força de inércia}} = 0 \quad (5)$$

Para o sistema em análise, isto se traduz em (6):

$$-F_k - F_b - m\ddot{x} = 0 \quad (6)$$

Substituindo as expressões das forças, recupera-se a equação diferencial do movimento, obtendo-se a Equação (7):

$$m\ddot{x} + b\dot{x} + kx = 0 \quad (7)$$

Esta combinação de métodos - análise newtoniana via DCL complementada pelo princípio de D'Alembert - oferece uma compreensão física mais intuitiva da dinâmica do sistema, evidenciando como:

- A força da mola atua como termo restaurador ($\propto x$);
- O amortecedor introduz dissipação de energia ($\propto \dot{x}$);
- A inércia se manifesta através da aceleração ($\propto \ddot{x}$).

Cabe destacar que a literatura especializada apresenta variações na notação do coeficiente de amortecimento viscoso: enquanto obras como Ogata (2014) adotam a constante b , outras referências, incluindo Chapra e Canale (2016), utilizam a notação c para representar o mesmo parâmetro físico. Essa dualidade de notações não implica diferenças conceituais, sendo meramente uma questão de convenção bibliográfica. Esta equação apresenta como variáveis e parâmetros:

- **Variável dependente:** Posição $x(t)$ [m];
- **Variável independente:** Tempo t [s];
- **Parâmetros:** Massa m [kg], coeficiente de amortecimento c [N·s/m], constante elástica k [N/m].

2.1.1.1 Equações Diferenciais

De acordo com o exposto em Chapra e Canale (2016), as Equações Diferenciais podem ser descritas, como equações matemáticas que envolvem uma função desconhecida,

usualmente denotada por $y(t)$, caso a variável independente denote o tempo t , e suas derivadas. Tais equações também descrevem a relação entre uma função e suas taxas de variação. De forma geral, uma equação diferencial pode ser escrita pela Equação (8)

$$F\left(y, \frac{dy}{dt}, \frac{d^2y}{dt^2}, \dots, \frac{d^ny}{dt^n}\right) = 0 \quad (8)$$

onde:

- $y(t)$ é a função desconhecida (variável dependente);
- t é a variável independente;
- $\frac{d^ny}{dt^n}$ representa a n -ésima derivada de y em relação a t .

As Equações Diferenciais Ordinárias (EDOs) representam ferramentas fundamentais para a modelagem matemática de sistemas físicos na Engenharia, sendo particularmente adequadas para descrever fenômenos com taxas de variação temporal. Suas principais características consistem em:

- Envolverem apenas uma variável independente (ex: tempo t ou posição x);
- Relacionarem uma função e suas derivadas em relação a essa única variável.

É importante ressaltar que as equações diferenciais são classificadas quanto à sua ordem. Matematicamente, uma equação diferencial é de ordem n quando a n -ésima derivada da função desconhecida $y(t)$ corresponde à derivada de maior ordem presente na equação, como pode ser observado na Equação (8).

Em suma, as EDOs são particularmente úteis porque modelam sistemas de parâmetros concentrados, onde propriedades são uniformes no espaço (como massas pontuais ou circuitos elétricos com parâmetros globais), admitem técnicas analíticas robustas — como separação de variáveis e transformadas de Laplace — e servem de base para métodos numéricos clássicos, incluindo algoritmos do tipo Euler e Runge-Kutta.

2.1.2 Solução analítica

A solução analítica para equações diferenciais do tipo massa-mola amortecido é amplamente discutida na literatura de vibrações mecânicas e sistemas dinâmicos. Conforme apresentado por Inman e Singh (2014) e Rao (2011), a solução geral para o caso subamortecido pode ser derivada através de métodos de resolução de equações diferenciais ordinárias lineares de segunda ordem com coeficientes constantes. Para o sistema descrito pela Equação (7), quando satisfeita a condição $c^2 < 4mk$, obtém-se a solução (9):

$$x(t) = e^{-\zeta\omega_n t} [A \cos(\omega_d t) + B \sin(\omega_d t)] \quad (9)$$

Onde:

- Fator de amortecimento: $\zeta = \frac{c}{2\sqrt{mk}}$ [adimensional];
- Frequência natural: $\omega_n = \sqrt{\frac{k}{m}}$ $\left[\frac{rad}{s}\right]$;
- Frequência natural amortecida: $\omega_d = \omega_n\sqrt{1 - \zeta^2}$ $\left[\frac{rad}{s}\right]$;
- A e B são constantes determinadas pelas condições iniciais.

Esta solução exata descreve o comportamento oscilatório amortecido do sistema, com amplitude decaindo exponencialmente. Quando não é possível obter soluções analíticas como esta, por exemplo para caso de sistemas não-lineares, ou quando a solução se torna muito onerosa, recorre-se a métodos numéricos para aproximar o comportamento do sistema.

2.2 MODELAGEM NO ESPAÇO DE ESTADOS

A modelagem no espaço de estados, conforme descrito por Ogata (2014), é uma representação matemática compacta e poderosa de sistemas dinâmicos, que expressa as relações entre entradas, saídas e variáveis internas (estados) através de equações diferenciais de primeira ordem. Sua importância reside na capacidade de:

1. **Generalização:** Representar sistemas lineares e não-lineares, contínuos ou discretos, em uma forma unificada;
2. **Análise:** Facilitar o estudo de estabilidade, controlabilidade e observabilidade;
3. **Implementação Computacional:** Simplificar a discretização e integração numérica, especialmente para métodos como Euler e Runge-Kutta.

O estado de um sistema dinâmico representa o menor conjunto de variáveis (denominadas variáveis de estado) cujo conhecimento em um instante inicial t_0 , combinado com as entradas para $t \geq t_0$, determina completamente o comportamento futuro do sistema. Essa definição aplica-se não apenas a sistemas físicos, mas também a sistemas biológicos, econômicos e sociais.

As variáveis de estado constituem um conjunto mínimo de variáveis (como $x_1, x_2, \dots, x_n$) capazes de descrever totalmente a dinâmica do sistema, mesmo que não sejam diretamente mensuráveis. Essa flexibilidade na escolha das variáveis é uma vantagem da abordagem em espaço de estados, embora, na prática, prefira-se variáveis mensuráveis para facilitar a implementação de estratégias de controle.

Quando organizadas em um vetor x , essas variáveis formam o vetor de estado, que determina de modo único a evolução temporal do sistema. O espaço de estados é o ambiente n -dimensional cujos eixos são definidos pelas variáveis de estado, permitindo representar graficamente qualquer condição do sistema como um ponto nesse espaço (Ogata, 2014).

Considerando um sistema dinâmico com múltiplas entradas e múltiplas saídas (MIMO), composto por n estados. Sejam r entradas $u_1(t), u_2(t), \dots, u_r(t)$ e m saídas

$y_1(t), y_2(t), \dots, y_m(t)$. As variáveis de estado são $x_1(t), x_2(t), \dots, x_n(t)$ e representam o sistema na forma (10):

$$\begin{aligned}\dot{x}_1(t) &= f_1(x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_r; t) \\ \dot{x}_2(t) &= f_2(x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_r; t) \\ &\vdots \\ \dot{x}_n(t) &= f_n(x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_r; t)\end{aligned}\tag{10}$$

As saídas do sistema são descritas pelas relações (11):

$$\begin{aligned}y_1(t) &= g_1(x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_r; t) \\ y_2(t) &= g_2(x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_r; t) \\ &\vdots \\ y_m(t) &= g_m(x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_r; t)\end{aligned}\tag{11}$$

Definindo os vetores de estado $x(t)$, entrada $u(t)$ e saída $y(t)$ como (12):

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix}, \quad \mathbf{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_r(t) \end{bmatrix}, \quad \mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_m(t) \end{bmatrix}\tag{12}$$

E as funções vetoriais f e g como (13) e (14):

$$\mathbf{f}(\mathbf{x}, \mathbf{u}, t) = \begin{bmatrix} f_1(x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_r; t) \\ f_2(x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_r; t) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_r; t) \end{bmatrix}\tag{13}$$

$$\mathbf{g}(\mathbf{x}, \mathbf{u}, t) = \begin{bmatrix} g_1(x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_r; t) \\ g_2(x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_r; t) \\ \vdots \\ g_m(x_1, x_2, \dots, x_n; u_1, u_2, \dots, u_r; t) \end{bmatrix}\tag{14}$$

As equações (10) e (11) tornam-se (15):

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \quad (\text{Equação de Estado}) \\ \mathbf{y}(t) &= \mathbf{g}(\mathbf{x}, \mathbf{u}, t) \quad (\text{Equação de Saída})\end{aligned}\tag{15}$$

A equação $\dot{\mathbf{x}}(t)$ descreve a dinâmica do sistema, enquanto a equação $\mathbf{y}(t)$ define sua relação de saída. Quando as equações do sistema (15) são lineares e invariantes no tempo, obtém-se a representação matricial (16):

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) & (\text{Equação de Estado}) \\ \mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) & (\text{Equação de Saída}) \end{cases}\tag{16}$$

Onde:

- $\mathbf{x}(t)$: Vetor de estados;
- $\mathbf{u}(t)$: Vetor de entradas;
- $\mathbf{y}(t)$: Vetor de Saídas;
- $\mathbf{A}$: Matriz de estado;
- $\mathbf{B}$: Matriz de entrada;
- $\mathbf{C}$: Matriz de saída;
- $\mathbf{D}$: Matriz de transmissão direta;

No âmbito deste trabalho, a implementação computacional dos métodos numéricos será estruturada mediante a abordagem sistêmica no espaço de estados, pois ela proporciona uma representação matricial robusta para os sistemas dinâmicos. Esta formulação permite expressar o comportamento do sistema através de equações diferenciais ordinárias (EDOs) no formato da Equação (16) para sistemas lineares e invariantes no tempo (LIT).

2.3 MÉTODOS NUMÉRICOS

A modelagem matemática de fenômenos físicos frequentemente esbarra em uma limitação fundamental: muitos sistemas de interesse prático não admitem soluções analíticas exatas ou a solução analítica é impraticável devido à complexidade do problema. Como destacado por Chapra e Canale (2016) e Ogata (2014), essa realidade consolida os métodos numéricos como instrumentos essenciais para a Engenharia.

Nesse contexto, as técnicas numéricas emergem como solução viável, substituindo as operações de cálculo infinitesimal por aproximações discretas que apresentam bons resultados. Como demonstrado na Seção 2.1 através da Equação (7), a formulação diferencial completa pode exigir métodos matemáticos sofisticados para obtenção de soluções analíticas, conforme exemplificado pela Equação (9). Essa complexidade evidencia a necessidade de abordagens alternativas quando nos deparamos com as limitações analíticas,

soluções exatas podem ser inexistentes, como no caso de sistemas não-lineares ou computacionalmente inviáveis, devido à escala do problema.

2.3.1 Métodos Numéricos para Resolução de Sistemas Lineares

De acordo com Esfandiari (2017), uma equação diferencial de ordem n está associada a n condições auxiliares necessárias para determinar as constantes de integração da solução geral. Quando essas condições são especificadas no mesmo valor inicial da variável independente, tem-se um problema de valor inicial (PVI).

Considerando um PVI de primeira ordem na forma padrão (17):

$$\dot{y} = f(x, y), \quad y(a) = y_0, \quad x_0 = a \leq x \leq b = x_n \quad (17)$$

Onde:

- y_0 é a condição inicial conhecida;
- x é a variável independente, definida no intervalo $[a, b]$;
- $f(x, y)$ é uma função que descreve a taxa de variação de y ;
- Assume-se que existe uma solução única $y(x)$ no intervalo considerado.

Para resolver esse problema numericamente, o intervalo $[a, b]$ é discretizado em n subintervalos de tamanho h (tamanho do passo), gerando os pontos (18):

$$x_0, \quad x_1 = x_0 + h, \quad x_2 = x_0 + 2h, \quad \dots, \quad x_n = x_0 + nh \quad (18)$$

O objetivo fundamental consiste em estimar os valores da solução $y(x)$ nos pontos discretos $x_1, x_2, \dots, x_n$, partindo da condição inicial conhecida $y(x_0) = y_0$.

Os métodos numéricos de um passo constituem a abordagem mais elementar para resolução de PVIs em sistemas lineares. Como apresentado por Esfandiari (2017), estes métodos caracterizam-se por calcular cada estimativa y_{i+1} no ponto x_{i+1} utilizando exclusivamente informação do ponto imediatamente anterior (x_i, y_i) . A formulação geral desses métodos segue a estrutura (19):

$$y_{i+1} = y_i + h\varphi, \quad i = 0, 1, 2, \dots, n-1 \quad (19)$$

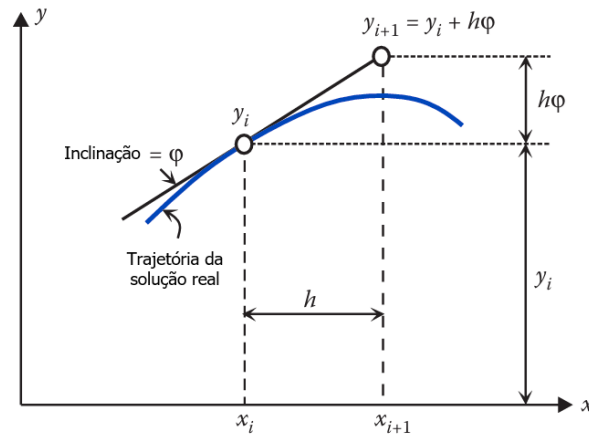
Esta estrutura representa uma reta entre dois valores consecutivos de y , a variável φ representa a inclinação da reta. A forma geral presente na Equação (19) descreve todos os métodos de um passo, porém, cada método específico emprega uma abordagem particular para estimar a inclinação φ . A estimativa da inclinação de φ é utilizada para projetar

linearmente a solução do valor conhecido y_i para o novo valor y_{i+1} , avançando em um passo de tamanho h ao longo do eixo independente, conforme pode ser observado pela Figura 3.

Esta formulação recursiva permite:

1. A propagação sequencial da solução;
2. A construção ponto a ponto da trajetória numérica;
3. A extrapolação controlada do comportamento da EDO.

Figura 3 – Descrição gráfica de um método de um passo.



Fonte: Adaptado de (Esfandiari, 2017).

A fundamentação teórica desses métodos de um passo reside na substituição da derivada contínua $\dot{y}$ presente na EDO por uma aproximação discreta baseada em diferenças finitas. Como destacam Richard L. Burden, Faires e Annette M. Burden (2016), essa substituição é o princípio fundamental que permite transformar um problema diferencial contínuo, como o definido em Equação (17), em um problema algébrico discreto, passível de ser resolvido iterativamente. Diferentes formulações para a aproximação da derivada (e, conseqüentemente, para o cálculo da inclinação φ) dão origem aos diferentes métodos, como os de Euler, Heun e Runge-Kutta, que variam em complexidade e precisão.

2.4 APROXIMAÇÃO DA DERIVADA

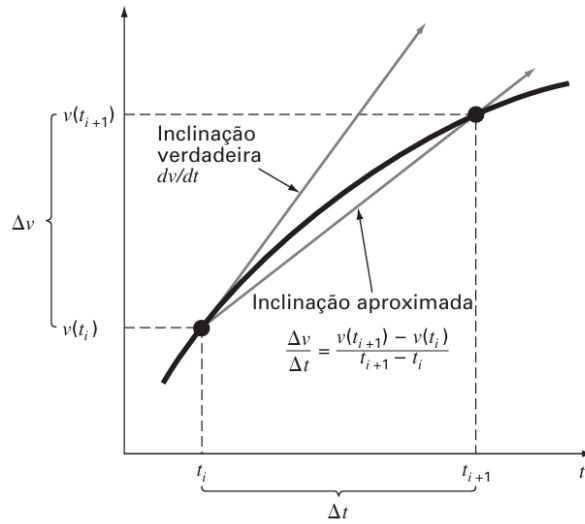
Como discutido anteriormente, os métodos numéricos consistem em reformular problemas matemáticos complexos de modo que possam ser resolvidos através de operações aritméticas básicas. Essa abordagem pode ser exemplificada pela aplicação à Segunda Lei de *Newton*, onde a taxa de variação temporal da velocidade ($\frac{dv}{dt}$) é aproximada pela Equação (20) e verificada através da Figura 4 (Chapra; Canale, 2016).

$$\frac{dv}{dt} \cong \frac{\Delta v}{\Delta t} = \frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i} \quad (20)$$

Nessa formulação:

- Δv e Δt representam incrementos finitos de velocidade e tempo, respectivamente;
- $v(t_i)$ denota a velocidade no instante inicial t_i ;
- $v(t_{i+1})$ corresponde à velocidade em um instante posterior t_{i+1} .

Figura 4 – Aproximação numérica da derivada de v com relação a t por diferença finita.



Fonte: (Chapra; Canale, 2016).

É crucial observar que a relação $\frac{dv}{dt} \cong \Delta v / \Delta t$ constitui uma aproximação, pois considera intervalos de tempo discretos (Δt finito) em vez de uma variação infinitesimal. Essa diferença fundamental remete aos conceitos de cálculo diferencial, onde a derivada verdadeira é definida no limite quando $\Delta t \rightarrow 0$ (Equação (21)).

$$\frac{dv}{dt} = \lim_{\Delta t \rightarrow 0} \frac{\Delta v}{\Delta t} \quad (21)$$

Segundo Chapra e Canale (2016), a Equação (20) descreve o processo inverso ao abordado anteriormente. Trata-se de uma aproximação por diferenças finitas para a derivada no instante t_i , que pode ser expressa como (22):

$$\frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i} \approx \left. \frac{dv}{dt} \right|_{t_i} \quad (22)$$

A aplicação prática da aproximação numérica da derivada pode ser ilustrada através de um exemplo clássico de dinâmica de corpos em queda livre: o modelo de um paraquedista, conforme apresentado por Chapra e Canale (2016). A equação diferencial que descreve o movimento de um paraquedista é dada por (23):

$$\frac{dv}{dt} = g - \frac{c}{m}v \quad (23)$$

Onde, m é a massa do paraquedista $[kg]$, g é a aceleração da gravidade $[m/s^2]$, v é a velocidade $[m/s]$ e c é o coeficiente de arrasto $[kg/s]$. Ao inserir essa aproximação (22) na equação que governa a dinâmica de um paraquedista (23), obtém-se (24):

$$\frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i} = g - \frac{c}{m}v(t_i) \quad (24)$$

Onde:

- $v(t_i)$: Velocidade no instante t_i ;
- $v(t_{i+1})$: Velocidade no instante subsequente t_{i+1} ;
- $\Delta t = t_{i+1} - t_i$: Intervalo de tempo discreto.

Isolando $v(t_{i+1})$, a fórmula iterativa Equação (25) é obtida:

$$v(t_{i+1}) = v(t_i) + \left[g - \frac{c}{m}v(t_i) \right] (t_{i+1} - t_i) \quad (25)$$

Na Equação (23) é possível observar, que a equação fornece uma maneira de calcular a taxa de variação (ou a inclinação) da velocidade v . Dessa forma, a equação diferencial é transformada em uma expressão algébrica (Equação (25)) que permite determinar a velocidade no instante (t_{i+1}) com base na inclinação e nos valores anteriores de v e t .

Se um valor inicial para a velocidade em um instante t_i for conhecido, é possível calcular facilmente a velocidade no próximo intervalo t_{i+1} . Esse novo valor pode, então, ser utilizado para estender o cálculo para t_{i+2} , e assim sucessivamente. Portanto, a cada passo da iteração, o valor novo é obtido a partir do valor anterior, da inclinação e do tamanho do passo (Equação (26)) (Chapra; Canale, 2016).

$$\text{Valor novo} = \text{Valor Antigo} + \text{Inclinação} \times \text{Tamanho do passo} \quad (26)$$

Essa fórmula (26) equivale à Equação (19) e constitui a base dos métodos numéricos de solução de equações diferenciais ordinárias, como o Método de Euler, e permite a propagação iterativa dos resultados ao longo do tempo.

2.5 MÉTODO DE EULER EXPLÍCITO

Como demonstra Esfandiari (2017), o método de Euler (RK1) tem sua base teórica na expansão em série de Taylor truncada no termo de primeira ordem. Dada uma EDO como a dada em (17).

Considerando a expansão de $y(x_1)$ em torno de x_0 , obtém-se a expressão (27):

$$y(x_1) = y(x_0 + h) = y(x_0) + hy'(x_0) + \frac{1}{2!}h^2y''(x_0) + \dots \quad (27)$$

Ao desprezar os termos de ordem superior e manter apenas a aproximação linear, a Equação (27) pode ser reescrita como (28):

$$y(x_1) = y(x_0) + hy'(x_0) \quad (28)$$

Generalizando essa abordagem para um passo qualquer, a expansão de $y(x_{i+1})$ em torno de x_i resulta em (29):

$$y(x_{i+1}) = y(x_i) + hy'(x_i) \quad (29)$$

Sabendo que $y'(x_i) = f(x_i, y_i)$ e adotando a notação $y_i = y(x_i)$, a solução aproximada y_{i+1} pode ser calculada através da fórmula de iteração (30):

$$y_{i+1} = y_i + hf(x_i, y_i), \quad i = 0, 1, 2, \dots, n-1 \quad (30)$$

O termo y_{i+1} é a aproximação numérica de $y(x_{i+1})$ e $f(x_i, y_i)$ é a inclinação (taxa de inclinação) no ponto atual. Essa abordagem permite calcular a solução iterativamente, propagando a estimativa inicial ao longo da partição do intervalo. Esta expressão constitui a formulação básica do método de Euler, estabelecendo uma relação recursiva para a estimativa da solução em cada passo de integração.

2.5.1 Análise dos Erros de Truncamento no Método de Euler

A compreensão dos erros associados ao método de Euler é fundamental para sua aplicação eficaz. Essa análise parte da expansão em série de Taylor da solução exata, que nos permite quantificar e classificar os tipos de erros introduzidos pela aproximação numérica. Como demonstra Chapra e Canale (2016), a partir da equação diferencial $y' = f(x, y)$, a expansão de Taylor fornece a Equação (31):

$$y_{i+1} = y_i + f(x_i, y_i)h + \frac{f'(x_i, y_i)}{2!}h^2 + \dots + \frac{f^{(n-1)}(x_i, y_i)}{n!}h^n + O(h^{n+1}) \quad (31)$$

O método de Euler utiliza apenas os dois primeiros termos da expansão anterior, resultando em (32):

$$y_{i+1} = y_i + f(x_i, y_i)h \quad (32)$$

Esses erros podem ser categorizados em dois componentes principais:

1. **Erro de Truncamento Verdadeiro**: Surge em cada iteração individual do método, decorrente da aproximação numérica adotada em um único passo de integração. Como demonstra a Equação (33), esse erro corresponde à diferença entre a solução exata (Equação (32)) e a aproximação numérica (Equação (31)), representando precisamente os termos de ordem superior da série de Taylor que foram desconsiderados na formulação do método (Chapra; Canale, 2016):

$$E_t = \frac{f'(x_i, y_i)}{2!}h^2 + \dots + O(h^{n+1}) \quad (33)$$

2. **Erro de Truncamento Local Aproximado**: Resulta do acúmulo sequencial das aproximações realizadas em todas as etapas anteriores do processo iterativo. Para valores suficientemente pequenos de h , o termo dominante do erro é dado pela Equação (33).

$$E_a = \frac{f'(x_i, y_i)}{2!}h^2 \rightarrow E_a = \mathcal{O}(h^2) \quad (34)$$

A combinação desses dois elementos constitui o erro de truncamento global, que determina a precisão final da solução numérica. A análise dos erros revela que a redução do passo de integração h provoca uma diminuição quadrática no erro local ($\mathcal{O}(h^2)$), enquanto o erro global acumulado diminui linearmente ($\mathcal{O}(h)$).

Esta relação evidencia o compromisso fundamental entre precisão e eficiência computacional na implementação do método de Euler - à medida que se reduz h para obter maior exatidão, aumenta-se proporcionalmente o número de cálculos necessários, demandando uma escolha criteriosa deste parâmetro para otimizar o desempenho do método.

2.5.2 Método de Euler Implícito

O método de Euler Implícito, constitui uma variação fundamental do esquema numérico tradicional para resolução de problemas de valor inicial. Diferentemente da formulação explícita clássica, esta abordagem avalia a função $f(x, y)$ no ponto futuro (x_{i+1}, y_{i+1}) , resultando em uma equação implícita que requer técnicas específicas para sua resolução (Kreyszig, 2020). A formulação básica do método é expressa por (35):

$$y_{i+1} = y_i + hf(x_{i+1}, y_{i+1}) \quad (35)$$

A natureza implícita desta equação demanda processos iterativos para sua solução, especialmente quando aplicada a funções $f(x, y)$ não-lineares. Contudo, para equações diferenciais ordinárias lineares, a solução pode ser obtida diretamente através de manipulação algébrica.

A principal vantagem desta abordagem reside em suas propriedades de estabilidade numérica. Enquanto o método explícito convencional impõe restrições severas ao tamanho do passo h para garantir convergência, a versão implícita apresenta estabilidade incondicional para sistemas lineares estáveis (Kreyszig, 2020). Esta característica torna-o particularmente adequado para a solução de problemas classificados como "rígidos", comuns em diversas áreas da engenharia, incluindo:

- Análise de vibrações mecânicas;
- Modelagem de circuitos elétricos;
- Simulação de reações químicas.

O conceito de rigidez, segundo Kreyszig (2020), refere-se a situações onde a solução numérica exige passos extremamente pequenos para evitar a amplificação descontrolada de erros. Em sistemas rígidos, o método implícito mantém a estabilidade da solução mesmo com incrementos h relativamente grandes, embora com redução progressiva da precisão à medida que h aumenta.

2.6 MÉTODOS DE RUNGE-KUTTA

Os métodos de Runge-Kutta constituem uma classe importante de técnicas numéricas para resolver equações diferenciais ordinárias com condições iniciais. Esses métodos se destacam por sua capacidade de aproximar soluções de forma precisa sem a necessidade de calcular derivadas de ordem superior, utilizando combinações inteligentes de avaliações da função em pontos estratégicos dentro de cada intervalo de tempo (Chapra; Canale, 2016).

O método mais conhecido, o Runge-Kutta de quarta ordem (RK4), alcança alta precisão através de uma média ponderada de quatro estimativas de inclinação, oferecendo um equilíbrio eficiente entre exatidão e custo computacional. Essas características tornam os métodos Runge-Kutta particularmente úteis para uma ampla gama de problemas em engenharia e ciências aplicadas, onde sistemas dinâmicos complexos demandam soluções numéricas robustas e confiáveis.

2.6.1 Runge-Kutta de Segunda Ordem

De acordo com Esfandiari (2017), os métodos de Runge-Kutta de Segunda Ordem (RK2) genéricos seguem a forma da Equação (36):

$$y_{i+1} = y_i + h(b_1k_1 + b_2k_2) \quad (36)$$

Em que a expressão $(b_1k_1 + b_2k_2)$ é uma estimativa da inclinação da reta. Esta estimativa é obtida pela média ponderada de duas estimativas em pontos distintos do intervalo Δx , k_1 e k_2 são:

$$\begin{cases} k_1 = f(x_i, y_i) \\ k_2 = f(x_i + c_1h, y_i + a_{11}k_1h) \end{cases} \quad (37)$$

As constantes b_1, b_2, c_1, a_{11} são determinadas por expansão em série de Taylor, resultando nas restrições (38) (Esfandiari, 2017):

$$b_1 + b_2 = 1, \quad b_2c_1 = \frac{1}{2}, \quad b_2a_{11} = \frac{1}{2} \quad (38)$$

Note que b_1 e b_2 são ponderações, c_1 é o tamanho do passo usado na estimativa e a_{11} é um fator de ponderação para o uso da estimativa anterior.

2.6.1.1 Método de Heun

Conforme apresentado por Chapra e Canale (2016), o método de Heun é uma técnica de Runge-Kutta de Segunda Ordem que melhora a precisão da estimativa da inclinação em relação ao método de Euler, utilizando um esquema preditor-corretor. A abordagem consiste em calcular duas derivadas no intervalo de integração: uma no ponto inicial e outra no ponto final, obtendo-se uma inclinação média mais precisa.

Primeiro, utiliza-se o método de Euler para extrapolar uma solução aproximada no final do intervalo através da (39):

$$y_{i+1}^0 = y_i + f(x_i, y_i) \cdot h \quad (39)$$

Essa estimativa inicial y_{i+1}^0 permite calcular a inclinação no ponto final:

$$f(x_{i+1}, y_{i+1}^0) \quad (40)$$

Em seguida, calcula-se uma inclinação média combinando as derivadas nos pontos inicial e final (41):

$$\bar{y}' = \frac{y'_i + y'_{i+1}}{2} = \frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^0)}{2} \quad (41)$$

Essa inclinação média é então aplicada na equação (30):

$$y_{i+1} = y_i + \frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^0)}{2} h \quad (\text{Eq. de atualização}) \quad (42)$$

Note que a Equação (42) pode ser escrita como Equação (36) com os coeficientes dados por:

$$b_1 = b_2 = \frac{1}{2}, \quad c_1 = a_{11} = 1 \quad (43)$$

Essa equivalência mostra que o Método de Heun é um caso específico da família RK2, com coeficientes otimizados para minimizar o erro. Por fim, o método pode ser também descrito através das equações (44) e (45):

$$y_{i+1} = y_i + \frac{1}{2} h (k_1 + k_2) \quad (44)$$

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + h, y_i + k_1 h) \end{aligned} \quad (45)$$

O método de Heun é classificado como um método de Runge-Kutta de segunda ordem, pois seu erro de truncamento local é $\mathcal{O}(h^3)$ e seu erro global é da ordem de $\mathcal{O}(h^2)$, proporcionando maior exatidão que o método de Euler (RK1). Essa técnica é particularmente útil para problemas onde se deseja um equilíbrio entre simplicidade computacional e precisão aprimorada.

2.6.1.2 Método do Ponto Médio (Midpoint)

O método do Ponto Médio, ou *Midpoint*, é uma variante específica dos métodos de Runge-Kutta de segunda ordem, obtido a partir da formulação geral RK2 quando se escolhe determinados parâmetros. Para derivar o método do Ponto Médio, adota-se (46):

$$b_1 = 0, b_2 = 1 \quad c_1 = a_{11} = \frac{1}{2} \quad (46)$$

Substituindo esses valores nas equações gerais, obtém-se (47) e (48):

$$y_{i+1} = y_i + hk_2 \quad (47)$$

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1h\right) \end{aligned} \quad (48)$$

Assim como o método de Heun, o método do Ponto Médio possui erro de truncamento local $\mathcal{O}(h^3)$ e global $\mathcal{O}(h^2)$. Enquanto o Heun utiliza uma média ponderada das inclinações no início e no fim do intervalo, o Ponto Médio baseia-se apenas na inclinação no centro do intervalo, o que pode oferecer vantagens em certos tipos de problemas (Esfandiari, 2017).

2.6.1.3 Método de Ralston

O método de Ralston é outra variante importante dos métodos de Runge-Kutta de segunda ordem, caracterizado por uma escolha específica de coeficientes que otimizam a precisão da solução numérica. Para este método os coeficientes são escolhidos como (49):

$$b_1 = \frac{1}{3}, b_2 = \frac{2}{3} \quad c_1 = a_{11} = \frac{3}{4} \quad (49)$$

Substituindo esses valores nas equações gerais, obtém-se (50) e (54):

$$y_{i+1} = y_i + \frac{1}{3}h(k_1 + 2k_2) \quad (50)$$

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f\left(x_i + \frac{3}{4}h, y_i + \frac{3}{4}k_1h\right) \end{aligned} \quad (51)$$

Como característica, este método também apresenta o erro de truncamento local $\mathcal{O}(h^3)$ e global $\mathcal{O}(h^2)$, típico dos métodos de RK2. A escolha dos coeficientes $(\frac{1}{3}, \frac{2}{3})$ minimiza o erro de truncamento, tornando-o mais preciso que outras variantes RK2, pois oferece uma alternativa eficaz para problemas com dinâmicas não lineares. Outra particularidade do método é que ele requer apenas duas avaliações de função por passo, sem necessidade de calcular derivadas explícitas (Esfandiari, 2017).

2.6.2 Runge-Kutta de Quarta Ordem

O método de Runge-Kutta de Quarta Ordem (RK4 Clássico) é amplamente reconhecido como um dos mais eficientes para resolver equações diferenciais ordinárias (EDOs) com condições iniciais. Ele aproxima a solução com precisão equivalente a um método de

Taylor de quarta ordem, mas sem exigir o cálculo explícito de derivadas superiores. Em vez disso, utiliza quatro avaliações da função por passo, combinadas de forma ponderada para minimizar o erro.

A forma clássica do RK4 é descrita por Esfandiari (2017) e Chapra e Canale (2016) como (52):

$$y_{i+1} = y_i + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \quad (52)$$

Para este método os coeficientes são escolhidos como (53):

$$b_1 = \frac{1}{6}, b_2 = b_3 = \frac{1}{3}, b_4 = \frac{1}{6} \quad (53)$$

Onde as inclinações k_1 a k_4 são calculadas sequencialmente (54):

$$\begin{cases} k_1 = f(x_i, y_i) \\ k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1h\right) \\ k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2h\right) \\ k_4 = f(x_i + h, y_i + k_3h) \end{cases} \quad (54)$$

O método de RK4 Clássico possui erro de truncamento local $\mathcal{O}(h^5)$ e global $\mathcal{O}(h^4)$, sendo ideal para problemas com soluções suaves e sistemas não lineares, onde métodos de ordem inferior (como Euler ou Heun) podem falhar.

2.7 MÉTODOS NUMÉRICOS PARA SISTEMAS NO FORMATO ESPAÇO DE ESTADOS

Os métodos numéricos no formato matricial são utilizados para sistemas de ordem superior e como relatado anteriormente, o sistema deve ser descrito através de variáveis de estado. Como visto anteriormente, um sistema de ordem n pode ser descrito através da Equação (10) onde $x(t)$ representa o vetor de estados e $u(t)$ representa o vetor de entradas.

2.7.1 Euler Explícito Matricial

Considerando-se a equação diferencial ordinária (EDO) no formato de espaço de estados como (55):

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \quad (55)$$

A discretização do método de Euler Explícito pode ser efetuada utilizando-se a aproximação da derivada pela Equação (56), onde h é o passo de discretização (período de amostragem):

$$\dot{x}(t) \approx \frac{x(k+1) - x(k)}{h} \quad (56)$$

Substituindo a aproximação na Equação (55) obtém-se (57):

$$\frac{\mathbf{x}(k+1) - \mathbf{x}(k)}{h} = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \quad (57)$$

Rearranjando a equação anterior e acrescentando o termo $\mathbf{I}$ que representa a matriz identidade, encontra-se (58) :

$$\mathbf{x}(k+1) = (h\mathbf{A} + \mathbf{I})\mathbf{x}(k) + h\mathbf{B}\mathbf{u}(k) \quad (58)$$

Por fim, considerando-se h como a constante Δt que melhor representa o sistema em tempo discreto, a forma matricial em espaço de estados do método de Euler Explícito é dada pela Equação (59):

$$\begin{cases} \mathbf{x}[k+1] = (\Delta t \cdot \mathbf{A} + \mathbf{I})\mathbf{x}[k] + \Delta t \cdot \mathbf{B}\mathbf{u}[k] \\ \mathbf{y}[k] = \mathbf{C}\mathbf{x}[k] + \mathbf{D}\mathbf{u}[k] \end{cases} \quad (59)$$

2.7.2 Euler Implícito Matricial

De maneira similar, para o método Euler Implícito, a aproximação da derivada utiliza o estado futuro (60):

$$\dot{x}(t+1) \approx \frac{x(k+1) - x(k)}{h} \quad (60)$$

Substituindo a aproximação na Equação (55) obtém-se (61):

$$\frac{\mathbf{x}(k+1) - \mathbf{x}(k)}{h} = \mathbf{A}\mathbf{x}(k+1) + \mathbf{B}\mathbf{u}(k+1) \quad (61)$$

Rearranjando a equação anterior, encontra-se (62) :

$$\begin{aligned} \mathbf{x}(k+1) - h\mathbf{A}\mathbf{x}(k+1) &= \mathbf{x}(k) + h\mathbf{B}\mathbf{u}(k+1) \\ (\mathbf{I} - h\mathbf{A})\mathbf{x}(k+1) &= \mathbf{x}(k) + h\mathbf{B}\mathbf{u}(k+1) \end{aligned} \quad (62)$$

Assumindo $(\mathbf{I} - \Delta t \cdot \mathbf{A})$ como inversa, tem-se (63):

$$\mathbf{x}(k+1) = (\mathbf{I} - h\mathbf{A})^{-1}\mathbf{x}(k) + (\mathbf{I} - h\mathbf{A})^{-1}h\mathbf{B}\mathbf{u}(k+1) \quad (63)$$

Por fim, a forma matricial em espaço de estados do método de Euler Implícito é dada pela Equação (64):

$$\begin{cases} \mathbf{x}[k+1] = (\mathbf{I} - \Delta t \cdot \mathbf{A})^{-1}\mathbf{x}[k] + (\mathbf{I} - \Delta t \cdot \mathbf{A})^{-1}\Delta t \cdot \mathbf{B}\mathbf{u}[k+1] \\ \mathbf{y}[k] = \mathbf{C}\mathbf{x}[k] + \mathbf{D}\mathbf{u}[k] \end{cases} \quad (64)$$

2.8 MÉTODOS DE RUNGE-KUTTA NO FORMATO ESPAÇO DE ESTADOS

A seguir, apresenta-se o desenvolvimento da formulação matricial do método de Runge-Kutta, utilizando como exemplo o Método do Ponto Médio (RK2). O exemplo baseia-se em um sistema de ordem 2 dado por (65):

$$\begin{aligned} \dot{x}_1 &= f_1(\mathbf{x}(t), \mathbf{u}(t)) \\ \dot{x}_2 &= f_2(\mathbf{x}(t), \mathbf{u}(t)) \\ &\vdots \\ \dot{x}_n &= f_n(\mathbf{x}(t), \mathbf{u}(t)) \\ y &= g(\mathbf{x}(t), \mathbf{u}(t)) \end{aligned} \quad (65)$$

Ou em notação matricial (66):

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} a_{p11} & a_{p12} \\ a_{p21} & a_{p22} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \cdot u \quad (66)$$

$$y = \begin{bmatrix} c_1 & c_2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} d_1 \end{bmatrix} \cdot u \quad (67)$$

Uma abordagem comum na representação matricial consiste em utilizar coeficientes ponderados, denotados por $\mathbf{m}_1$ e $\mathbf{m}_2$, para cada estágio de cálculo. No caso específico do Método do Ponto Médio (RK2), que possui ordem 2, são necessários dois estágios sequenciais:

1. **Primeiro estágio ($\mathbf{m}_1$):** Calcula-se a derivada inicial do estado no instante atual ($t = k \cdot \Delta t$):

$$\mathbf{m}_1 = \begin{bmatrix} m_{11} \\ m_{12} \end{bmatrix} = \begin{bmatrix} f_1(\mathbf{x}[k], \mathbf{u}[k]) \\ f_2(\mathbf{x}[k], \mathbf{u}[k]) \end{bmatrix} \quad (68)$$

2. **Segundo estágio ($\mathbf{m}_2$):** Determina-se a derivada no ponto intermediário, meio passo à frente ($t = (k + \frac{1}{2}) \cdot \Delta t$) resultando em:

$$\mathbf{m}_2 = \begin{bmatrix} m_{21} \\ m_{22} \end{bmatrix} = \begin{bmatrix} f_1 \left(\mathbf{x}[k] + \frac{1}{2} \cdot \mathbf{m}_1 \cdot \Delta t \right), \mathbf{u} \left((k + \frac{1}{2}) \cdot \Delta t \right) \\ f_2 \left(\mathbf{x}[k] + \frac{1}{2} \cdot \mathbf{m}_1 \cdot \Delta t \right), \mathbf{u} \left((k + \frac{1}{2}) \cdot \Delta t \right) \end{bmatrix} \quad (69)$$

Os sinais discretizados preservam os valores dos sinais contínuos nos instantes de amostragem, de modo que $x(k \cdot \Delta t)$ corresponde exatamente a $x[k]$, onde Δt representa o intervalo fixo entre amostras.

No primeiro estágio do método, obtém-se as equações:

$$\begin{aligned} m_{11} &= f_1(\mathbf{x}[k], \mathbf{u}(k \cdot \Delta t)) = a_{p11} \cdot x_1[k] + a_{p12} \cdot x_2[k] + b_1 \cdot u(k \cdot \Delta t) \\ m_{12} &= f_2(\mathbf{x}[k], \mathbf{u}(k \cdot \Delta t)) = a_{p21} \cdot x_1[k] + a_{p22} \cdot x_2[k] + b_2 \cdot u(k \cdot \Delta t) \end{aligned} \quad (70)$$

Para o segundo estágio do método, encontra-se:

$$\begin{aligned} m_{21} &= f_1 \left(\mathbf{x}[k] + \frac{1}{2} \cdot \mathbf{m}_1 \cdot \Delta t \right), \mathbf{u} \left(\left(k + \frac{1}{2} \right) \cdot \Delta t \right) \rightarrow \\ m_{21} &= a_{p11} \cdot \left[\left(x_1[k] + \frac{1}{2} \cdot m_{11} \cdot \Delta t \right) \right] + a_{p12} \cdot \left[\left(x_2[k] + \frac{1}{2} \cdot m_{12} \cdot \Delta t \right) \right] + \dots \\ &\quad \dots + b_1 \cdot u \left(k \cdot \Delta t + \frac{1}{2} \cdot \Delta t \right) \end{aligned} \quad (71)$$

e

$$\begin{aligned} m_{22} &= f_2 \left(\mathbf{x}[k] + \frac{1}{2} \cdot \mathbf{m}_1 \cdot \Delta t \right), \mathbf{u} \left(\left(k + \frac{1}{2} \right) \cdot \Delta t \right) \rightarrow \\ m_{22} &= a_{p21} \cdot \left[\left(x_1[k] + \frac{1}{2} \cdot m_{11} \cdot \Delta t \right) \right] + a_{p22} \cdot \left[\left(x_2[k] + \frac{1}{2} \cdot m_{12} \cdot \Delta t \right) \right] + \dots \\ &\quad \dots + b_2 \cdot u \left(k \cdot \Delta t + \frac{1}{2} \cdot \Delta t \right) \end{aligned} \quad (72)$$

Na etapa final, o cálculo dos estados no instante seguinte é realizado através da combinação de duas estimativas de taxa de variação, conforme expresso nas equações de atualização do sistema:

$$\begin{aligned} x_1[k+1] &= x_1[k] + \Delta t \cdot (a_1 m_{11} + a_2 m_{21}) \\ x_2[k+1] &= x_2[k] + \Delta t \cdot (a_1 \cdot m_{12} + a_2 \cdot m_{22}) \end{aligned} \quad (73)$$

Para o ponto intermediário como visto na Seção 2.6.1.2 o valor do passo é $\Delta t = \frac{1}{2}$, os valores dos coeficientes $a_1 = 0$ e $a_2 = 1$, assim, substituindo-se os valores em (73) obtém-se (74):

$$\begin{aligned}x_1[k+1] &= x_1[k] + \Delta t \cdot m_{21} \\x_2[k+1] &= x_2[k] + \Delta t \cdot m_{22}\end{aligned}\tag{74}$$

2.9 TABELA DE BUTCHER

A Tabela de Butcher, desenvolvida pelo matemático John C. Butcher na década de 1960 (Butcher, 2008), é uma ferramenta essencial para a análise e implementação de métodos de Runge-Kutta (RK), amplamente utilizados na solução numérica de equações diferenciais ordinárias (EDOs). Essa tabela padroniza a representação dos coeficientes que definem os métodos RK, facilitando a comparação entre diferentes esquemas, a análise de suas propriedades matemáticas e sua implementação computacional eficiente. Sua estrutura matricial organiza de forma clara os coeficientes de estágio, pontos de amostragem e pesos de combinação, permitindo uma visão unificada de métodos explícitos e implícitos.

Um método de Runge-Kutta com n estágios é definido pelas seguintes equações (Butcher, 2008):

$$y_{n+1} = y_n + h \sum_{i=1}^s b_i k_i \tag{75}$$

Neste caso, as inclinações k_i são calculadas como (76):

$$k_i = f \left(t_n + c_i h, y_n + h \sum_{j=1}^s a_{ij} k_j \right), \quad i = 1, 2, \dots, s \tag{76}$$

Onde:

- h é o passo de integração;
- b_i são os pesos de combinação;
- c_i são os pontos no intervalo $[0, 1]$ onde as inclinações são avaliadas;
- a_{ij} são os coeficientes que determinam a dependência entre os estágios.

A Tabela de Butcher organiza esses coeficientes em uma estrutura matricial que pode ser observada em (77):

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} \tag{77}$$

De acordo com Butcher (2008), a estrutura é dividida em três componentes principais:

1. Matriz de Coeficientes (A):

- Uma matriz $s \times s$ onde a_{ij} define como as inclinações k_j contribuem para o cálculo de k_i ;
- Para métodos explícitos, A é triangular inferior ($a_{ij} = 0$ para $j \geq i$);
- Para métodos implícitos, A pode ser uma matriz completa.

Por exemplo, para o método de Ralston (RK2):

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{3}{4} & \frac{3}{4} & 0 \\ \hline & \frac{1}{3} & \frac{2}{3} \end{array} \quad (78)$$

2. Vetor de Nós (c):

- Pontos c_i no intervalo $[0, 1]$ onde as inclinações são avaliadas, calculados por:

$$c_i = \sum_{j=1}^s a_{ij} \quad (79)$$

- Garante que as inclinações sejam amostradas em locais estrategicamente escolhidos.

3. Vetor de Pesos (b):

- Coeficientes b_i que ponderam as inclinações k_i na atualização final. Devem satisfazer (80):

$$\sum_{i=1}^s b_i = 1 \quad (\text{condição de consistência}). \quad (80)$$

- Determina a ordem de precisão do método.

A implementação numérica dos métodos de Runge-Kutta de segunda, terceira e quarta ordens (RK2, RK3 e RK4) neste trabalho será realizada utilizando as Tabelas de Butcher, que fornecem uma estrutura padronizada para os coeficientes de cada método. Essa abordagem permite uma implementação genérica e modular, onde a mesma função computacional pode ser reutilizada para diferentes esquemas RK, bastando alterar a tabela de coeficientes. Para exemplificar, a seguir listam-se as tabelas principais (81), (82) e (83), as demais tabelas podem ser verificadas no Anexo A:

• Método de RK2 - Ponto Médio:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array} \quad (81)$$

- Coeficientes:

$$c = \left[0, \frac{1}{2}\right], A = \begin{bmatrix} 0 & 0 \\ \frac{1}{2} & 0 \end{bmatrix}, b = [0, 1].$$

- Método de RK3 - Clássico:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 1 & -1 & 2 & 0 \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \quad (82)$$

- Coeficientes:

$$c = \left[0, \frac{1}{2}, 1\right], A = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ -1 & 2 & 0 \end{bmatrix}, b = \left[\frac{1}{6}, \frac{2}{3}, \frac{1}{6}\right].$$

- Método de RK4 - Clássico:

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array} \quad (83)$$

- Coeficientes:

$$c = \left[0, \frac{1}{2}, \frac{1}{2}, 1\right], A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, b = \left[\frac{1}{6}, \frac{1}{3}, \frac{1}{3}, \frac{1}{6}\right].$$

A tabela de Butcher para um método de Runge-Kutta não é única para uma determinada ordem. As condições de ordem formam um sistema subdeterminado de equações, permitindo graus de liberdade na escolha dos coeficientes. Estes graus de liberdade são frequentemente utilizados para desenvolver métodos com propriedades adicionais desejáveis, como regiões de estabilidade ampliada, eficiência computacional ou a capacidade de produzir estimativas de erro para implementações de passo variável. Portanto, diferentes implementações de Runge-Kutta de mesma ordem, mas com tabelas de Butcher distintas, estão disponíveis na literatura.

2.10 GNU OCTAVE

O GNU Octave foi desenvolvido em 1988 como um *software* complementar para um livro universitário sobre projeto de reatores químicos, escrito por James B. Rawlings (Uni-

versidade de Wisconsin-Madison) e John G. Ekerdt (Universidade do Texas). Inicialmente planejado como uma ferramenta especializada para problemas de engenharia química, os desenvolvedores logo perceberam a necessidade de uma solução mais flexível e abrangente (Octave Community, 2023).

Apesar das sugestões para usar Fortran (considerada a "linguagem da engenharia"), os criadores optaram por desenvolver um ambiente interativo que permitisse aos estudantes focar nos conceitos de engenharia, sem se perder em problemas de programação. O desenvolvimento em tempo integral começou em 1992, com o primeiro lançamento *alpha* em 1993 e a versão 1.0 em 1994. Desde então, o Octave tornou-se um *software* amplamente utilizado, integrado em distribuições GNU/Linux e adotado globalmente para ensino, pesquisa e aplicações comerciais (Octave Community, 2023).

De acordo com Octave Community (2023), o GNU Octave é uma linguagem de alto nível projetada para cálculos numéricos, oferecendo uma interface de linha de comando interativa para resolver problemas lineares e não lineares. Sua sintaxe é compatível com MATLAB, permitindo a transição fácil entre as duas plataformas. Além disso, o Octave pode ser usado para processamento em *batch* (modo não- interativo). A ferramenta apresenta como principais recursos:

- Álgebra linear numérica: Solução de sistemas de equações, decomposições matriciais, etc;
- Equações não-lineares: Cálculo de raízes e otimização;
- Integração numérica: Solução de equações diferenciais;
- Manipulação de polinômios e equações diferenciais-algébricas;
- Extensibilidade: Permite a criação de funções personalizadas em Octave ou a integração de módulos em C++, C e Fortran.

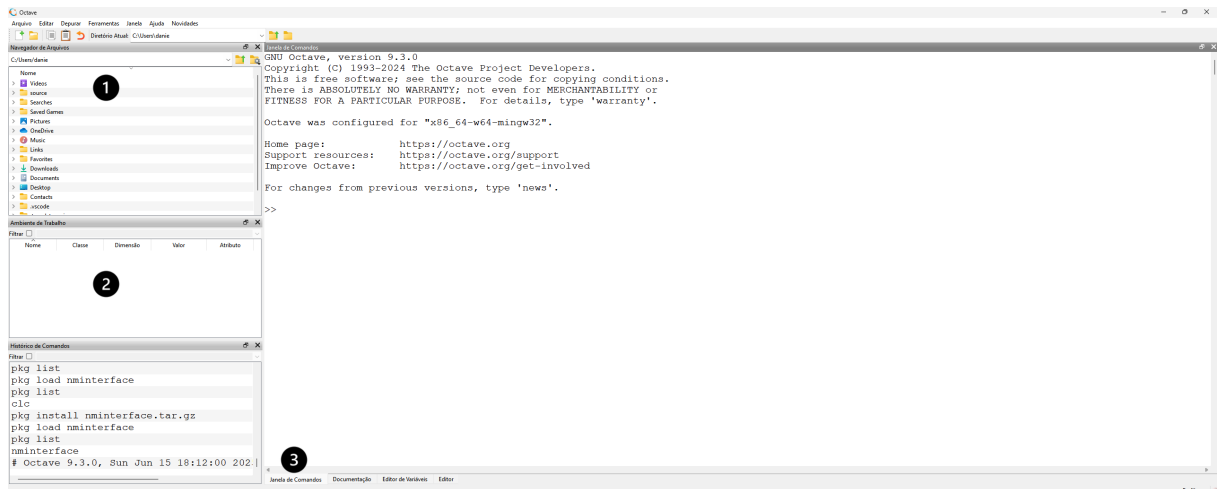
O Octave é um *software* livre, distribuído sob a GNU GPL (do inglês, *General Public License*), o que significa que usuários podem modificá-lo e redistribuí-lo livremente, incentivando a colaboração da comunidade. O projeto incentiva a contribuição de novos recursos e a reportagem de problemas para aprimorar a ferramenta, fortalecendo assim, o seu desenvolvimento contínuo.

O ambiente integrado de desenvolvimento do *software* GNU Octave apresenta uma arquitetura funcional dividida em três componentes fundamentais, ilustrado pela Figura 5, projetados para otimizar o fluxo de trabalho em computação numérica:

1. Navegador de Arquivos (Painel superior esquerdo):

Esta seção apresenta funcionalidades essenciais para gestão de projetos, permitindo:

Figura 5 – Tela inicial do GNU Octave.



Fonte: Elaborado pela autora (2025).

- Definição do *workspace* de trabalho;
- Organização hierárquica de arquivos de código-fonte (.m);
- Armazenamento estruturado de conjuntos de dados e resultados.

A correta utilização deste módulo é particularmente relevante em projetos complexos envolvendo múltiplos arquivos interdependentes.

2. Ambiente de Trabalho (*workspace* - Área subjacente ao navegador):

O (*workspace* apresenta uma visão tabular abrangente de todas as variáveis definidas no ambiente atual, contendo dados fundamentais para a análise:

- Nome: Identificador único;
- Classe: Tipo de dado (numérico, celular, estrutural);
- Dimensão: Matriz;
- Valor: Conteúdo armazenado;
- Atributo: Propriedades especiais (Complexidade, Esparsidade).

Esta representação sistemática facilita significativamente processos de validação e depuração durante o desenvolvimento de algoritmos.

3. Janela de Comandos (Parte central inferior):

Oferece um ambiente interativo para:

- a) Processamento imediato de comandos;
- b) Prototipagem rápida de soluções;
- c) Análise exploratória de dados.

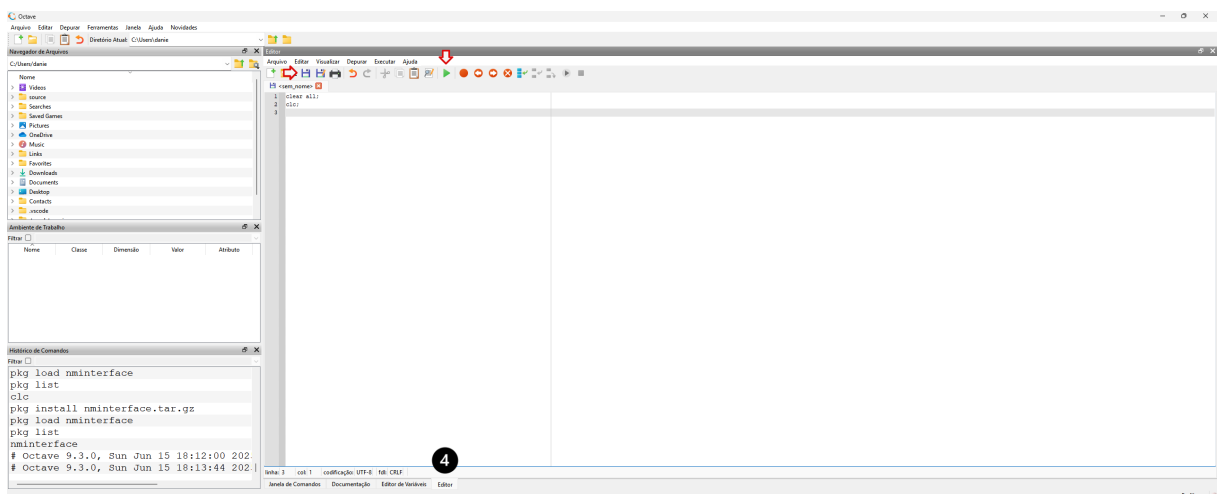
É importante destacar que a janela de comandos opera em um regime não persistente, onde as instruções inseridas permanecem disponíveis apenas durante a sessão atual, sendo descartadas ao final da operação.

Outro componente de suma importância é o editor de *scripts*, como pode ser observado na Figura 6. O editor define a parte fundamental da interface, e é possível acessá-lo através do botão no canto inferior direito ("Editor"), essa janela permite:

- Escrever e salvar *scripts* (extensão .m) para reutilização;
- Executar códigos diretamente no ambiente Octave (via botão "Salvar Arquivo e Executá-lo"— representados pelos ícones disquete azul e triângulo verde);
- Automatizar tarefas repetitivas, garantindo reprodutibilidade.

Como vantagens pedagógicas, o Octave é ideal para aulas de modelagem e simulação, onde estudantes podem testar algoritmos e comparar resultados além de efetuar a integração entre edição, execução e visualização de variáveis em um único ambiente.

Figura 6 – Editor de *scripts* do GNU Octave.



Fonte: Elaborado pela autora (2025).

2.11 GUIEDITOR

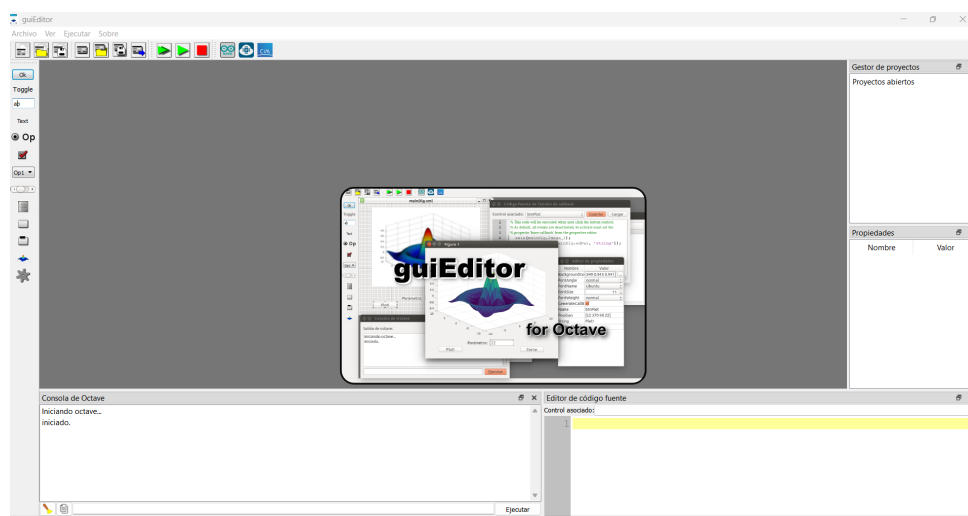
O guiEditor (Burgos; Adam, 2025) é um ambiente de desenvolvimento integrado (IDE) projetado para facilitar a criação de interfaces gráficas de usuário (GUIs) no GNU Octave, especialmente voltado para aplicações educacionais e de engenharia. Desenvolvido como parte de um projeto colaborativo entre laboratórios de universidades argentinas, essa ferramenta surgiu da necessidade de permitir que usuários com conhecimentos básicos de programação pudessem construir aplicações interativas sem depender de linguagens

complexas como C++ ou Qt (Burgos; Adam, 2020). Portanto, o objetivo principal era desenvolver uma ferramenta que permitisse a usuários com conhecimentos básicos de GNU Octave criar aplicações com interfaces gráficas (GUI). A ferramenta apresenta como principais funcionalidades:

1. **Editor Visual Intuitivo:** Permite a construção de GUIs através de movimentos como arrastar e soltar, gerando automaticamente *scripts* compatíveis com funções nativas do Octave como *uicontrol*;
2. **Gerenciamento de Projetos:** Organiza componentes em pastas dedicadas (janelas (gui), imagens (img), funções (fcn)), exportando projetos como *scripts* ou pacotes multiplataforma;
3. **Integração com GNU Octave:** Execução e depuração direta no ambiente, com exibição de resultados em terminal integrado.

Segundo Burgos e Adam (2020), aplicações desenvolvidas no guiEditor facilita a curva de aprendizado em disciplinas como Controle de Processos, permitindo que alunos concentrem-se nos fundamentos de controle e modelagem de sistemas, abstraindo as dificuldades inerentes à programação de interfaces gráficas. A ferramenta também se caracteriza por ser multiplataforma, assim, as aplicações desenvolvidas possuem compatibilidade *cross-platform*, sendo executáveis em qualquer ambiente que suporte o GNU Octave. Por fim, o guiEditor, conforme a Figura 7, apresenta versatilidade de uso, uma vez que o ambiente é apropriado tanto para a criação ágil de interfaces quanto para o desenvolvimento de *toolboxes* especializadas, atendendo a diferentes níveis de complexidade de projeto.

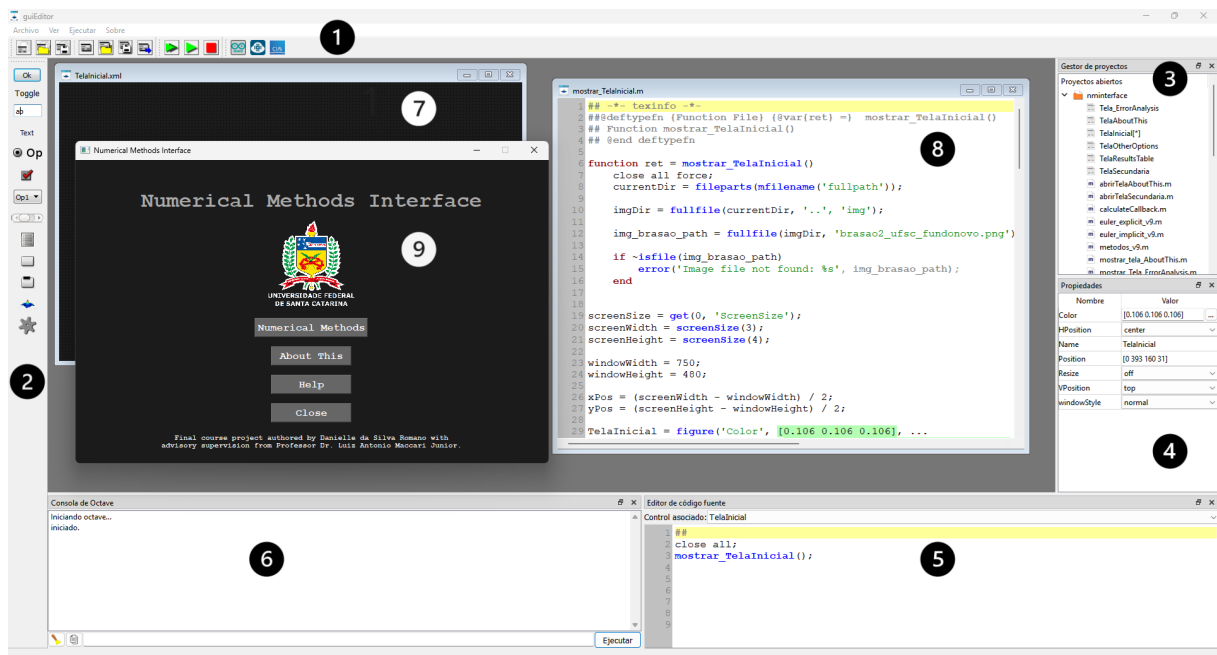
Figura 7 – Tela inicial guiEditor.



Fonte: Elaborado pela autora (2025).

O ambiente do guiEditor é composto por nove elementos principais, organizados para facilitar o desenvolvimento de interfaces gráficas no GNU Octave, como pode ser observado na Figura 8:

Figura 8 – guiEditor executando o *script* do *Software Didático Numerical Methods Interface*.



Fonte: Elaborado pela autora (2025).

1. **Barra de Ferramentas Superior:** Contém opções para gerenciamento de arquivos, depuração do projeto e configuração de dispositivos de *hardware*;
2. **Barra de Controles Gráficos:** Permite adicionar elementos interativos à interface, como botões de ação, gráficos dinâmicos, menu suspensos (*comboboxes*), campos de texto editáveis e rótulos e títulos;
3. **Gerenciador de projetos:** Organiza a estrutura do projeto, possibilitando a adição/remoção de janelas de diálogo, inclusão de *scripts* em formato .m, importação de recursos visuais (imagens) e configurações gerais do aplicativo;
4. **Editor de Propriedades:** Permite personalizar cada componente da GUI, ajustando atributos como posicionamento (*coordinates*), dimensões (largura e altura - do inglês, *width* e *height*), estilo (fonte, cor de fundo, bordas) e nome de identificação (*tag*);
5. **Editor de Código-Fonte:** Ambiente dedicado para programação dos *scripts* associados a eventos de componentes (ex.: *callback de botões*) e funções auxiliares;

6. **Terminal do Octave:** Exibe em tempo real os resultados numéricos das simulações, mensagens de erro ou warnings e saída de comandos executados;
7. **Área de Design da Janela de Diálogo:** Espaço visual para construção *drag-and-drop* (arrastar e soltar) da interface gráfica.
8. **Arquivo de *Script*:** Armazenam funções personalizadas chamadas pela interface principal;
9. **Visualização da Interface:** Prévia da GUI após compilação, mostrando o *layout* final com todos os componentes posicionados.

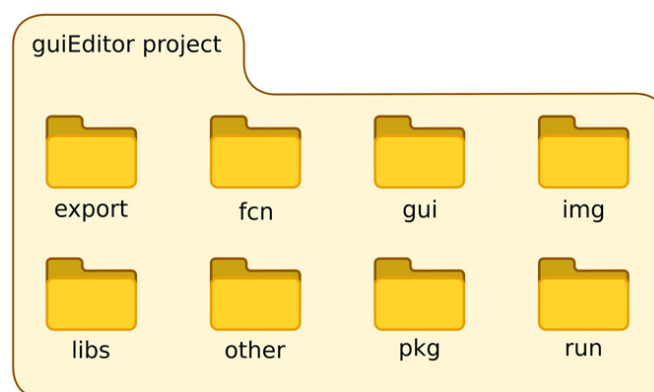
Conforme destacado por (Burgos; Adam, 2020), o guiEditor adota uma estrutura de projetos para organizar o desenvolvimento de aplicações com interfaces gráficas no GNU Octave. Cada projeto pode conter:

- Janelas de diálogo (GUIs);
- Arquivos de funções (.m);
- Recursos visuais (imagens).

Ao finalizar o desenvolvimento, o projeto pode ser exportado como arquivos de *script* individuais ou como pacotes do GNU Octave, seguindo a abordagem descrita pelos autores, onde a janela principal é exibida automaticamente ao carregar o pacote, enquanto as demais são acessíveis via chamada programática.

A Figura 9 ilustra a arquitetura padrão de um projeto no ambiente de desenvolvimento, com sua estrutura de diretórios gerada automaticamente pelo sistema. Particularmente, o diretório *pkg* (do inglês, *packages*) tem função essencial no processo de distribuição, pois armazena o arquivo compactado do pacote Octave pronto para implantação e todos os *scripts* necessários para execução autônoma da aplicação.

Figura 9 – Estrutura de diretório de um projeto no guiEditor.



Fonte: (Burgos; Adam, 2020).

Como destacam os autores, esta organização facilita significativamente o compartilhamento e a portabilidade das soluções desenvolvidas, características fundamentais para ferramentas educacionais no ambiente GNU Octave (Burgos; Adam, 2020).

3 SOFTWARE DESENVOLVIDO

3.1 DESENVOLVIMENTO DO *SOFTWARE* DIDÁTICO

Este capítulo detalha a metodologia empregada no desenvolvimento do *Numerical Methods Interface* (NMI), ferramenta computacional *open-source* criada em GNU Octave para análise de sistemas dinâmicos no espaço de estados. Conforme estabelecido na Seção 1.1.2, o *software* integra métodos numéricos clássicos e avançados (Euler, Runge-Kutta) em uma interface gráfica intuitiva, destinada a aplicações didáticas em Engenharia. A implementação realizada apresenta como características:

1. Implementação no GNU Octave, aproveitando suas bibliotecas matemáticas nativas;
2. *Design* de GUI via guiEditor, facilitando a interação do usuário com os métodos implementados;
3. Validação pedagógica, assegurando precisão computacional e usabilidade em ambientes educacionais.

As seções subsequentes descrevem a arquitetura do ambiente GNU Octave e suas funcionalidades para simulação numérica e também abordam a estrutura do guiEditor, detalhando os principais elementos que compõem o editor de interfaces gráficas. A seção também aborda o processo de desenvolvimento da interface gráfica, desde o *design* até a implementação e a integração entre os métodos numéricos e a representação visual dos resultados.

3.2 *SOFTWARE* PROPOSTO

O *software* desenvolvido neste trabalho tem como objetivo principal fornecer uma interface intuitiva e funcional para a simulação de sistemas dinâmicos representados no espaço de estados, utilizando diferentes métodos numéricos. A interface, construída no ambiente GNU Octave, permite que o usuário insira os parâmetros necessários para a simulação de forma clara e organizada, sem exigir conhecimentos avançados de programação.

Por meio da interface gráfica, o usuário deve fornecer as matrizes do espaço de estados (**A**, **B**, **C** e **D**), os parâmetros de simulação (tempo inicial **t0**, tempo final **tf**, passo de integração **dt** e condição inicial **x0**), além de selecionar o método numérico desejado e o tipo de entrada a ser aplicada ao sistema — como degrau, senoidal, exponencial ou um vetor personalizado. Caso o usuário deseje, também é possível incluir a função analítica da resposta do sistema. É importante destacar que, caso essa solução não seja fornecida, a simulação será realizada utilizando a função **lsim** do próprio Octave, para prover a solução

de referência para o cálculo do erro dos métodos estudados, assegurando a flexibilidade do *software*.

3.2.1 Lista de funcionalidades

O *software* desenvolvido oferece um conjunto abrangente de funcionalidades projetadas para facilitar a análise e simulação de sistemas dinâmicos lineares. A seguir, são elencadas as principais capacidades da interface:

1. Configuração do Modelo no Espaço de Estados

- Entrada das matrizes do sistema (A, B, C, D) diretamente pelo usuário.
- Definição da condição inicial do sistema (x_0).

2. Parâmetros de Simulação Customizáveis

- Especificação do intervalo de simulação: tempo inicial (t_0) e tempo final (t_f).
- Definição do passo de integração (dt) para os métodos numéricos.

3. Seleção de Métodos Numéricos

- Escolha entre métodos explícitos, implícitos e runge-kutta.

4. Tipos de Entrada do Sistema

- Seleção entre diferentes sinais de entrada predefinidos: Degrau (*heaviside*), Impulso (*Impulse*), Senoidal (*Sinusoidal*), Exponencial (*Exponential*) e possibilidade de definir um sinal de entrada personalizada pelo usuário.

5. Análise Comparativa com Solução Analítica

- Campo opcional para inserção da função analítica do sistema.
- Caso não fornecida, a simulação é realizada utilizando a função **lsim** do GNU Octave como referência.

6. Visualização de Resultados

- Gráfico da resposta do sistema ao longo do tempo.
- Gráfico da evolução das variáveis de estado.
- Geração da função de transferência do sistema.
- Geração de diagramas de análise dinâmica do sistema (Bode, Nyquist, Root Locus, PZMap).
- Geração de tabelas numéricas com os resultados da simulação.

7. Gerenciamento de Dados e Simulações

- Opção para salvar configurações e resultados.
- Botão de *reset* para reiniciar parâmetros.

8. Interação com o Usuário

- Mensagens de ajuda integradas.
- Interface gráfica intuitiva desenvolvida no GNU Octave.
- Informações sobre autoria e orientação acessíveis diretamente na interface.

9. Validação Educacional

- Permite comparação entre métodos numéricos e solução analítica.
- Facilita o entendimento de conceitos de modelagem e sistemas dinâmicos.

3.2.2 Lista de Entradas e Saídas

O *software* desenvolvido opera com um conjunto bem definido de entradas fornecidas pelo usuário e gera saídas correspondentes para análise do sistema. A seguir, apresenta-se a especificação completa das entradas e saídas do sistema:

Entradas do Sistema

As entradas requeridas do usuário são organizadas de forma lógica na interface gráfica:

1. Parâmetros do Modelo no Espaço de Estados

- Matriz A (Matriz de Estado);
- Matriz B (Matriz de Entrada);
- Matriz C (Matriz de Saída);
- Matriz D (Matriz de Transmissão Direta).

2. Condições Iniciais e Parâmetros de Simulação

- Vetor de Condições Iniciais (x_0);
- Tempo Inicial da Simulação (t_0);
- Tempo Final da Simulação (t_f);
- Passo de Integração (dt).

3. Configurações de Métodos Numéricos

- Seleção do Método: Euler Explícito, Euler Implícito, RK (RK2/RK3/RK4).

4. Configuração de Entrada

- Tipo de Sinal de Entrada: degrau, impulso, senoidal, exponencial, personalizada.

5. Função Analítica (opcional)

- Nome da Função Analítica.

Saídas do Sistema

As saídas geradas pelo sistema permitem uma análise completa dos resultados:

1. Resultados Gráficos

- Gráfico "*System Response*": Resposta temporal do sistema com eixo Y em amplitude e eixo X em tempo (s).
- Gráfico "*Evolution of States*": Evolução temporal dos estados do sistema.

2. Resultados Numéricos

- Tabela "*Results Table*": Apresenta valores temporais, estados e saídas do sistema.
- Tabela "*Error Analysis*": Apresenta erros absolutos e relativos quando comparado com solução analítica ou **lsim**.
- Estrutura "*Results Simulation*": Contém dados completos da simulação para análise posterior.

3. Análise do Sistema

- Função de Transferência: Cálculo e exibição da função de transferência equivalente derivada das matrizes de espaço de estados.
- Diagramas de análise dinâmica: Bode, Nyquist, Root Locus, PZMap.

4. Arquivos e Relatórios

- Salvamento dos gráficos e tabelas geradas através do botão "*Save*".

5. Mensagens do Sistema

- Mensagens de erro para entradas inválidas.
- Confirmações de operação bem-sucedida.

3.2.3 Diagrama de funções

A implementação computacional deste trabalho foi desenvolvida iniciando-se pela construção do código principal **main_v9.m**, responsável por integrar todas as funções necessárias para a simulação numérica do sistema dinâmico. A estrutura do código segue uma arquitetura modular em três níveis, conforme ilustrado na Figura 10:

Onde:

1. Nível 0 (Orquestração): O *script* principal **main_v9.m** coordena todo o processo, desde a configuração do sistema até a plotagem dos resultados, chamando iterativamente o gerenciador **metodos_v9.m**;

Figura 10 – Diagrama de Estrutura e Relação entre os Códigos.



Fonte: Elaborado pela autora (2025).

2. **Nível 1 (Gerenciador):** `metodos_v9.m` atua como interface, direcionando via *switch-case* para os solucionadores específicos (Euler ou Runge-Kutta);
3. **Nível 2 (Solucionadores):** Módulos especializados (`euler_explicit_v9.m`, `euler_implicit_v9.m`, `rk_v9.m`) executam os cálculos numéricos de forma isolada.

Essa estrutura promove reúso de código, facilita a manutenção e permite a extensão para novos métodos.

Esta abordagem modular oferece três vantagens principais: (1) separação clara de responsabilidades, com cada nível executando funções bem definidas; (2) escalabilidade, permitindo a adição de novos métodos numéricos através de simples atualizações no gerenciador (Nível 1); e (3) clareza na visualização, onde o diagrama explicita intuitivamente o fluxo de dados entre os módulos, facilitando a compreensão e manutenção do sistema.

3.2.4 Função principal `main_v9.m`

1) Definição do Sistema no formato Espaço de Estados

O sistema dinâmico é modelado na representação de espaço de estados, que permite uma descrição compacta e eficiente para análise e simulação. Conforme ilustrado no Código 3.1, o usuário deve fornecer as quatro matrizes fundamentais que definem completamente o comportamento do sistema, bem como os parâmetros de simulação necessários.

Código 3.1 – Definição e criação do sistema.

```
1 close all; clc; clear all;
2
3 A = [-1 -2 -3; %Matriz de estado
4      3 -4 -5;
5      0 4 -2];
6 B = [1; 1; 0]; %Matriz de entrada
7 C = [1 0 0];   %Matriz de saída
8 D = [0];       %Matriz de transmissao direta
9
10 sys = ss(A, B, C, D); % Criacao do sistema
```

Fonte: Elaborado pela autora (2025).

As matrizes e parâmetros apresentados servem como exemplo para validação do *software*. O usuário deve substituí-los pelos valores do sistema que deseja analisar, mantendo as dimensões consistentes:

- Matriz A: dimensão $[n \times n]$, onde n é o número de estados;
- Matriz B: dimensão $[n \times p]$, onde p é o número de entradas;
- Matriz C: dimensão $[q \times n]$, onde q é o número de saídas
- Matriz D: dimensão $[q \times p]$

O código principal do *software* obtém automaticamente as variáveis definidas pelo usuário no *workspace* do Octave e utiliza esses valores para executar as simulações numéricas. Esta abordagem permite flexibilidade na configuração dos parâmetros e facilita a realização de múltiplas simulações com diferentes configurações.

Esta representação matricial, combinada com os parâmetros de simulação fornecidos, fornece ao *software* todas as informações necessárias para realizar as simulações temporais, calcular a função de transferência equivalente e gerar as análises de erro do sistema.

2) Configuração dos Parâmetros de Simulação

Além das matrizes do sistema, o usuário deve definir no *workspace* do Octave os parâmetros de simulação, entradas e função analítica caso deseje, conforme detalhado na Seção 3.2.2. Como exemplo apresentado, para validação posterior da interface utilizou-se os seguintes parâmetros de simulação, incluindo:

- **Tempo de simulação:** $t_0 = 0$ a $t_f = 4$ segundos, com passo $dt = 0.01$;
- **Entrada:** Degrau unitário ($u = @(t) \text{double}(t \geq 0)$). O uso de *double* é uma correção para o degrau unitário tradicional - função *heaviside* - no Octave, pois ela retorna o valor de 0.5 e não 1. Após a correção a função retornou corretamente o valor 1 para $t \geq 0$ e 0 para $t < 0$);
- **Condições iniciais:** $x_0 = [0; 0; 0]$.

3) Chamada das Funções dos Métodos Numéricos

O código principal integra as funções implementadas para os métodos de Euler (explícito e implícito) e Runge-Kutta (RK2, RK3 e RK4), com variantes como Ponto Médio, Ralston e Heun. O exemplo pode ser observado em 3.2:

Código 3.2 – Código parcial das chamadas de funções utilizadas.

```
1 x_explicit = euler_explicit_v9(sys, u, x0, dt, t);
2 x_implicit = euler_implicit_v9(sys, u, x0, dt, t);
3 x_rk4_classico = rk_v9(sys, u, x0, dt, t, 'rk4_classico');...
```

Fonte: Elaborado pela autora (2025).

4) Chamada da função `metodos_v9.m`

Esta parte do código é responsável por simular a resposta temporal do sistema dinâmico (**sys**) usando diferentes métodos de integração numérica. A função **metodos_v9.m** atua como um gerenciador central que direciona a simulação para o método numérico selecionado pelo usuário.

Na interface gráfica final, a função **metodos_v9.m** é chamada uma única vez, de acordo com o método escolhido pelo usuário (Euler Explícito, Euler Implícito ou uma das variações de Runge-Kutta). No entanto, no exemplo didático apresentado no Código 3.3, a função é chamada múltiplas vezes com diferentes métodos para fins de comparação e validação do *software*.

Código 3.3 – Chamada dos métodos numéricos para comparação didática.

```
1 fprintf('\nMetodos disponiveis: explicit, implicit, rk2_ponto_medio,...')
   );
2 [y_explicit,x_explicit] = metodos_v9(sys, u, x0, dt, t, 'explicit');
3 [y_implicit,x_implicit] = metodos_v9(sys, u, x0, dt, t, 'implicit');
4 [y_rk_ponto_medio, x_rk_ponto_medio] = metodos_v9(sys, u, x0, dt, t, 'rk2_ponto_medio');
```

Fonte: Elaborado pela autora (2025).

Cada chamada retorna:

- `y_*`: Vetor de saída do sistema ao longo do tempo;
- `x_*`: Vetor de estados do sistema (evolução de $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$).

Estas variáveis armazenam os resultados para posterior análise comparativa, permitindo a geração de gráficos, tabelas e o cálculo de erros entre os diferentes métodos numéricos. No software operacional, apenas um conjunto de resultados é gerado por simulação, correspondente ao método selecionado pelo usuário na interface gráfica.

5) Comparação com a Função `lsim`

Para validação, os resultados foram comparados com a função nativa `lsim` (do inglês, *Linear Simulation*) do Octave. Esta função é utilizada para efetuar uma simulação da resposta de um sistema dinâmico linear a um sinal de entrada arbitrário (como degrau, rampa, senoidal, etc.), e permite que seja realizado o cálculo da saída do sistema (`y_lsim`) ao longo do tempo, a evolução dos estados (`x_lsim`) do sistema (no caso de um modelo em espaço de estados) e o vetor de tempo (`t_lsim`) utilizado na simulação. A sintaxe básica da função pode ser observada em Código 3.4:

Código 3.4 – Validação com a função `lsim`.

```
1 in_lsim = u(t);
2 [y_lsim, t_lsim, x_lsim] = lsim(sys, in_lsim, t);
```

Fonte: Elaborado pela autora (2025).

Sintaxe básica:

- `sys`: Sistema linear (definido por `ss`, `tf` ou `zpk`);
- `u (in_lsim)`: Sinal de entrada (deve ter o mesmo número de pontos que `t`);
- `t`: Vetor de tempo para a simulação;
- `x0` (opcional): Condições iniciais dos estados (para modelos em espaço de estados).

Na construção do código, a função `lsim` serviu como referência para validar os métodos numéricos implementados (Euler e Runge-Kutta), através da saída `y_sim`, comparando-a com os resultados dos métodos para fornecer uma solução precisa.

6) Geração Automatizada de Tabelas de Resultados

O código 3.5 produz uma tabela comparativa organizando os resultados de todos os métodos implementados, através da geração automatizada de tabelas de resultados, garantindo uma análise quantitativa precisa e organizada. Utilizando a função `table` do GNU Octave, os dados são estruturados em colunas que comparam os resultados dos diferentes métodos numéricos (Euler explícito, Euler implícito, RK2, RK3, RK4).

Código 3.5 – Código parcial da construção da Tabela de Resultados.

```
1 % Arredondamento dos resultados para 8 casas decimais
2 n = 8;
3 x_explicit_rounded = round(x_explicit * 10^n)/10^n;
4 x_implicit_rounded = round(x_implicit * 10^n)/10^n;
5 x_rk_ponto_medio_rounded = round(x_rk_ponto_medio * 10^n)/10^n;
6
7 % Formatacao da tabela
8 results_table = table(t', x_explicit_rounded(1,:), x_implicit_rounded(1
    ,:)', x_rk_ponto_medio_rounded(1,:),...);
9 results_table.Properties.VariableNames = {'Tempo', 'Euler_Explicito', '
    Euler_Implicito', 'RK2_PMedio', ...};
10
11 for i = 1:size(results_table, 1)
12     fprintf('%-10.4f %-15.8f %-15.8f %-15.8f ... \n', ...
13         results_table.Tempo(i), ...
14         results_table.Euler_Explicito(i), ...
15         results_table.Euler_Implicito(i), ...
16         results_table.RK2_Ponto_Medio(i), ...
17 end
```

Fonte: Elaborado pela autora (2025).

A formatação é ajustada para exibir valores com 8 casas decimais, permitindo a identificação de discrepâncias sutis entre os métodos. Na Figura 11 demonstra-se um exemplo parcial da exibição da tabela de resultados:

Figura 11 – Exemplo de exibição da tabela de resultados.

Janela de Comandos			
Metodos disponiveis: explicit, implicit, rk2_ponto_medio,			
Metodo escolhido: Euler Explcito			
Metodo escolhido: Euler Implicito			
Metodo escolhido: RK2 Ponto Medio			
Metodo escolhido: RK2 Ralston			
Metodo escolhido: RK2 Heun			
Metodo escolhido: RK3 Classico			
Metodo escolhido: RK3 Ralston			
Metodo escolhido: RK3 Heun			
Metodo escolhido: RK4 Classico			
Metodo escolhido: RK4 Ralston			
Metodo escolhido: RK4 Regra dos 3/8			
Resultados da Simulação:			
Tempo	Euler Expl	Euler Impl	RK2_PMedio
0.0000	0.00000000	0.00000000	0.00000000
0.0100	0.01000000	0.00969391	0.00985000
0.0200	0.01970000	0.01907655	0.01939323
0.0300	0.02909300	0.02814359	0.02862408
0.0400	0.03817293	0.03689158	0.03753783
0.0500	0.04693464	0.04531788	0.04613064
0.0600	0.05537386	0.05342068	0.05439952
0.0700	0.06348723	0.06119891	0.06234234
0.0800	0.07127224	0.06865230	0.06995775

Fonte: Elaborado pela autora (2025).

7) Análise e Visualização Gráfica do Sistema

O Código 3.6 gera gráficos sobrepostos para comparar visualmente a entrada ao degrau ($u(t)$), as soluções numéricas (neste exemplo consta Euler explícito e implícito) e a referência **lsim**. O objetivo desta implementação é identificar erros de regime permanente ou atrasos de fase nos métodos numéricos e validar a precisão de métodos de alta ordem (ex.: RK4) contra **lsim**.

Código 3.6 – Código parcial da visualização gráfica do sistema.

```

1 %Plotagem de Resposta de Saida dos Metodos e Lsim
2 subplot(2, 1, 1);
3 hold on;
4 plot(t, u(t), 'LineWidth', 3, 'Color', [0.8500 0.3250 0.0980]); %degrau
5 plot(t, y_explicit, 'c--', 'LineWidth', 2); % Saida do Euler Explcito
6 plot(t, y_implicit, 'm--', 'LineWidth', 2); % Saida do Euler Implicito
7 plot(t_lsim, y_lsim, 'b--', 'LineWidth', 2); % Saida do lsim
8
9 hold off;
10 title('Comparacao da Entrada ao Degrau e das Saidas Y dos metodos e Lsim

```

```

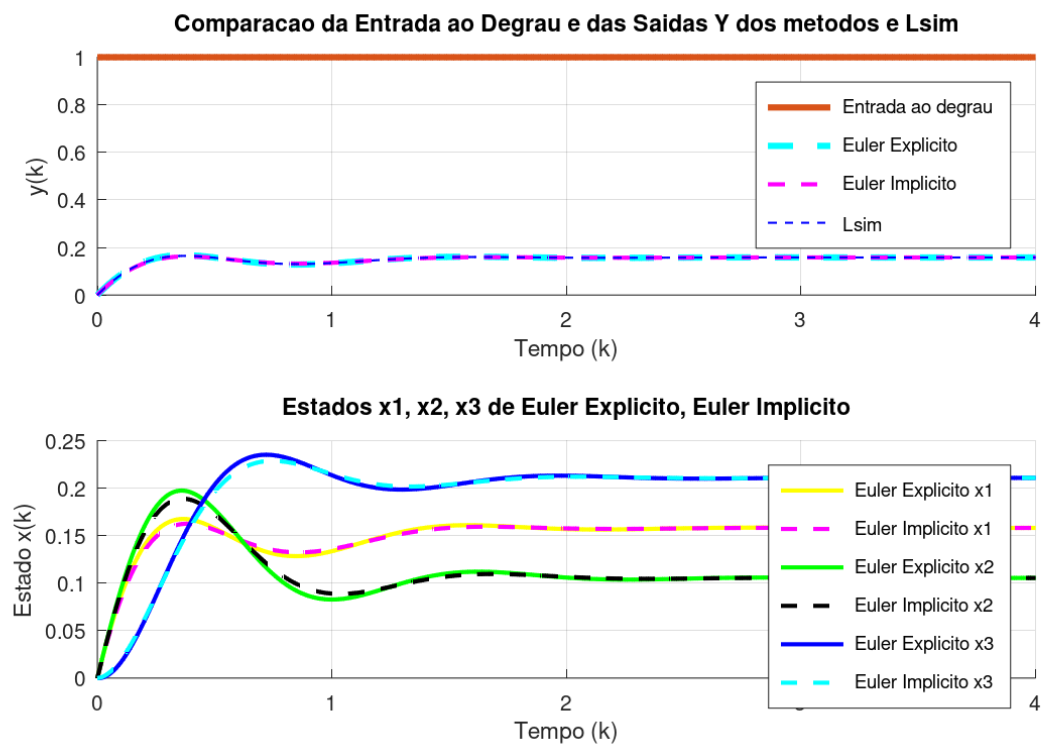
    ');
11 xlabel('Tempo (k)');
12 ylabel('y(k)');
13 legend('Entrada ao degrau','Euler Explícito', 'Euler Implícito', ..., '
    Lsim');
14 grid on;
15
16 % Plotagem Evolucao dos Estados
17 subplot(2, 1, 2);
18 hold on;
19 plot(t, x_explicit(1, :), 'y', 'LineWidth', 2); % Estado x1
20 plot(t, x_implicit(1, :), 'm--', 'LineWidth', 2);
21 plot(t, x_explicit(2, :), 'g', 'LineWidth', 2); % Estado x2
22 plot(t, x_implicit(2, :), 'k--', 'LineWidth', 2);
23 plot(t, x_explicit(3, :), 'b', 'LineWidth', 2); % Estado x3
24 plot(t, x_implicit(3, :), 'c--', 'LineWidth', 2); ...
25
26 hold off;
27 title('Estados x1, x2, x3 de Euler Explícito, Euler Implícito, ...');
28 xlabel('Tempo (k)');
29 ylabel('Estado x(k)');
30 legend('Euler Explícito x1', 'Euler Implícito x1', ...
31        'Euler Explícito x2', 'Euler Implícito x2',...
32        'Euler Explícito x3', 'Euler Implícito x3'...);
33 grid on;

```

Fonte: Elaborado pela autora (2025).

O segundo gráfico exibido plota a trajetória dos estados do sistema (x_1, x_2, x_3) para cada método, usando cores e estilos de linha distintos para cada método, permitindo analisar a dinâmica interna do sistema. Através desta análise é possível observar como os estados evoluem, mesmo quando a saída $(y(t))$ parece similar entre métodos e averiguar a estabilidade, pois estados divergentes indicam instabilidades numéricas (ex.: Euler explícito com passo grande). O resultado da implementação parcial pode ser observado na Figura 12 exemplo:

Figura 12 – Visualização Gráfica da simulação implementada.



Fonte: Elaborado pela autora (2025).

9) Diagramas de análise dinâmica

Para complementar a análise no domínio do tempo e fornecer uma caracterização abrangente do sistema, o código incorpora um conjunto de ferramentas clássicas de análise de sistemas dinâmicos, conforme detalhado na Tabela 3:

Tabela 3 – Objetivos e aplicações dos diagramas de análise dinâmica.

Diagrama	Objetivo e Aplicação
Bode	Avalia a resposta em frequência do sistema, identificando ganho em dB e fase em graus em função da frequência. Essencial para análise de filtros e sintonia de controladores.
Nyquist	Verifica a estabilidade em malha fechada através do critério de Nyquist, particularmente útil para sistemas com atrasos ou não mínimos de fase.
Root Locus (Lugar das Raízes)	Estuda a migração dos polos de malha fechada em função da variação de ganho, permitindo projetar sistemas de controle robustos.
Pole-Zero Map (Mapa de Polos e Zeros)	Analisa a distribuição espacial dos polos (estabilidade) e zeros (transmissão direta) no plano complexo, revelando modos dominantes do sistema.

Fonte: Elaborado pela autora (2025).

Estes diagramas permitem investigar propriedades fundamentais do sistema que não são evidentes apenas na resposta temporal, incluindo estabilidade, resposta em frequência e sensibilidade paramétrica. A implementação destas análises, mostrada no Código 3.7, foi cuidadosamente estruturada para facilitar a interpretação.

Código 3.7 – Código da plotagem dos Diagramas de Análise.

```

1
2 figure('Name', 'Diagramas de Analise do Sistema', 'Position', [100 100
   1200 800]);
3
4 % 1. Diagrama de Bode
5
6 bode(sys); grid on; title('Bode');
7
8 % 2. Diagrama de Nyquist
9 subplot(2,2,1); nyquist(sys); grid on; title('Nyquist'); axis equal;
10
11 % 3. Root Locus
12 subplot(2,2,2); rlocus(sys); grid on; title('Root Locus');
13
14 % 4. Mapa de Polos e Zeros
15 subplot(2,2,3); pzmap(sys); grid on; title('Pole-Zero Map');
```

Fonte: Elaborado pela autora (2025).

Um exemplo de resultado obtido a partir da execução do Código 3.7 é mostrada na Figura 13:

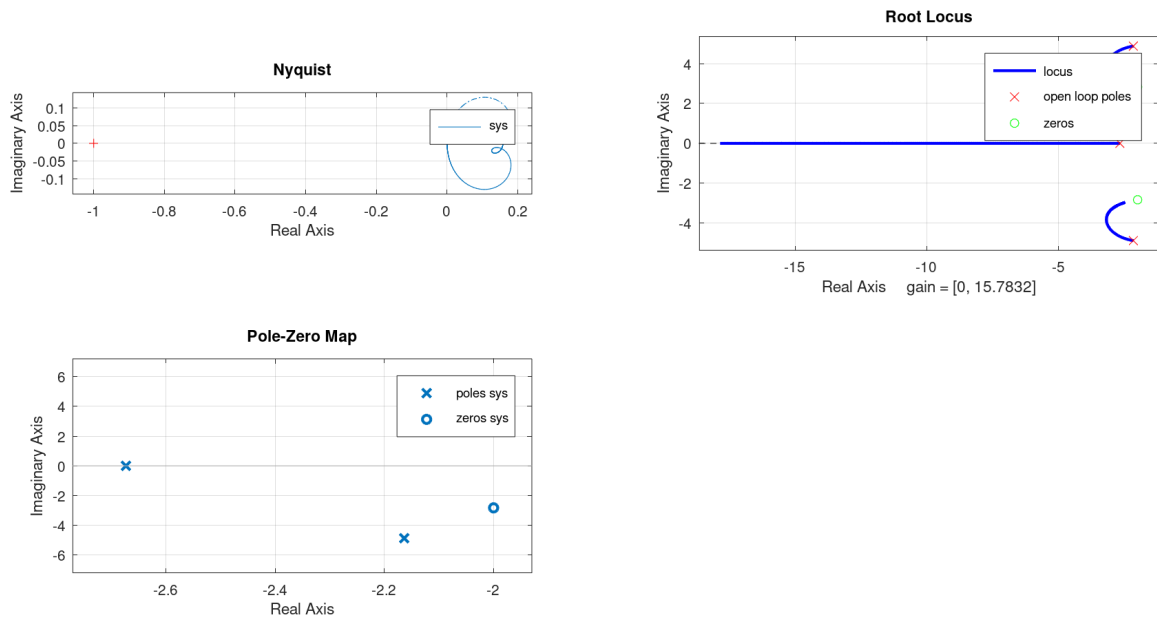
O código completo foi estruturado em seções claras, utilizando comentários detalhados para facilitar a manutenção e reprodução. A formatação dos resultados (arredondamento para 8 casas decimais) e a geração automática de gráficos garantem a robustez da análise. A construção total do código **main_v9.m** pode ser encontrada no Apêndice A.

Esta implementação serve como base para a seção seguinte, onde são detalhados os algoritmos dos métodos numéricos (**euler_explicit_v9**, **euler_implicit_v9**, **rk_v9**) e sua integração no código principal.

3.2.5 Implementação dos Métodos Numéricos

A construção do *software* iniciou-se com a implementação no GNU Octave, dos métodos numéricos básicos para solução de sistemas dinâmicos, priorizando uma abordagem matricial para garantir eficiência computacional e clareza na estruturação do código. O desenvolvimento iniciou através da implementação dos métodos de Euler Explícito e Implícito, escolhidos por sua simplicidade e importância didática.

Figura 13 – Plotagem dos diagramas de análise dinâmica.



Fonte: Elaborado pela autora (2025).

3.2.5.1 Implementação do Método de Euler Explícito Matricial

A implementação do método de Euler Explícito no espaço de estados foi desenvolvida visando eficiência computacional e clareza na estruturação do algoritmo. O código apresentado (**euler_explicit_v9.m**) segue uma abordagem matricial, aproveitando as propriedades de sistemas LIT, onde as matrizes **A**, **B**, **C** e **D** são constantes. A função recebe como entrada o sistema (**sys**), sendo este uma variável da classe **ss**, o sinal de entrada (**u**), sendo esta uma função definida pelo usuário, o estado inicial (**x0**), sendo um vetor de $n \times 1$, o passo de integração (**dt**) e o vetor de tempo (**t**), vetor de $1 \times n_2$, retornando a evolução temporal dos estados (**x_explicit**), um vetor de $n \times n_2$. No formato matricial, o Euler Explícito foi implementado conforme a equação encontrada na Equação (59) da Seção 2.7.1, como pode ser observado em Código 3.8:

Código 3.8 – Implementação matricial do método de Euler Explícito.

```

1 function [x_explicit] = euler_explicit_v9(sys, u, x0, dt, t)
2
3 [A, B, C, D] = ssdata(sys);
4
5 na = length(t);
6
7 x_explicit = zeros(size(A, 1), na + 1);
8
9 x_explicit(:, 1) = x0;
10

```

```

11     I = eye(size(A));
12
13     for k = 1:na
14
15         x_explicit(:, k+1) = (A * dt + I) * x_explicit(:, k) + dt * B *
16         u(t(k));
17     end
18 end

```

Fonte: Elaborado pela autora (2025).

O código executa as seguintes etapas:

1. **Extração das Matrizes do Sistema:** A função utiliza *ssdata(sys)* para obter as matrizes **A**, **B**, **C** e **D** da classe **ss**. Isso permite que o método seja aplicado a qualquer sistema modelado nesse formato, garantindo flexibilidade;
2. **Inicialização das Variáveis:**

- O número de iterações (**na**) é calculado através da função *length(t)*, onde *t* é o vetor de tempo definido para a simulação. Essa abordagem é essencial para que o método percorra todos os instantes de tempo fornecidos no vetor *t*;
- A matriz **x_explicit** é inicializada com zeros, onde as linhas correspondem ao número de estados (*n* dimensão de **A**, ou seja, *n*) e as colunas ao número de passos de tempo mais um (para incluir a condição inicial);
- O estado inicial **x0** é atribuído à primeira coluna de **x_explicit**;

3. **Iteração do Método de Euler:**

- A atualização do estado é realizada conforme a equação (84):

$$\mathbf{x}_{k+1} = (\mathbf{A} \cdot \Delta t + \mathbf{I}) \cdot \mathbf{x}_k + \Delta t \cdot \mathbf{B} \cdot \mathbf{u}_k \quad (84)$$

onde:

- $(\mathbf{A} \cdot \Delta t + \mathbf{I})$ é a discretização da matriz dinâmica para um passo Δt ;
- $\Delta t \cdot \mathbf{B}$ representa a discretização da matriz de entrada atual;
- A entrada $\mathbf{u}(t)$ deve ser avaliada no tempo t_k , ou seja, $\mathbf{u}(t_k)$, já que $\mathbf{u}$ é definida como uma função do tempo (e não um vetor pré-computado). Isso é essencial para manter a consistência matemática do método explícito.

3.2.5.2 Implementação do Método de Euler Implícito Matricial

Diferentemente do Euler Explícito, o método de Euler Implícito requer a valores futuros das entradas, garantindo maior estabilidade numérica. Conforme a Equação (64) encontrada na Seção 2.7.2, a implementação matricial (`euler_implicit_v9.m`) segue a estrutura apresentada em Código 3.9:

Código 3.9 – Implementação matricial do método de Euler Implícito.

```

1 function [x_implicit] = euler_implicit_v9(sys, u, x0, dt, t)
2
3 [A, B, C, D] = ssdata(sys);
4
5 na = length(t);
6
7 x_implicit = zeros(size(A, 1), na + 1);
8
9 x_implicit(:, 1) = x0;
10
11 I = eye(size(A));
12 inv_IAdt = inv(I - A * dt);
13
14 for k = 1:na-1
15     x_implicit(:, k+1) = inv_IAdt * x_implicit(:, k) + inv_IAdt * dt
16     * B * u(t(k+1));
17
18 end

```

Fonte: Elaborado pela autora (2025).

A seguir, informa-se as características da implementação:

1. Discretização Implícita:

A atualização do estado é dada por (85):

$$\mathbf{x}_{k+1} = (\mathbf{I} - \mathbf{A} \cdot \Delta t)^{-1} \cdot \mathbf{x}_k + (\mathbf{I} - \mathbf{A} \cdot \Delta t)^{-1} \Delta t \cdot \mathbf{B} \cdot \mathbf{u}_{k+1} \quad (85)$$

onde a inversão da matriz $(\mathbf{I} - \mathbf{A} \cdot \Delta t)$ pode ser crítica para sistemas de ordem elevada.

2. Passos Chave:

- **Inversão Matricial:** Calculada via $\text{inv}(\mathbf{I} - \mathbf{A} \cdot \Delta t)$, assegurando a solução exata do sistema implícito;

- **Entrada em $\mathbf{u}_{k+1}$:** Utiliza o valor da entrada no instante futuro (t_{k+1}), característica do método implícito, ou seja, $\mathbf{u}(t_{k+1})$. Note que o valor futuro é conhecido pois a função de entrada foi fornecida.

3. Vantagens:

- **Estabilidade:** Funciona bem para passos Δt maiores que os do Euler Explícito mantendo a estabilidade numérica (Esfandiari, 2017);
- **Precisão:** Reduz erros acumulados em sistemas de dinâmica rápida (Chapra; Canale, 2016).

3.2.5.3 Implementação dos Métodos de Runge-Kutta através da Tabela de Butcher

A implementação dos métodos de Runge-Kutta (RK2, RK3 e RK4) foi desenvolvida de forma genérica e modular através da utilização de tabelas de Butcher, apresentadas na Seção 2.9, que encapsulam os coeficientes característicos de cada método (estágios a_{ij} , pesos b_i e nós c_i). Essa abordagem permite a seleção dinâmica do método desejado (**metodo_rk**) sem alterações na estrutura principal do código, garantindo flexibilidade e reutilização. O código apresentado (**rk_v9.m**) segue três etapas principais:

1. Estrutura das Tabelas de Butcher:

As tabelas de Butcher são armazenadas em estruturas (*struct*) separadas por ordem do método (RK2, RK3, RK4), contendo:

- Vetor de nós (c): Vetor de nós que define os instantes de tempo relativos para cálculo de cada estágio (ex: $[0; 1/2]$ no RK2 Ponto Médio);
- Matriz de coeficientes (a): Matriz triangular inferior que especifica como as derivadas de estágios anteriores influenciam os estágios subsequentes (ex: $\begin{bmatrix} 0 & 0 \\ 1/2 & 0 \end{bmatrix}$ no RK2 Ponto Médio);
- Vetor de pesos (b): Vetor de pesos para combinação linear dos estágios na atualização final do estado (ex: $[0 \quad 1]$ no RK2 Ponto Médio).

O desenvolvimento da *struct* pode ser observado no código parcial 3.10 a seguir:

Código 3.10 – Implementação da estrutura da Tabela de Butcher para RK2 Ponto Médio.

```

1 function [x_rk] = rk_v9(sys, u, x0, dt, t, metodo_rk)
2
3 [A, B, C, D] = ssdata(sys);
4
5 na = length(t);
6
7 x_rk = zeros(size(A, 1), na + 1);
8
9 x_rk(:, 1) = x0;
```

```

10
11 % Estrutura da tabela de butcher para RK2 Ponto Medio
12 butcher_table = struct();
13 butcher_table.ponto_medio = struct(...
14     'c', [0;
15           1/2], ...
16     'a', [0    0;
17           1/2  0], ...
18     'b', [0 1]);

```

Fonte: Elaborado pela autora (2025).

2. Cálculo dos Estágios e Atualização do Estado:

Para cada método, os estágios ($m_1, m_2, \dots$) são calculados sequencialmente, utilizando a matriz do sistema (A, B) e a entrada $u(t)$ no tempo apropriado ($t(k) + c_i \cdot dt$). A função *isfield* é utilizada no código para verificar dinamicamente se o método de Runge-Kutta selecionado (**metodo_rk**) existe nas tabelas de Butcher predefinidas (**butcher_table**, **butcher_table_rk3** ou **butcher_table_rk4**). A sintaxe da função é definida como *isfield*(estrutura, campo), onde ela retorna 1 (verdadeiro) se campo existir na estrutura, ou 0 (falso) caso contrário. Essa verificação é essencial para garantir que o código execute apenas métodos válidos e evite erros. A atualização do estado combina os estágios conforme os pesos b_i , esta implementação é apresentada em 3.11:

Código 3.11 – Implementação do cálculo dos estágios e atualização do estado para RK2 Ponto Médio.

```

1  if isfield(butcher_table, metodo_rk)
2
3      % Extracao dos coeficientes da tabela de Butcher
4      c = butcher_table.(metodo_rk).c;
5      a = butcher_table.(metodo_rk).a;
6      b = butcher_table.(metodo_rk).b;
7
8      for k = 1:na
9          % Calculo dos estagios (k1, k2) em t(k)
10         m1 = A * x_rk(:, k) + B * u(t(k));
11         m2 = A * (x_rk(:, k) + a(2,1) * m1 * dt) + B * u(t(k) + c(2)
12         * dt);
13
14         % Atualizacao do estado
15         x_rk(:, k+1) = x_rk(:, k) + (b(1) * m1 + b(2) * m2) * dt;
16     end

```

Fonte: Elaborado pela autora (2025).

A implementação dos métodos de Runge-Kutta de terceira (RK3) e quarta ordem

(RK4) segue uma estrutura análoga à do RK2, mantendo o mesmo padrão modular baseado nas tabelas de Butcher. A principal diferença reside no número de estágios calculados, mas a lógica permanece idêntica.

Em termos de comparação, no RK2, calculam-se dois estágios ($m1, m2$), enquanto para o método RK3, acrescenta-se um terceiro estágio ($m3$), e no RK4, um quarto ($m4$), sempre seguindo os coeficientes a_{ij} e nós c_i da tabela de Butcher. Em relação a atualização do estado, a combinação linear dos estágios (com pesos b_i) é estendida para incluir mais termos, mas a operação matricial básica se mantém.

O desenvolvimento do código também fornece flexibilidade, pois ele aproveita a mesma estrutura de seleção (*if-elseif*) e vetorização, apenas expandindo o número de operações conforme a ordem do método. Essa padronização reduz a complexidade de implementação e facilita a comparação entre métodos.

3. Vantagens da Implementação:

A implementação apresenta quatro vantagens principais:

- (1) **Generalização:** Um único código resolve métodos de RK2 a RK4 mediante troca da tabela de Butcher;
- (2) **Eficiência:** Utiliza operações matriciais nativas do Octave/MATLAB;
- (3) **Clareza:** Estrutura espelha diretamente a formulação matemática;
- (4) **Extensibilidade:** Novos métodos são adicionados simplesmente incluindo suas tabelas.

Essa abordagem unifica precisão numérica, desempenho computacional e flexibilidade para análise de sistemas dinâmicos. Para o desenvolvimento deste trabalho, foram escolhidos 3 tipos de métodos para cada tipo de ordem superior, sendo eles:

1. **RK2:** Ponto Médio (*Midpoint*), Ralston e Heun;
2. **RK3:** Clássico, Ralston e Heun;
3. **RK4:** Clássico, Ralston e 3/8-*Rule*.

A implementação completa dos métodos de Runge-Kutta efetuada no *software* GNU Octave encontra-se no Apêndice B. De forma similar, todas as tabelas de Butcher também utilizadas na implementação encontram-se listadas no Anexo A.

3.2.5.4 Implementação do Código `metodos_v9.m`

A função `metodos_v9.m` implementa diversos métodos numéricos para solução de sistemas dinâmicos lineares, com seleção flexível via parâmetro `metodo` (uma *string* que especifica o algoritmo desejado). Sua saída consiste nos vetores de estado (x) e saída (y) do sistema ao longo do tempo. Como estrutura da função apresenta:

1. **Seleção do Método:** Utiliza uma estrutura condicional *switch-case* para direcionar a execução ao método numérico especificado, garantindo modularidade e extensibilidade;
2. **Cálculo das Saídas:** Após a obtenção dos estados x , calcula as saídas y por meio da equação $y = Cx + Du$, onde C e D são matrizes do sistema no espaço de estados.

A função também conta com uma validação robusta, pois caso o método selecionado seja inválido, ela retorna um erro listando todas as opções disponíveis, facilitando a depuração. Como cálculo de saída a função apresenta y , vetor de saída do sistema derivado da combinação linear $y = Cx + Du$ e x que representa a matriz de estados calculada pelo método numérico selecionado. O código parcial pode ser observado e Código 3.12:

Código 3.12 – Código parcial metodos_v9.m.

```

1 function [y, x] = metodos_v9(sys, u, x0, dt, t, metodo)
2     switch metodo
3
4         case 'explicit'
5             fprintf('Metodo escolhido: Euler Explcito\n');
6             x = euler_explicit_v9(sys, u, x0, dt, t);
7
8         case 'implicit'
9             fprintf('Metodo escolhido: Euler Implicito\n');
10            x = euler_implicit_v9(sys, u, x0, dt, t);
11            ...
12
13        otherwise
14            error(['Metodo desconhecido. Escolha entre: \n'...
15                'Euler: "explicit", "implicit" \n'...
16                'RK2: "rk2_ponto_medio", "rk2_ralston", "rk2_heun" \n'...
17                'RK3: "rk3_classico", "rk3_ralston", "rk3_heun" \n'...
18                'RK4: "rk4_classico", "rk4_ralston" ou "rk4_38rule".']);
19    end
20
21    ...
22
23    in = u(t);
24
25    [A, B, C, D] = ssdata(sys);
26    na = length(t);
27    x = x(:, 1:na);
28
29    y = C * x + D * in;
30 end

```

Fonte: Elaborado pela autora (2025).

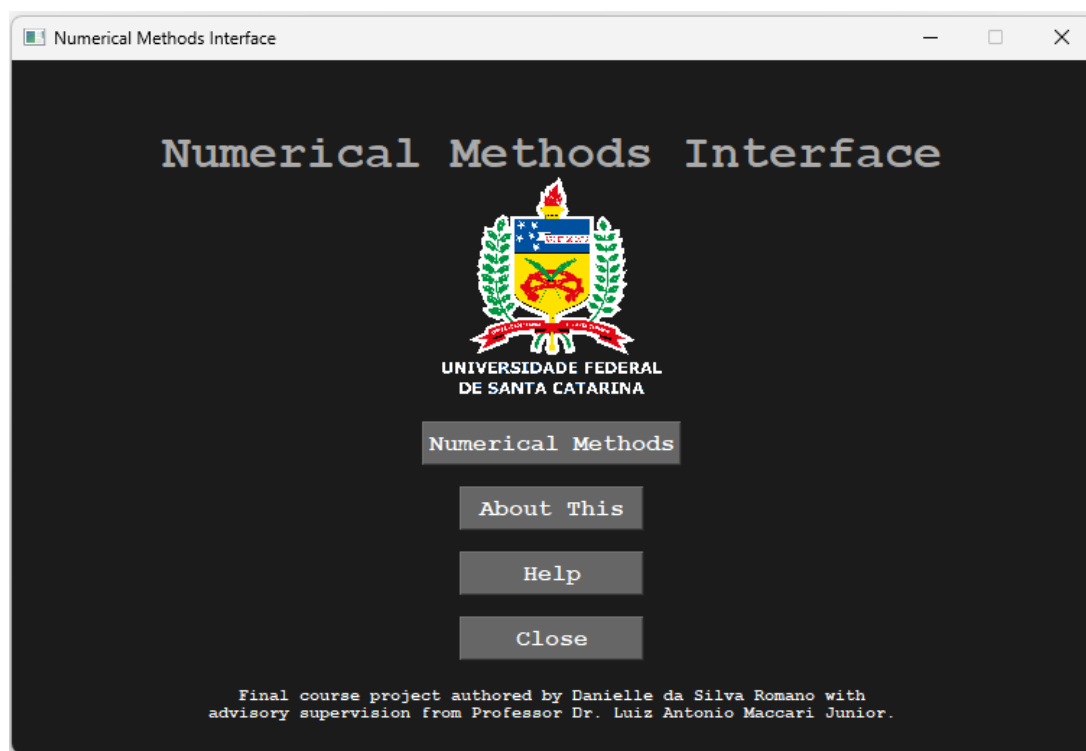
Esta implementação unificada permite a comparação sistemática entre diferentes abordagens numéricas para integração de sistemas dinâmicos, assegurando consistência na análise e redução de código redundante. A função completa pode ser encontrada no Apêndice C.

3.2.6 Programação Interface

A implementação da interface gráfica no GNU Octave foi estruturada para integrar métodos numéricos avançados com usabilidade didática, seguindo um fluxo modular. A programação combinou as bibliotecas matemáticas nativas do Octave com o guiEditor, ferramenta dedicada à criação de interfaces gráficas (GUIs), garantindo interatividade e visualização intuitiva dos resultados.

O desenvolvimento da interface gráfica iniciou-se com a elaboração de um esboço detalhado no guiEditor do GNU Octave, definindo a disposição dos elementos visuais, como botões, campos de entrada e áreas de plotagem. Esta etapa foi crucial para planejar a usabilidade do *software*, garantindo que a interface fosse intuitiva e atendesse aos requisitos pedagógicos. Os principais componentes, como os painéis para inserção das matrizes de estado (**A**, **B**, **C**, **D**) e os parâmetros de simulação (**t0**, **tf**, **dt**, **x0**), foram organizados de forma lógica, enquanto os gráficos para visualização dos resultados foram posicionados em destaque. Decidiu-se por uma interface com 3 telas principais:

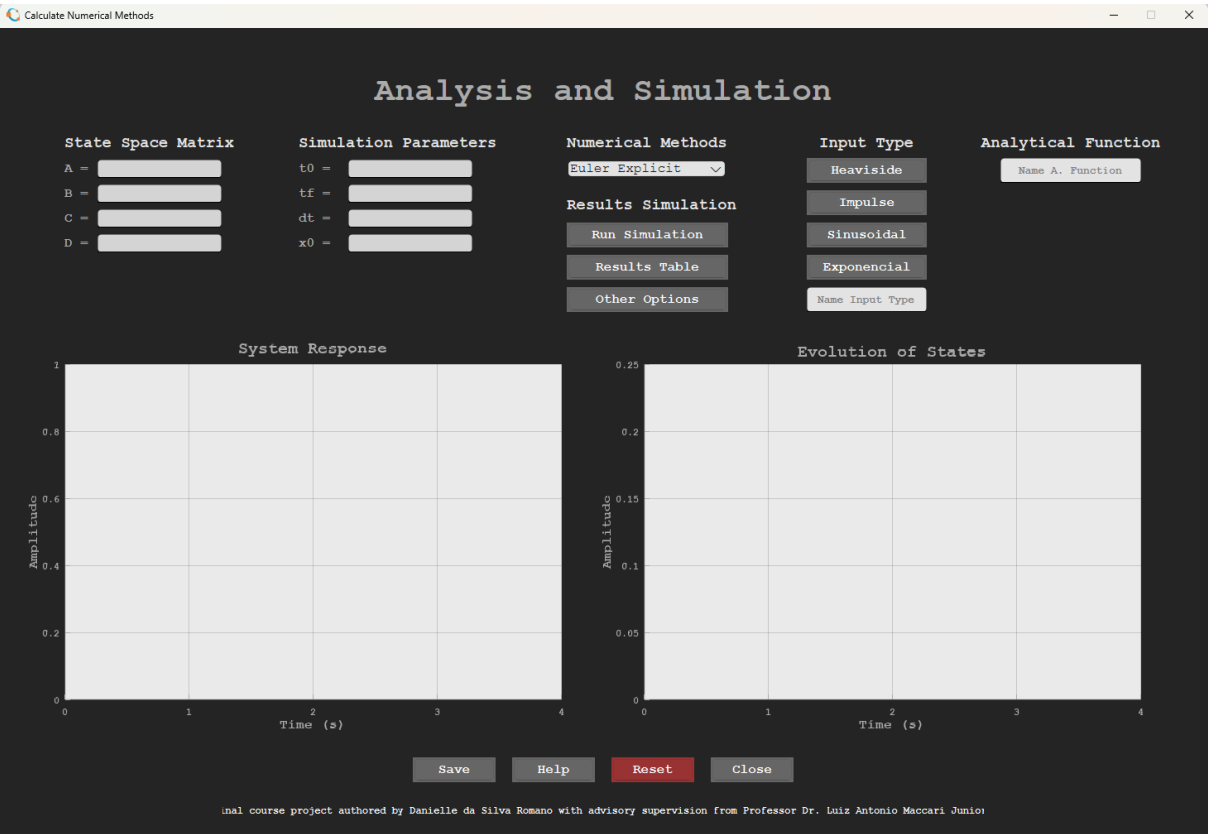
- Tela "*Numerical Methods Interface*": Interface inicial ao carregar o *software*.

Figura 14 – Tela *Numerical Methods Interface*.

Fonte: Elaborado pela autora (2025).

- Tela "Analysis and Simulation": Interface principal do *software*.

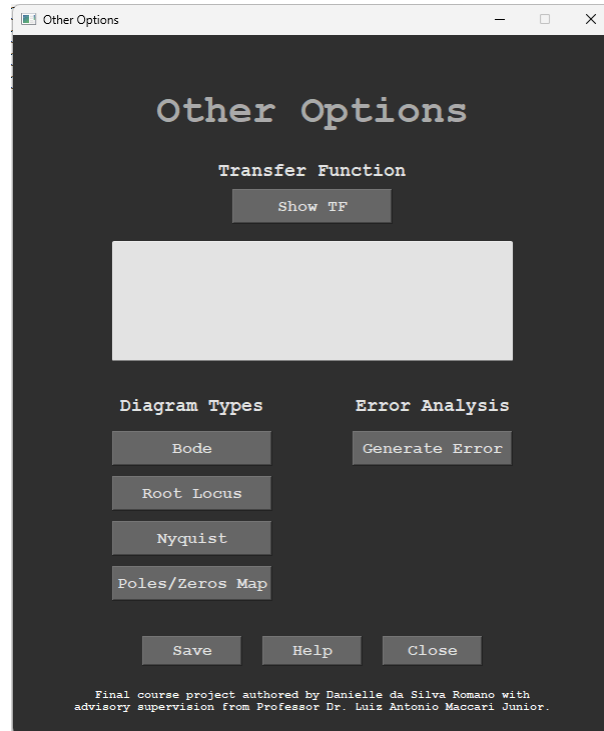
Figura 15 – Tela *Analysis and Simulation*.



Fonte: Elaborado pela autora (2025).

- Tela "Other options": Tela adicional do *Other options*.

Figura 16 – Tela adicional do *software*.



Fonte: Elaborado pela autora (2025).

Com o esboço das interfaces definido, o próximo passo foi implementar os códigos dos métodos numéricos previamente desenvolvidos (Euler, Runge-Kutta, etc.) e integrá-los às funcionalidades da GUI. Através de *callbacks*, os botões da interface foram vinculados às funções de cálculo, permitindo que a simulação fosse executada automaticamente após a entrada dos parâmetros. Por exemplo, ao clicar em "*Run Simulation*", os dados inseridos são validados, o método numérico selecionado é aplicado e os resultados são exibidos em tempo real nos gráficos. Essa abordagem garantiu que as simulações ocorressem de maneira fluida e precisa, facilitando a comparação entre diferentes técnicas numéricas.

A combinação do planejamento visual no guiEditor com a implementação robusta dos métodos numéricos resultou em uma ferramenta didática eficiente, capaz de auxiliar no ensino de sistemas dinâmicos de forma interativa e visualmente intuitiva.

Para a construção completa da interface gráfica, foram desenvolvidas 21 funções no total, cada uma responsável por uma parte específica do *software*, desde a inicialização das telas até a execução dos métodos numéricos e geração de resultados. Essa modularização permitiu uma organização clara do código, facilitando a manutenção e a possibilidade de futuras expansões. Dentre essas funções, destacam-se:

- Funções de Interface: Como **mostrar_TelaInicial.m**, **mostrar_TelaSecun-**

daria.m, **mostrar_Tela_OtherOptions.m** que gerenciam a apresentação e interação do usuário com o *software*;

- Funções de Cálculo: Implementações dos métodos numéricos (Euler, Runge-Kutta) e funções auxiliares para validação de dados;
- Funções de Saída: Geração de gráficos, tabelas e exportação de resultados.

A seguir, será detalhada a função **mostrar_TelaInicial.m**, que configura a primeira interação do usuário com o *software*, um resumo das funções **mostrar_TelaSecundaria.m**, responsável pela simulação e análise dos sistemas dinâmicos e **mostrar_Tela_OtherOptions.m**, responsável por trazer opções avançadas para a análise dos sistemas.

3.2.6.1 Detalhamento da Função *mostrar_TelaInicial.m*

A função *mostrar_TelaInicial.m* estabelece a base da interface gráfica e é responsável por criar a tela de abertura do *software*, seguindo um padrão de organização que será replicado nas demais telas (Figura 14). Esta função apresenta o título, o brasão institucional e os botões de acesso às demais funcionalidades, com os seguintes elementos principais:

1. Configuração Inicial:

- Fechamento de janelas anteriores (*close all force*) para evitar sobreposição;
- Cálculo dinâmico do diretório de imagens (*currentDir* e *imgDir*), garantindo portabilidade;
- Carregamento do brasão da UFSC com validação de arquivo (*isfile*), crítico para a identidade visual do projeto.

2. Personalização da Janela Principal:

- Centralização automática: A janela é posicionada no centro da tela, calculando dinamicamente as coordenadas (*xPos*, *yPos*) com base no tamanho do monitor (*screenWidth*, *screenHeight*);
- Estilo visual profissional: Utilização de cores escuras ([0.106 0.106 0.106]) e fontes técnicas (Courier New), alinhadas ao tema "profissional técnico" (Johnson, 2020);
- Layout consistente: Desativação de redimensionamento (*'Resize'*, *'off'*) para manter a organização visual.

3. Elementos Visuais e Interação:

- Brasão da UFSC: Exibido através de *axes* com posicionamento preciso (*imgXPos*, *imgYPos*), carregado a partir de um arquivo de imagem com validação de existência para evitar erros;

- Botões de Navegação:
 - **Numerical Methods:** Vinculado a um *callback* direto (`@(~, ~) abrirTelaSecundaria(TelaInicial)`), demonstrando a arquitetura modular e permitindo acesso à tela secundária de simulação;
 - **About This e Help:** Fornecem informações adicionais sobre o *software*;
 - **Close:** Encerra a aplicação de forma controlada.

4. Créditos:

Exibição das informações da autora e do orientador, reforçando a natureza acadêmica do projeto.

A função segue um padrão claro de organização e reusabilidade, com configurações visuais centralizadas e *callbacks* bem definidos. A validação de recursos (como imagens) garante robustez, enquanto a centralização dinâmica melhora a experiência do usuário em diferentes resoluções de tela. Os elementos gráficos e botões são posicionados de forma lógica, facilitando a navegação e o uso intuitivo do *software*. No Código 3.13 é possível verificar um trecho do código da implementação desenvolvida:

Código 3.13 – Código parcial da função `mostrar_TelaInicial.m`.

```

1 Button_NumericalMethods = uicontrol('parent', TelaInicial, ...
2                               'Style', 'pushbutton', ...
3                               'Units', 'pixels', ...
4                               'BackgroundColor', [0.400 0.400 0.400]
5                               , ...
6                               'FontAngle', 'normal', ...
7                               'FontName', 'Courier New', ...
8                               'FontSize', 12, ...
9                               'FontWeight', 'bold', ...
10                              'ForegroundColor', [1 1 1], ...
11                              'Position', [mainButtonXPos
mainButtonYPos mainButtonWidth buttonHeight], ...
12                              'String', 'Numerical Methods', ...
                              'Callback', @(~, ~)
                              abrirTelaSecundaria(TelaInicial));

```

Fonte: Elaborado pela autora (2025).

3.2.6.2 Detalhamento da Função `mostrar_TelaSecundaria.m`

A função `mostrar_TelaSecundaria` gerencia a tela principal de simulação, onde o usuário interage com os métodos numéricos e visualiza os resultados (Figura 15). Esta tela foi cuidadosamente estruturada para oferecer uma experiência intuitiva e eficiente,

combinando elementos gráficos avançados com uma lógica de programação robusta. Seus principais componentes são:

Estrutura e Componentes-Chave

1. Áreas de Plotagem

- Dois eixos gráficos (axes) para visualização simultânea:
 - *System Response*: Exibe a resposta temporal do sistema;
 - *Evolution of States*: Mostra a evolução dos estados ao longo do tempo.
- Personalização com *grids*, *labels* em Courier New e cores temáticas para clareza visual.

2. Entrada de Dados

- **Matrizes de Estado**: Campos editáveis (*uicontrol* 'Style', 'edit') para as matrizes A, B, C, D;
- **Parâmetros de Simulação**: *Inputs* para tempo inicial (t0), final (tf), passo (dt) e condições iniciais (x0).

3. Seleção de Métodos Numéricos

- Menu suspenso (*dropdown*) com opções como:
 - Euler Explícito/Implícito;
 - *Runge-Kutta (RK2, RK3, RK4)*;
 - Métodos clássicos e variações (Ralston, Heun).

4. Botões de Ação

- *Run Simulation*: Vinculado a *calculateCallback*, que:
 - a) Lê os parâmetros inseridos;
 - b) Executa o método numérico selecionado;
 - c) Plota os resultados em tempo real.
- *Results Table* e *Other Options*: Geram tabelas de dados e opções avançadas.

Callbacks e Interatividade

• Input Types:

- *Callbacks* como *heavisideCallback* e *sinusoidalCallback* configuram entradas pré-definidas (degrau, senoidal, impulso). É possível observar o exemplo da implementação no Código 3.14.

Código 3.14 – Implementação da função *sinusoidalCallback*.

```
1 function sinusoidalCallback(~, ~, handles)
```

```

2   prompt = {'Amplitude:', 'Frequency (Hz):'};
3   answer = inputdlg(prompt, 'Sinusoidal Parameters', [1 35; 1 35], {'1', '0.5'});
4   if ~isempty(answer)
5       params.amplitude = str2double(answer{1});
6       params.frequency = str2double(answer{2});
7       setappdata(handles.TelaSecundaria, 'input_type', 'sinusoidal');
8       setappdata(handles.TelaSecundaria, 'input_params', params);
9   end
10 end

```

Fonte: Elaborado pela autora (2025).

- Exportação e Utilitários:

- *Save*: Salva gráficos em formatos como JPG, FIG ou PNG (*saveFigures*);
- *Help* e *Close*: Botões padronizados com *callbacks* reutilizáveis.

Organização do Código

- Estrutura *handles*:

- Armazena todos os componentes da interface (ex: *handles.graphAxes1*, *handles.edit_a*), facilitando acesso e modificações em *callbacks*. Exemplo de uso em Código 3.15:

Código 3.15 – Código parcial do uso de *handles* na função.

```

1 handles.edit_a = uicontrol('Style', 'edit', 'Tag', 'edit_a', ...);
2 guidata(TelaSecundaria, handles); % Salva a estrutura

```

Fonte: Elaborado pela autora (2025).

- Validação de Dados:

- Verificação de *inputs* numéricos e tratamento de erros para evitar falhas na simulação.

A tela secundária representa um modelo de integração eficaz entre a interface gráfica e os métodos numéricos, combinando três qualidades fundamentais: flexibilidade, interatividade e robustez. Em primeiro lugar, oferece flexibilidade ao suportar múltiplos métodos de cálculo (como Euler e Runge-Kutta) e diversos tipos de entrada (degrau, senoidal, impulso), adaptando-se a diferentes cenários de simulação. Em segundo lugar, destaca-se pela interatividade, apresentando diálogos dinâmicos para configuração de parâmetros e proporcionando *feedback* visual imediato através de gráficos atualizados em tempo real. Por fim, demonstra robustez através de mecanismos de validação de dados que previnem erros e de uma estrutura modular que facilita a inclusão de novas funcionalidades.

Essa combinação de características resulta em uma ferramenta versátil e confiável para análise de sistemas dinâmicos, capaz de atender tanto às necessidades didáticas quanto às exigências técnicas do projeto.

3.2.6.3 Detalhamento da Função `mostrar_Tela_OtherOptions.m`

A função `mostrar_Tela_OtherOptions` implementa uma tela de opções avançadas para análise de sistemas dinâmicos, complementando as funcionalidades principais da interface (Figura 16). Seguindo o mesmo padrão das demais telas, ela combina organização visual e recursos técnicos especializados:

Estrutura e Componentes Principais

- Configuração da Janela:
 - Dimensões fixas (600x700 pixels) e centralização automática na tela;
 - Estilo visual alinhado ao tema do *software* (fundo escuro [0.184 0.184 0.184], fontes Courier New).
- Seções Organizadas:
 - Função de Transferência:
 - * Botão “*Show TF*” para exibir a função de transferência convertida das matrizes de estado (via `ss2tf`);
 - * Área de texto formatada para visualização clara (com divisores e centralização).
 - Tipos de Diagramas:
 - * Botões para geração de diagramas clássicos: Bode, Root Locus, Nyquist, Mapa de Polos/Zeros;
 - * Cada diagrama é plotado em uma nova janela com *grids* e títulos descritivos.
 - Análise de Erros:
 - * Botão “*Generate Error*” vinculado a uma tela específica para avaliação de erros.
 - Utilitários:
 - * Botões padrão (*Save*, *Help*, *Close*) com *callbacks* reutilizáveis.
 - * Exibição de créditos acadêmicos.

Funcionalidades Destacadas

- Conversão para Função de Transferência:
 - Acessa as matrizes de estado (A, B, C, D) da tela principal (`tela_secundaria`);

- Converte para forma polinomial usando **ss2tf** e formata a exibição (ex: $(s + 1)/(s^2 + 2s + 3)$);
 - Inclui tratamento de erros para dados ausentes ou inválidos.
- Plotagem de Diagramas:
 - Bode, Root Locus, Nyquist, PZMap;
- Integração com a Tela Principal:
 - Utiliza *getappdata* para acessar parâmetros compartilhados;
 - Mantém consistência visual e operacional com as demais telas.

Padrões de Implementação

- Callbacks Modularizados:
 - Cada botão chama funções dedicadas (ex: plotBodeDiagram, plotPZMap);
 - Uso de guidata para compartilhar a estrutura *handles*.
- Validação de Robustez:
 - Verificação de existência de dados antes de processar;
 - Mensagens de erro claras via `errordlg`.
- Formatação Profissional:
 - Textos centralizados e diagramas com labels em Courier New.

A seguir no Código 3.16 é possível observar a estrutura de um trecho de *callback* utilizado para plotagem do diagrama de bode na interface:

Código 3.16 – Exemplo Trecho de Callback na função.

```
1 function plotBodeDiagram(handles)
2     try
3
4         mainFig = findobj('Tag', 'tela_secundaria');
5         if isempty(mainFig)
6             error('Main window not found.');
```

```
6         end
7
8
9         A = getappdata(mainFig, 'A_matrix');
10        B = getappdata(mainFig, 'B_matrix');
11        C = getappdata(mainFig, 'C_matrix');
12        D = getappdata(mainFig, 'D_matrix');
```

```
13
14        sys = ss(A, B, C, D);
15
16        fig = figure('Name', 'Bode Diagram', 'NumberTitle', 'off', 'Tag',
    'fig_bode');
```

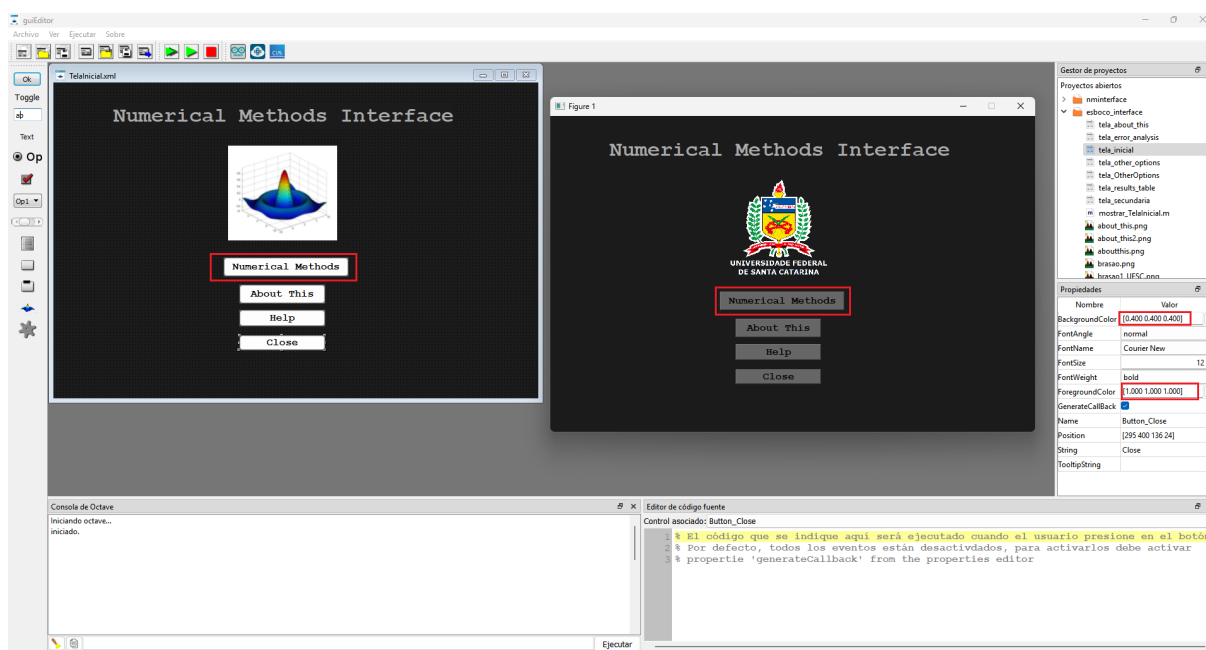
```
17
18     bode(sys);
19     grid on;
20     title('Bode Diagram');
21
22     catch ME
23         errordlg(['Error plotting Bode Diagram: ' ME.message], 'Error');
24     end
25 end
```

Fonte: Elaborado pela autora (2025).

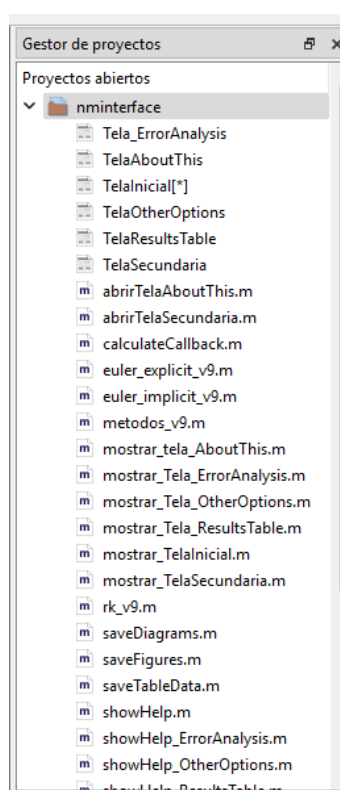
3.2.7 Integração guiEditor e GNU Octave

O desenvolvimento da interface gráfica iniciou-se com a criação das funções núcleo no GNU Octave, responsáveis pelos métodos numéricos e cálculos de sistemas dinâmicos. Estas funções foram posteriormente integradas ao guiEditor, ferramenta de criação de interfaces gráficas do Octave, onde foram definidos os *layouts* e elementos visuais. No entanto, devido às limitações do guiEditor em renderização de cores e flexibilidade de componentes (como pode ser observado na Figura 17), optou-se por implementar a interface diretamente via código, garantindo maior controle sobre o *design* e funcionalidades.

Para facilitar a distribuição, todas as funções e interfaces foram empacotadas em um pacote instalável (.pkg), ilustrado na Figura 18, permitindo que usuários finais possam adicionar a ferramenta ao seu Octave com um simples comando, sem necessidade de recompilação ou instalação manual de arquivos. Esta abordagem assegura que o *software* mantenha sua portabilidade e possa ser facilmente compartilhado como recurso educacional.

Figura 17 – Tela inicial da interface exibida no *Software* guiEditor.

Fonte: Elaborado pela autora (2025).

Figura 18 – Integração das funções no *Software* guiEditor.

Fonte: Elaborado pela autora (2025).

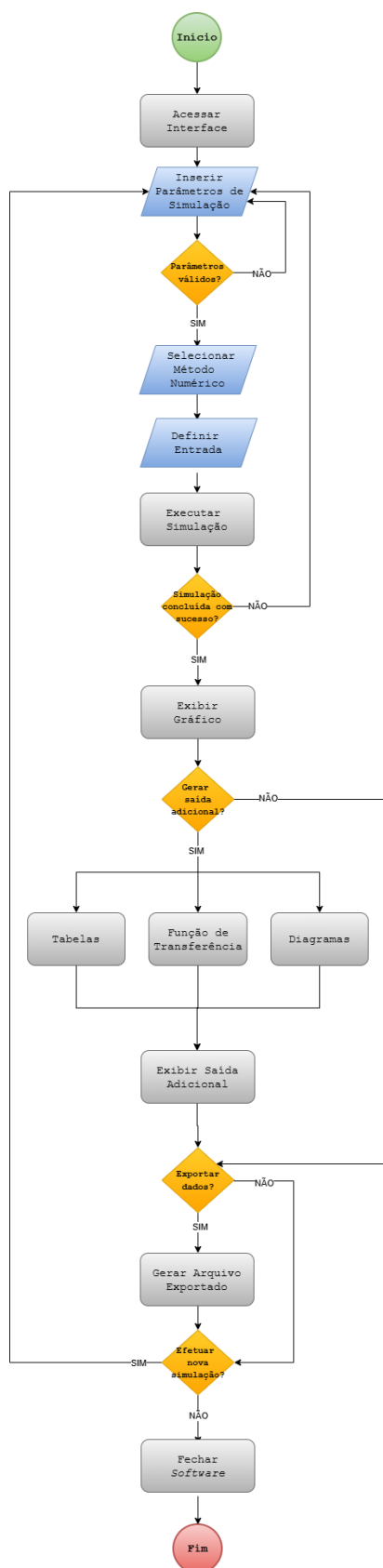
A combinação entre GNU Octave e guiEditor ofereceu vantagens significativas no desenvolvimento desta ferramenta. Em primeiro lugar, proporcionou maior flexibilidade na implementação de recursos, permitindo a criação de componentes personalizados além dos disponíveis no guiEditor. Em segundo lugar, garantiu precisão visual nos elementos da interface, assegurando que cores, fontes e *layouts* correspondessem exatamente ao projeto original, superando as limitações de renderização de cores e a rigidez de componentes observadas no uso exclusivo do editor gráfico. A facilidade de distribuição como pacote instalável simplifica a adoção por outros usuários, enquanto a integração otimizada entre funções numéricas e interface gráfica resultou em um *software* bem estruturado, onde cálculos complexos e visualizações interativas funcionam de forma sincronizada. Esses benefícios tornaram a ferramenta não apenas mais robusta tecnicamente, mas também mais acessível e adaptável às necessidades futuras.

3.3 FLUXOGRAMA DE OPERAÇÃO DA INTERFACE

O fluxograma apresentado na Figura 19 descreve o fluxo de operação do *software* desenvolvido. O processo inicia com o acesso à interface do usuário, onde são inseridos os parâmetros de simulação. Caso os parâmetros sejam inválidos, o sistema exibe uma mensagem de erro orientativa, solicitando correções antes de permitir a seleção do método numérico. Essa validação em tempo real previne inconsistências e otimiza o tempo do usuário.

Após a inserção dos parâmetros válidos, o usuário define o tipo de entrada do sistema, que inclui opções como impulso, senoidal, exponencial, degrau, vetor de entrada personalizado ou também uma função analítica personalizada. Essa flexibilidade atende tanto a aplicações didáticas quanto a demandas específicas de pesquisa. Uma vez preenchidos todos os campos necessários, a simulação é executada e os resultados são avaliados automaticamente. Se aprovados, podem ser visualizados em múltiplos formatos: gráficos dinâmicos, tabelas detalhadas ou diagramas de análise (Bode, Root Locus, Nyquist e Mapa de Polos e Zeros).

Além das etapas descritas, o fluxograma também incorpora mecanismos de validação em tempo real, assegurando que inconsistências nos dados de entrada sejam identificadas antes da execução da simulação. Por exemplo, caso o usuário introduza uma entrada incompatível com o sistema definido, o *software* exibe uma mensagem de erro orientativa, solicitando a revisão das configurações. Essa abordagem preventiva reduz a ocorrência de falhas durante a simulação e otimiza o tempo do usuário. Adicionalmente, após a geração dos resultados, o fluxograma prevê a opção de ajuste fino dos parâmetros (como tolerância de erro ou passo de integração), permitindo refinamentos iterativos sem necessidade de reiniciar todo o processo. A estrutura modular do fluxo, com retornos a etapas anteriores (como indicado pelas setas de "NÃO"), reforça a usabilidade do sistema, tornando-o acessível tanto para usuários iniciantes quanto para pesquisadores experientes.

Figura 19 – Fluxograma do *Software Numerical Methods Interface*.

Fonte: Elaborado pela autora (2025).

3.4 INSTALAÇÃO DO PACOTE NMINTERFACE.TAR.GZ NO GNU OCTAVE

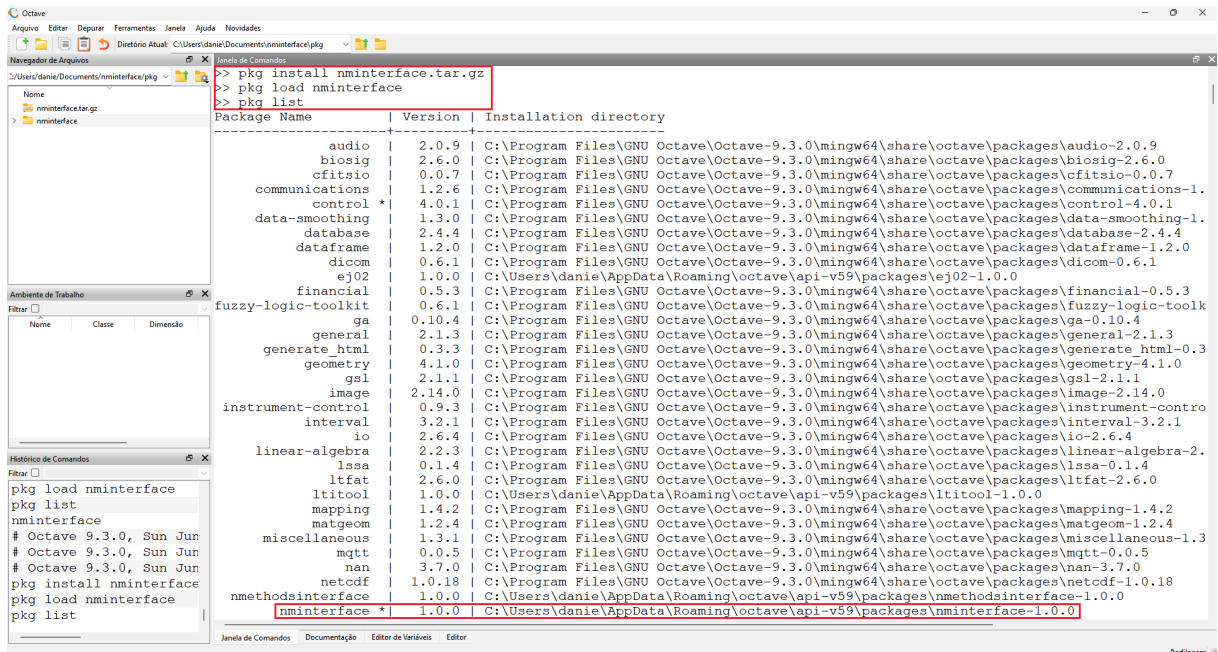
A instalação do pacote **nminterface.tar.gz** no GNU Octave para o sistema operacional *Windows* segue um processo estruturado, conforme ilustrado nas telas da Figura 20 e Figura 21. Primeiramente, o pacote é carregado através do comando **pkg install nminterface.tar.gz**, que realiza a extração e configuração dos arquivos necessários. Após a instalação, o pacote é listado na tabela de pacotes disponíveis, juntamente com sua versão (1.0.0) e o diretório de instalação (C:\Users\Administrador\AppData\Roaming\octave\api-v59\packages\nminterface-1.0.0). Essa localização indica que o pacote foi instalado no perfil do usuário, garantindo acesso personalizado sem exigir privilégios administrativos.

O Octave também gerencia dependências automaticamente, como evidenciado pela presença de outros pacotes essenciais (ex: control-4.0.1, statistics-1.7.0) nos diretórios padrão (C:\Program Files\GNU Octave\Octave-9.3.0\mingw64\share\octave\packages). Para verificar a instalação, o comando **pkg load nminterface** ativa as funcionalidades do pacote, integrando-o ao ambiente de trabalho. A presença de pacotes complementares:

1. **symbolic-3.2.1;**
2. **control-4.0.1;**
3. **tablicious-0.4.4;**
4. **signal-1.4.6;**
5. **optim-1.6.2.**

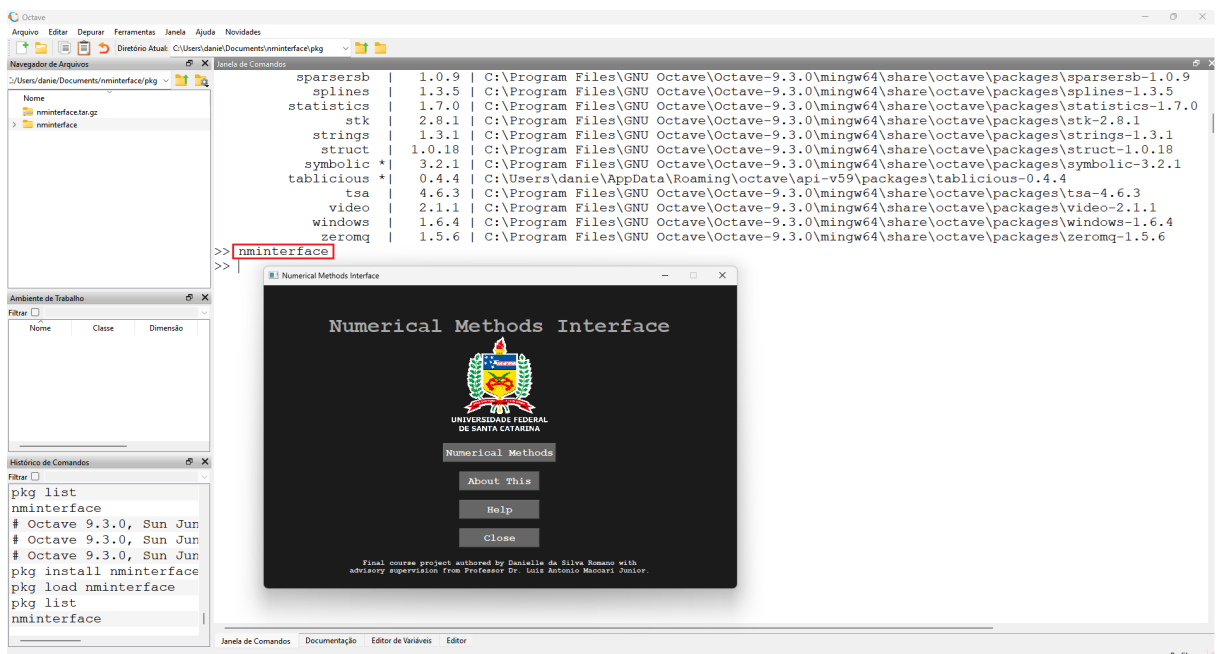
Reforça que o **nminterface** também depende de ferramentas adicionais para análises numéricas avançadas, as quais são pré-instaladas ou devem ser carregadas sob demanda, utilizando o comando **"pkg load" + "nome do pacote"**. Esse processo reflete a modularidade do Octave e a facilidade de extensão via pacotes, essencial para aplicações em métodos numéricos e simulações dinâmicas.

Para carregar a interface é necessário digitar como comando o nome da interface **nminterface**. Importante ressaltar que enquanto a interface permanece aberta não é possível efetuar interação com o *software* GNU Octave.

Figura 20 – Instalação do pacote do *Software*.

Fonte: Elaborado pela autora (2025).

Figura 21 – Carregamento da interface.



Fonte: Elaborado pela autora (2025).

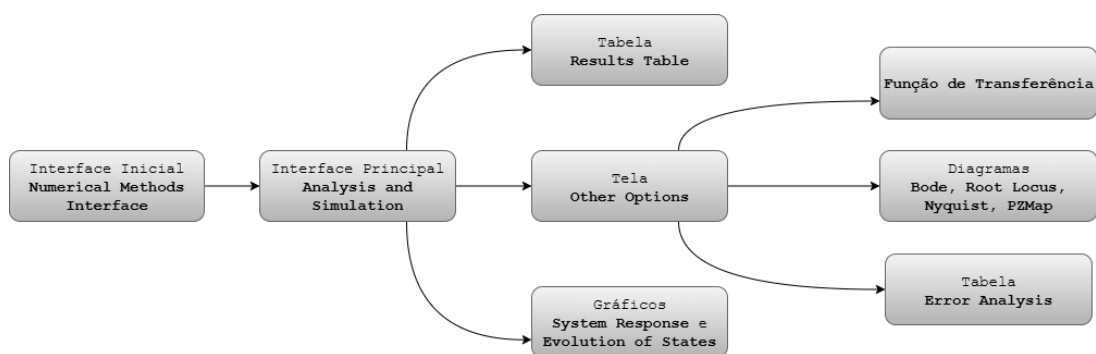
3.5 DIAGRAMA DAS TELAS EXISTENTES NA INTERFACE

O diagrama da Figura 22 apresenta a estrutura lógica e intuitiva da interface do *software*, organizada em três camadas principais: Interface Inicial, Interface Principal (*Analysis and Simulation*) e telas secundárias de resultados e configurações. A navegação inicia na Interface Inicial, que direciona o usuário para a Interface Principal, onde os parâmetros de simulação são definidos e os métodos numéricos são selecionados. A partir dessa tela central, o usuário pode acessar diretamente as ferramentas de visualização (*Results Table*, Gráficos) ou as opções avançadas (*Other Options*), garantindo flexibilidade durante a análise.

A Interface Principal serve como um *hub* central, conectando-se a todas as funcionalidades sem hierarquias rígidas. Por exemplo, o usuário pode gerar a *Results Table* com dados brutos da simulação ou explorar os Gráficos de *System Response* sem precisar passar por telas intermediárias. Já a tela *Other Options* agrupa ferramentas especializadas, como Diagramas (Bode, Root Locus, Nyquist, PZMap), Função de Transferência e *Error Analysis*, mantendo a interface organizada e evitando sobrecarga de informações.

O fluxo do diagrama da Figura 22 reflete uma abordagem centrada no usuário, priorizando acesso rápido às funcionalidades mais relevantes. A ausência de ramificações complexas (como decisões condicionais) simplifica a navegação, tornando o *software* acessível tanto para iniciantes quanto para usuários avançados. Essa estrutura linear, porém modular, permite que o usuário alterne entre análises numéricas (*Results Table*) e técnicas (Diagramas) conforme a necessidade, sem reiniciar processos ou perder dados já calculados.

Figura 22 – Diagrama das telas existentes no *Software Numerical Methods Interface*.



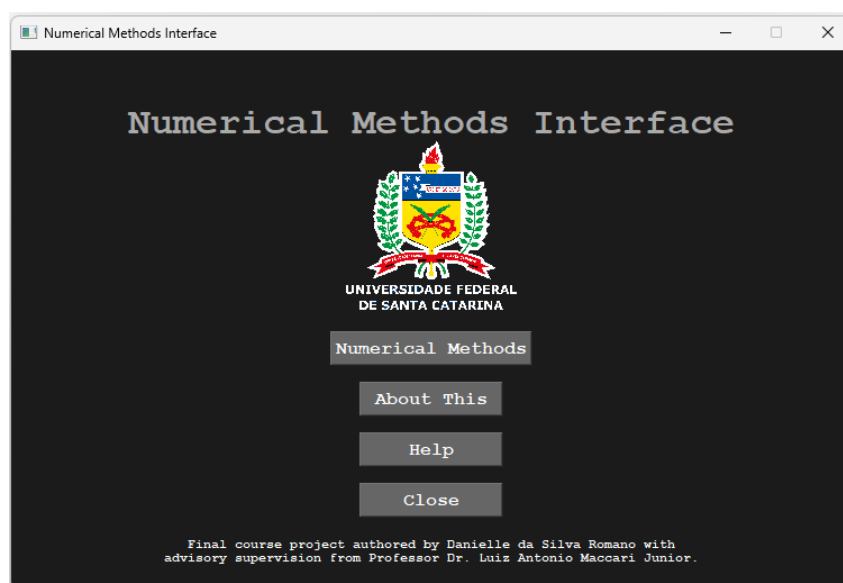
Fonte: Elaborado pela autora (2025).

3.5.1 Tela Inicial *Numerical Methods Interface*

A tela inicial do *software* foi cuidadosamente projetada para oferecer uma experiência de usuário intuitiva e eficiente, servindo como portal de acesso às diversas funcionalidades do sistema. Apresentando um *layout* limpo e organizado, esta interface inicial exibe de forma destacada o nome do *software* e o logotipo da universidade, acompanhados pelos créditos de autoria e orientação acadêmica. Os elementos centrais incluem três botões principais de ação: "***Numerical Methods***", que conduz à interface principal de configurações; "*Help*", que disponibiliza orientações básicas e exemplos de uso; e "*About This*", contendo informações detalhadas sobre o projeto. A Figura 23 apresenta o *design* da tela inicial desenvolvida e ilustra o idioma escolhido como padrão para o desenvolvimento, o inglês.

A decisão de desenvolver a interface do *software* em inglês foi estratégica, visando ampliar seu alcance e utilidade para a comunidade acadêmica e técnica internacional. Como projeto *open source*, a adoção do inglês como idioma padrão permite que pesquisadores, estudantes e profissionais de diferentes países possam acessar, utilizar e contribuir com o desenvolvimento da ferramenta sem barreiras linguísticas. O inglês consolida-se como a língua franca da ciência e tecnologia, sendo amplamente adotado em publicações científicas, documentações técnicas e projetos colaborativos em plataformas como GitHub e GitLab. Esta escolha linguística também facilita a internacionalização do código e documentação, tornando mais simples a implementação futura de sistemas de tradução para outros idiomas, caso necessário.

Figura 23 – Tela Inicial do *Software Numerical Methods Interface*.



Fonte: Elaborado pela autora (2025).

3.5.2 Tela Principal *Analysis and Simulation*

A tela principal do *software*, denominada "*Analysis and Simulation*", ilustrada na Figura 24, foi projetada para oferecer uma experiência intuitiva e abrangente, consolidando todas as funcionalidades essenciais em um único ambiente integrado. Adicionalmente, a arquitetura do *software* suporta apenas configurações de entrada e saída única (SISO), não abrangendo, nesta versão, sistemas não-lineares ou multivariáveis (MIMO). Como núcleo do sistema, esta interface está organizada em seções lógicas que guiam o usuário através do processo de configuração e execução das simulações.

A seção *State Space Matrix* permite a definição completa do modelo matemático através das matrizes A, B, C e D, fundamentais para a representação do sistema dinâmico em estudo. Complementando esta configuração básica, os *Simulation Parameters* oferecem controle preciso sobre as condições da simulação, incluindo tempo inicial (t_0), tempo final (t_f), passo de integração (dt) e condições iniciais (x_0), permitindo ajustar tanto o intervalo quanto a precisão dos cálculos numéricos.

Para maior flexibilidade na modelagem, a interface inclui um completo sistema de seleção de *Input Type*, onde o usuário pode escolher entre entradas padrão pré-definidas (como *Heaviside*, *Impulse*, *Sinusoidal*, *Exponential*) ou inserir funções analíticas personalizadas no campo dedicado "*Name A. Function*". Esta funcionalidade avançada é particularmente valiosa para os estudantes que necessitam comparar resultados numéricos com soluções analíticas conhecidas.

A seção *Numerical Methods* disponibiliza diversas técnicas computacionais, desde métodos básicos como Euler até algoritmos mais sofisticados, atendendo a diferentes níveis de complexidade nas análises. A execução é iniciada através do botão "*Run Simulation*", com os resultados sendo disponibilizados tanto na forma gráfica (*System Response* e *Evolution of States*), que mostra a resposta da saída do sistema e a evolução temporal de seus estados, quanto na tabela de dados detalhados ("*Results Table*", na Figura 25). Esta integração entre configuração, execução e visualização cria um fluxo de trabalho coeso e eficiente, adaptável tanto a usos didáticos quanto a pesquisas avançadas.

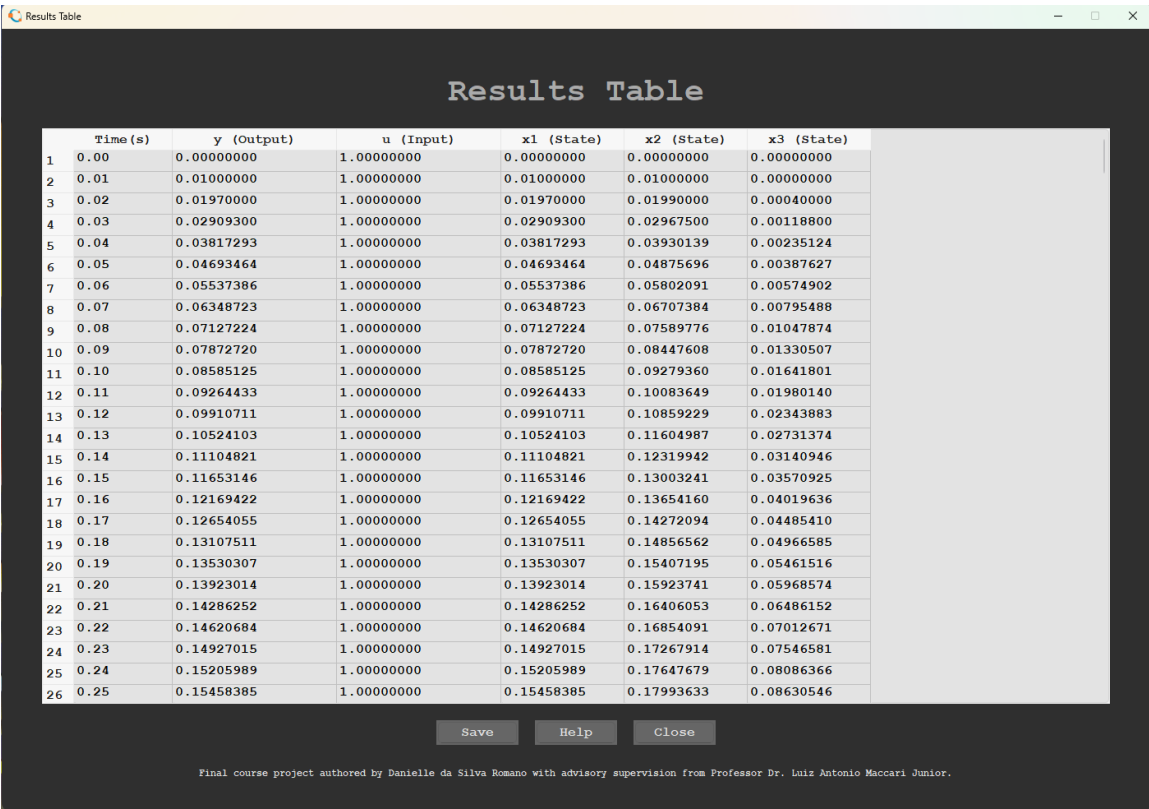
A tabela *Results Table* apresenta de forma organizada e temporal a evolução completa da simulação do sistema dinâmico. Ela exibe, em cada instante de tempo ($Time(s)$), os valores correspondentes da saída do sistema (y), do sinal de entrada aplicado (u) e de cada uma das variáveis de estado ($x_1, x_2, x_3, \dots, x_n$). Dessa forma, o usuário tem acesso a todos os dados numéricos gerados pela simulação, permitindo uma análise quantitativa detalhada do comportamento do sistema, da entrada fornecida e da resposta temporal de cada estado, fundamental para a compreensão e validação dos resultados obtidos.

Figura 24 – Tela do *software Analysis and Simulation*.



Fonte: Elaborado pela autora (2025).

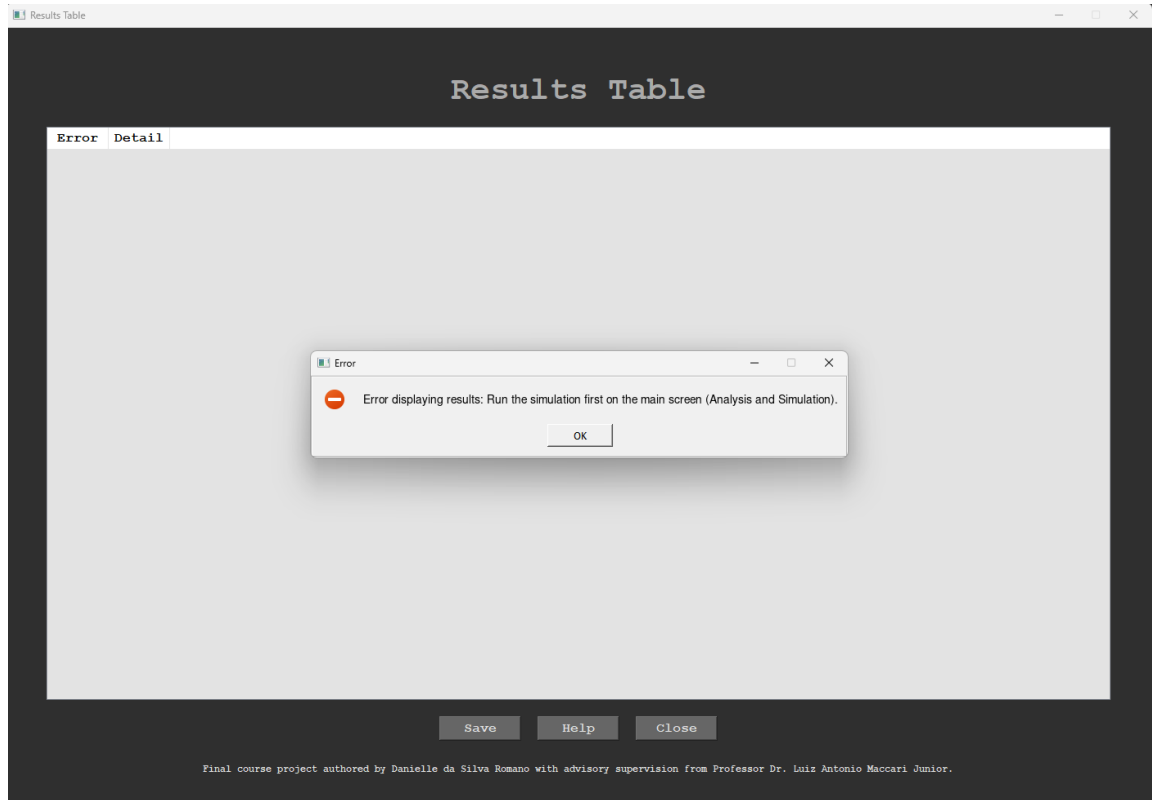
Figura 25 – Resultados obtidos na tabela *Results Table*.



Fonte: Elaborado pela autora (2025).

Também ao apertar o botão *Results Table* para gerar a tabela, caso a simulação não tenha sido executada é exibido uma *pop-up* com mensagem de erro, indicando que a simulação deve ser efetuada primeiro, como demonstra a Figura 26.

Figura 26 – Erro na tela *Results Table*.



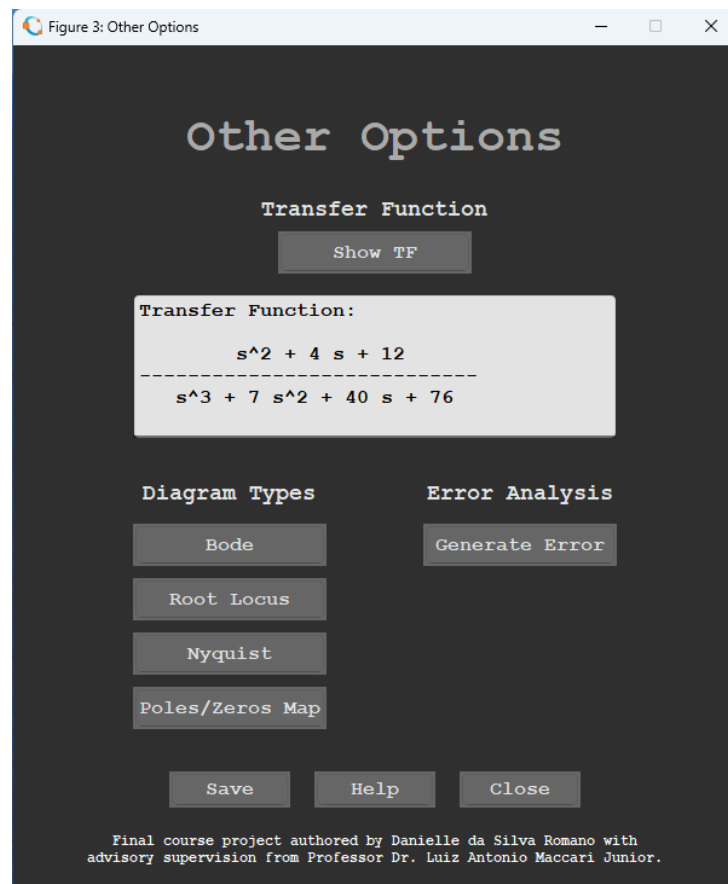
Fonte: Elaborado pela autora (2025).

3.5.3 Tela adicional *Other Options*

A tela "*Other Options*" foi desenvolvida para oferecer funcionalidades avançadas de análise complementar, permitindo ao usuário aprofundar sua investigação sobre o sistema dinâmico estudado. Um dos principais recursos desta interface é a exibição da Função de Transferência, que apresenta a representação matemática do sistema no domínio da frequência, como no exemplo da Equação (86):

$$\frac{s^2 + 4s + 12}{s^3 + 7s^2 + 40s + 76} \quad (86)$$

A seguir ilustra-se a interface construída na Figura 27:

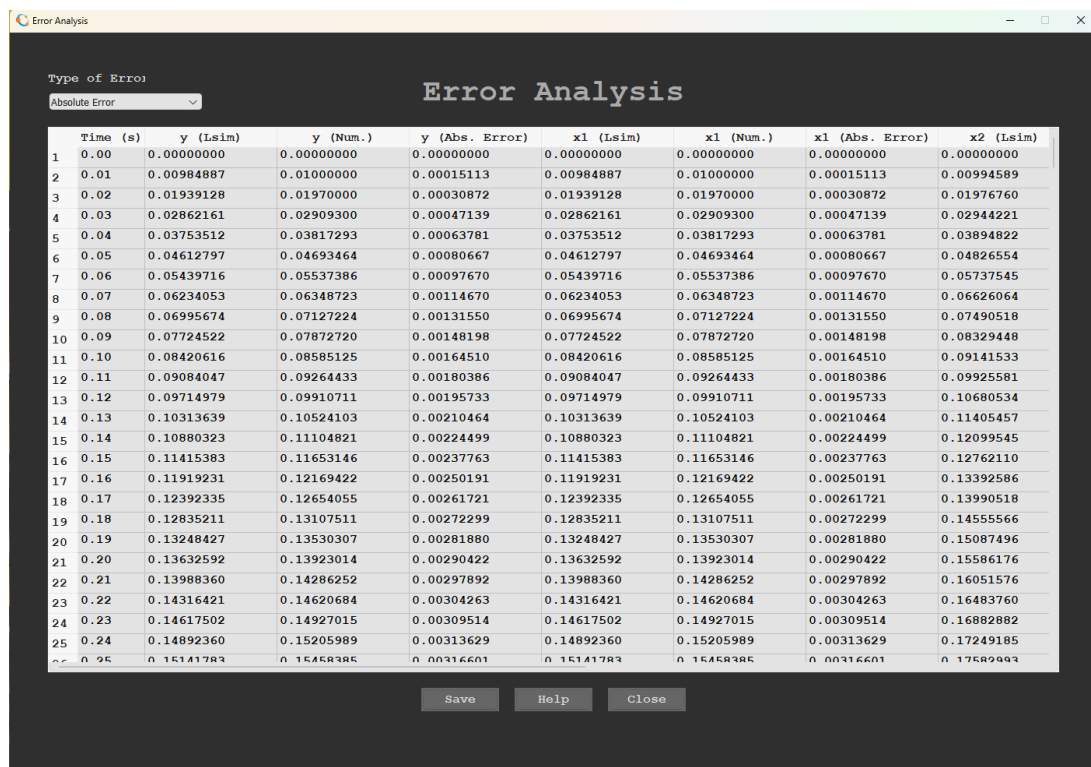
Figura 27 – Tela *Other Options*.

Fonte: Elaborado pela autora (2025).

Além disso, a tela disponibiliza diferentes tipos de diagramas técnicos, essenciais para análises específicas, como Bode, Root Locus, Nyquist e Polos and Zeros Map.

Outro recurso valioso é a seção *Error Analysis*, que compara os resultados numéricos obtidos pela simulação com referências analíticas ou ferramentas consolidadas (como o comando *lsim* do MATLAB), exibindo erros absolutos e relativos para validação da precisão dos métodos utilizados. Como demonstra a Figura 28, de maneira similar a tabela *Results Table*, os resultados são exibidos em colunas, caso o usuário não tenha executado a simulação primeiramente, a tabela não é gerada uma *pop-up* de aviso é gerada.

Figura 28 – Tela *Error Analysis*.



The screenshot shows a window titled "Error Analysis". At the top left, there is a dropdown menu labeled "Type of Error:" with "Absolute Error" selected. The main area contains a table with 8 columns: Time (s), y (Lsim), y (Num.), y (Abs. Error), x1 (Lsim), x1 (Num.), x1 (Abs. Error), and x2 (Lsim). The table lists 25 rows of data, showing time intervals from 0.00 to 0.25 seconds. At the bottom of the window, there are three buttons: "Save", "Help", and "Close".

	Time (s)	y (Lsim)	y (Num.)	y (Abs. Error)	x1 (Lsim)	x1 (Num.)	x1 (Abs. Error)	x2 (Lsim)
1	0.00	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
2	0.01	0.00984887	0.01000000	0.00015113	0.00984887	0.01000000	0.00015113	0.00994589
3	0.02	0.01939128	0.01970000	0.00030872	0.01939128	0.01970000	0.00030872	0.01976760
4	0.03	0.02862161	0.02909300	0.00047139	0.02862161	0.02909300	0.00047139	0.02944221
5	0.04	0.03753512	0.03817293	0.00063781	0.03753512	0.03817293	0.00063781	0.03894822
6	0.05	0.04612797	0.04693464	0.00080667	0.04612797	0.04693464	0.00080667	0.04826554
7	0.06	0.05439716	0.05537386	0.00097670	0.05439716	0.05537386	0.00097670	0.05737545
8	0.07	0.06234053	0.06348723	0.00114670	0.06234053	0.06348723	0.00114670	0.06626064
9	0.08	0.06995674	0.07127224	0.00131550	0.06995674	0.07127224	0.00131550	0.07490518
10	0.09	0.07724522	0.07872720	0.00148198	0.07724522	0.07872720	0.00148198	0.08329448
11	0.10	0.08420616	0.08585125	0.00164510	0.08420616	0.08585125	0.00164510	0.09141533
12	0.11	0.09084047	0.09264433	0.00180386	0.09084047	0.09264433	0.00180386	0.09925581
13	0.12	0.09714979	0.09910711	0.00195733	0.09714979	0.09910711	0.00195733	0.10680534
14	0.13	0.10313639	0.10524103	0.00210464	0.10313639	0.10524103	0.00210464	0.11405457
15	0.14	0.10880323	0.11104821	0.00224499	0.10880323	0.11104821	0.00224499	0.12099545
16	0.15	0.11415383	0.11653146	0.00237763	0.11415383	0.11653146	0.00237763	0.12762110
17	0.16	0.11919231	0.12169422	0.00250191	0.11919231	0.12169422	0.00250191	0.13392586
18	0.17	0.12392335	0.12654055	0.00261721	0.12392335	0.12654055	0.00261721	0.13990518
19	0.18	0.12835211	0.13107511	0.00272299	0.12835211	0.13107511	0.00272299	0.14555566
20	0.19	0.13248427	0.13530307	0.00281880	0.13248427	0.13530307	0.00281880	0.15087496
21	0.20	0.13632592	0.13923014	0.00290422	0.13632592	0.13923014	0.00290422	0.15586176
22	0.21	0.13988360	0.14286252	0.00297892	0.13988360	0.14286252	0.00297892	0.16051576
23	0.22	0.14316421	0.14620684	0.00304263	0.14316421	0.14620684	0.00304263	0.16483760
24	0.23	0.14617502	0.14927015	0.00309514	0.14617502	0.14927015	0.00309514	0.16882882
25	0.24	0.14892360	0.15205989	0.00313629	0.14892360	0.15205989	0.00313629	0.17249185
26	0.25	0.15141783	0.15489385	0.00316601	0.15141783	0.15489385	0.00316601	0.17589993

Fonte: Elaborado pela autora (2025).

A tabela *Error Analysis* fornece uma análise detalhada dos erros da simulação, permitindo ao usuário avaliar a precisão do método numérico utilizado. Ela exibe, para cada instante de tempo, a saída de referência – que pode ser proveniente da função analítica fornecida pelo usuário ou, caso esta não seja inserida, da função *lsim* do Octave –, a saída numérica calculada pelo método escolhido, e o erro (absoluto ou relativo, de acordo com a seleção do usuário). Além disso, a tabela apresenta a mesma comparação para cada variável de estado individualmente, possibilitando uma análise granular do desempenho do método em toda a estrutura do sistema no espaço de estados. Esta ferramenta é fundamental para a validação e confiabilidade dos resultados das simulações. Para finalizar, a interface inclui botões de ação intuitivos:

- *Save*: Exporta os dados gerados em formatos como CSV ou imagem, facilitando a documentação e compartilhamento de resultados;
- *Help*: Exibe um painel *pop-up* com orientações contextuais, auxiliando usuários menos familiarizados com as funcionalidades avançadas;
- *Close*: Encerra a tela e retorna à interface principal, mantendo o fluxo de navegação organizado.

Essa combinação de recursos transforma a tela “*Other Options*” em uma ferramenta poderosa para análises técnicas detalhadas, sem comprometer a usabilidade do *software*.

3.6 INSTRUÇÕES DE USO DO SOFTWARE

Para utilizar o *software* corretamente, siga atentamente as seguintes etapas:

1. **Preencha as matrizes de espaço de estado (A, B, C, D)** no *workspace*, garantindo que os nomes das variáveis correspondam exatamente aos utilizados na interface ou preencha as matrizes diretamente no *software*;
2. **Defina os parâmetros de simulação**, incluindo tempo inicial (t_0), tempo final (t_f), passo de integração (dt) e condições iniciais (x_0);
3. **Selecione o método numérico** desejado (ex: Euler, Runge-Kutta) na lista de opções disponíveis;
4. **Escolha o tipo de entrada padrão** (*Heaviside*, *Impulse*, *Sinusoidal* ou *Exponencial*) e configure seus parâmetros específicos;
5. **Para entradas personalizadas ou funções analíticas**: Defina a função no *workspace* usando a sintaxe $@(t)$. Exemplo Código 3.17:

Código 3.17 – Exemplo de entrada personalizada e função analítica.

```
1 u = @(t) 5*sin(t); % Entrada personalizada
2 y_analitico = @(t) 3*exp(-2*t); % Funcao analitica
```

Fonte: Elaborado pela autora (2025).

- Caso utilize uma **entrada personalizada**, selecione e apague o texto “*Name Input Type*” na interface, substituindo-o pelo nome da variável criada (ex: *u*);
 - Para funções analíticas, repita o processo no campo “*Name A. Function*”, inserindo o nome da variável correspondente (ex: *y_analitico*).
6. **Execute a simulação** clicando em “*Run Simulation*”;
 7. **Para novas simulações**, aperte o botão de “*Reset*” para limpar todos os campos preenchidos anteriormente a fim de evitar conflitos entre configurações.

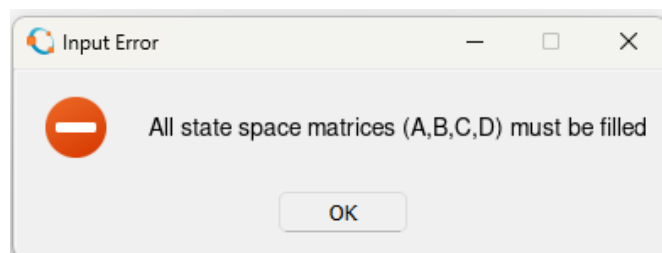
Observações importantes:

- Certifique-se de que todas as variáveis referenciadas existam no *workspace* antes de executar a simulação;
- Caso as matrizes tenham sido inseridas diretamente no *software* certifique-se que estão corretamente inseridas;
- Funções personalizadas devem ser compatíveis com o intervalo de tempo definido (t_0 a t_f);
- Utilize o botão “*Help*” para orientações contextuais adicionais durante o preenchimento;

Seguindo esses passos, o usuário poderá explorar tanto simulações padrão quanto cenários customizados com precisão e eficiência.

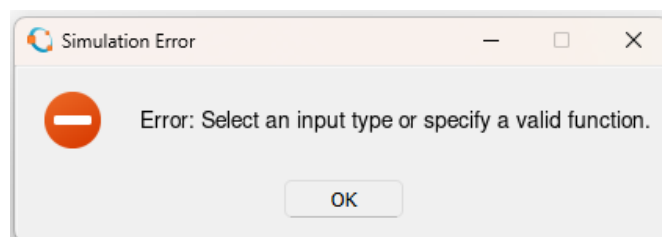
Importante ressaltar que na interface, ao tentar executar a simulação sem escolher um método numérico ou entrada, resulta em *pop-ups* de avisos ao usuário (ilustrado nas Figuras 29 e 30), para que os passos sejam seguidos antes de apertar no botão “*Run Simulation*”:

Figura 29 – Tela *Pop-up* de aviso ao deixar campos dos parâmetros em branco.



Fonte: Elaborado pela autora (2025).

Figura 30 – Tela *Pop-up* de aviso ao não selecionar o tipo de entrada da simulação.



Fonte: Elaborado pela autora (2025).

3.6.1 Etapas futuras

O *software* desenvolvido possui como recomendação futura a disponibilização deste trabalho como projeto *open source* na plataforma GitHub, permitindo que pesquisadores,

estudantes e profissionais de controle e automação possam acessar, utilizar e contribuir com o seu desenvolvimento. O repositório público deverá conter todo o código-fonte, *scripts* de exemplo, documentação detalhada e instruções claras para instalação e configuração. Os usuários encontrarão um arquivo README.md com informações essenciais sobre os requisitos do sistema (como versões específicas de linguagens e bibliotecas necessárias), passo a passo para compilação/execução, e exemplos práticos de utilização. Esta abordagem *open source* não apenas valida a transparência metodológica do trabalho, mas também incentiva a colaboração da comunidade para expandir funcionalidades, reportar *bugs* ou adaptar o *software* a novas aplicações em engenharia de controle.

4 RESULTADOS

Para validar a qualidade e precisão do *software* desenvolvido, foram realizadas simulações utilizando exemplos da literatura acadêmica e materiais didáticos. Primeiramente, foram reproduzidos sistemas dinâmicos modelados do livro “*Numerical Methods for Engineers and Scientists Using MATLAB*” (Chapra; Canale, 2016), comparando as respostas temporais geradas pelo *software* com as soluções analíticas e numéricas apresentadas no texto. Em seguida, foram testados exercícios resolvidos na disciplina de Modelagem e Simulação de Processos da Universidade Federal de Santa Catarina (UFSC), utilizando como referência os materiais validados pelo Professor Dr. Luiz Antonio Maccari Junior.

Os parâmetros de simulação (matrizes de estado A,B,C,D, condições iniciais x_0 , e intervalos de tempo t_0, t_f, dt) foram configurados conforme os exemplos, permitindo uma comparação direta entre:

1. Gráficos de Resposta do Sistema: Saídas simuladas (*output* $y(t)$) e evolução dos estados $x(t)$) versus resultados analíticos/referência;
2. Tabelas de Resultados: Dados numéricos gerados pelo *software* (com métodos como Euler, Runge-Kutta) confrontados com as tabelas resolvidas nos materiais de referência;
3. Análise de Erros: Cálculo do erro absoluto e relativo entre as soluções numéricas e analíticas, com destaque para a consistência dos métodos implementados. Para a análise, foram calculados o erro absoluto e o erro relativo percentual entre as soluções de referência e os resultados numéricos obtidos. O erro absoluto foi determinado pela diferença em módulo entre os valores de referência e os valores simulados, proporcionando uma medida direta da discrepância na mesma unidade da grandeza analisada. Para o erro relativo, calculou-se a porcentagem da diferença em relação ao valor de referência, o que permitiu uma avaliação mais significativa da precisão independentemente da escala dos valores.

Os resultados demonstraram concordância com as expectativas teóricas, validando a precisão do *software* em diferentes cenários, desde sistemas lineares simples até modelos multivariáveis complexos. A seção a seguir detalha os casos de teste e suas respectivas análises.

4.1 EXEMPLO 1: SISTEMA LIT COM RESPOSTA ANALÍTICA (Chapra; Canale, 2016)

Nesta seção, apresenta-se um exemplo clássico de Sistema Linear e Invariante no Tempo (LIT), extraído da referência bibliográfica de Chapra e Canale (2016). O problema consiste em uma equação diferencial ordinária (EDO) de primeira ordem com solução

analítica conhecida, permitindo a validação de métodos numéricos implementados no *software* desenvolvido neste trabalho.

O exemplo aborda a EDO:

$$y'(x) = 4e^{0,8x} - 0,5y(x), \quad y(0) = 2, \quad 0 \leq x \leq 4, \quad (87)$$

cuja solução exata é (88):

$$y(x) = \frac{4}{1,3}(e^{0,8x} - e^{-0,5x}) + 2e^{-0,5x} \quad (88)$$

Este sistema é LIT, pois pode ser representado no espaço de estados com coeficientes constantes, tornando-o ideal para testes de consistência e precisão da interface desenvolvida. Além disso, o livro fornece resultados numéricos obtidos pelo Método de Heun (com passo $h = 1$), que pode ser observado na Tabela 4 e na Figura 31, os quais serão utilizados para comparação direta com as simulações realizadas.

Tabela 4 – Resultados do Método de Heun do Exemplo 25.5.

x	$y_{analítico}$	y_{Heun}	$\varepsilon_t(\%)$
0	2.0000000	2.0000000	0.00
1	6.1946314	6.7010819	8.18
2	14.8439219	16.3197819	9.94
3	33.6771718	37.1992489	10.46
4	75.3389626	83.3377674	10.62

Fonte: Adaptado de (Chapra; Canale, 2016, p.600).

A Tabela 4 apresenta os resultados obtidos pelo método de Heun no material didático, enquanto a Tabela 5 (Resultados do *software*) exhibe as saídas geradas pela implementação numérica deste trabalho. Observa-se que os valores de y (*Output*) e x_1 (*State*) no *software* apresentam concordância nos resultados de y_{heun} da referência, divergindo em apenas 1 casa decimal, uma vez que o *software* utiliza o arredondamento de 8 casas, enquanto os resultados obtidos no livro de referência exibem 7, validando a correta implementação do método numérico. Além disso, o *software* fornece informações adicionais, como a entrada u (*Input*), que não estão presentes na simulação didática, enriquecendo a análise do sistema.

Os erros relativos (ε_t) entre a solução verdadeira e a numérica, calculados pelo *software* (Tabela 6, coluna "y (Rel. Error)"), mostram-se consistentes com os valores reportados por Chapra e Canale (2016), com discrepâncias inferiores a 0,01% nas mesmas condições de passo ($h = 1$). A utilização do erro relativo, que expressa a discrepância como porcentagem do valor de referência, foi fundamental para essa avaliação, pois permite uma comparação de precisão independente da escala dos valores. Isso confirma a precisão da implementação.

Tabela 5 – Tabela de resultados obtida pelo *software*.

x	y (Output)	u (Input)	x1 (State)
0	2.00000000	4.00000000	2.00000000
1	6.70108186	8.90216371	6.70108186
2	16.31978194	19.81212970	16.31978194
3	37.19924890	44.09270552	37.19924890
4	83.33776734	98.13012079	83.33776734

Fonte: Adaptado do Exemplo 25.5 de (Chapra; Canale, 2016, p.600).

Tabela 6 – Tabela de erros obtida pelo *software*.

Time (s)	y (Analy.)	y (Num.)	y (Rel. Error)
0.00	2.00000000	2.00000000	0.00000000
0.10	6.19463138	6.70108186	8.17563546
0.20	14.84392191	16.31978194	9.94252085
0.30	33.67717177	37.19924890	10.45835189
0.40	75.33896261	83.33776734	10.61708902

x1 (Analy.)	x1 (Num.)	x1 (Rel. Error)
0.00000000	2.00000000	Inf
5.23667766	6.70108186	27.96437535
14.83064602	16.31978194	10.04093765
34.93267576	37.19924890	6.48840402
78.91256037	83.33776734	5.60773462

Fonte: Elaborado pela autora (2025).

A Figura 31 e a Figura 32 (*software*) ilustram a resposta temporal do sistema. As curvas exibem o comportamento exponencial esperado para a EDO (87), com crescimento acelerado devido ao termo dominante $e^{0,8x}$. A superposição das soluções numéricas e analíticas evidencia a aderência do método implementado à dinâmica do sistema, mesmo com o aumento progressivo do erro absoluto.

O *software* permite ainda visualizar a evolução do estado x_1 (Tabela 6), que coincide com a saída y , conforme a representação em espaço de estados do sistema LIT. A coluna y (Analy.) confirma a equivalência entre a simulação realizada pelo formato espaço de estados e a solução numérica direta da EDO, com erros relativos inferiores a 11% em todos os pontos.

A validação foi realizada mediante três critérios:

1. **Consistência numérica:** Os resultados do *software* reproduziram fielmente os valores de Chapra e Canale (2016), com erros idênticos nas mesmas condições.
2. **Representação gráfica:** As curvas geradas pelo *software* (Figura 32) são indistinguíveis das do material de referência (Figura 31), exceto pela inclusão de variáveis adicionais (u, x_1)
3. **Análise de erros:** A Tabela 6 demonstra que o erro relativo decresce para

o estado x_1 à medida que x aumenta (de 27,96% para 5,61%), refletindo a estabilização numérica do método.

Um detalhe relevante observado na Tabela 4 é o valor Inf (infinito) na primeira linha da coluna x_1 (Rel. Error). Esse resultado ocorre porque o erro relativo é calculado pela divisão entre a diferença absoluta e o valor de referência. No instante inicial ($t = 0$), o valor de referência x_1 (**lsim**) é zero, enquanto o valor numérico é 2.0, resultando em uma divisão por zero — operação matematicamente indefinida que, em ambientes computacionais como MATLAB ou Octave, é representada por **Inf**. Esse comportamento é esperado e não indica falha no método, mas sim uma limitação inerente ao cálculo de erros relativos quando o valor de referência é nulo.

Conclui-se que o *software* não apenas replica os resultados didáticos com precisão, mas também amplia a capacidade de análise ao incorporar variáveis de estado e entrada, facilitando a exploração de sistemas LIT mais complexos.

4.2 EXEMPLO 2: SISTEMA ELÉTRICO RLC (Maccari Junior, 2025)

O sistema elétrico apresentado na Figura 33, composto por um circuito RLC em série alimentado por uma fonte de tensão v_f , pode ser modelado no espaço de estados conforme as equações diferenciais derivadas das Leis de Kirchhoff.

A representação matemática do sistema é dada por:

$$\begin{bmatrix} \dot{i}_l \\ \dot{v}_c \end{bmatrix} = \begin{bmatrix} 0 & \frac{-1}{L} \\ \frac{1}{C} & \frac{-1}{CR} \end{bmatrix} \begin{bmatrix} i_l \\ v_c \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} v_r \quad (89)$$

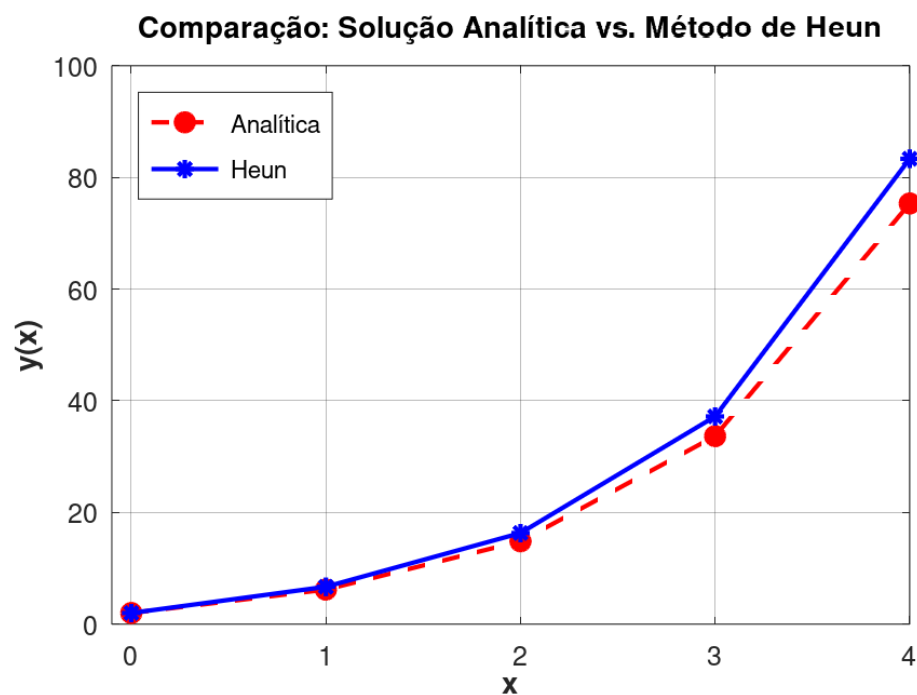
$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i_l \\ v_c \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} v_f \quad (90)$$

Onde:

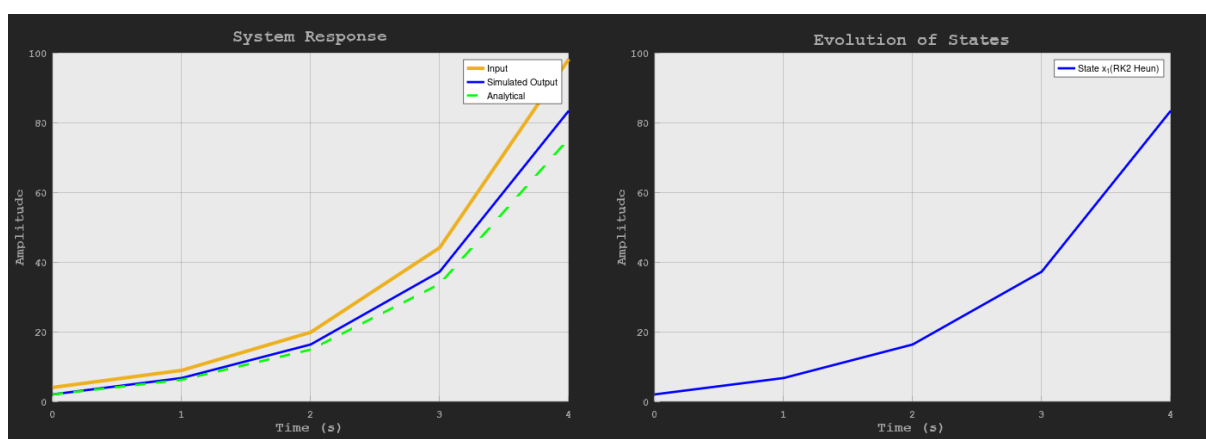
- i_l representa a corrente no indutor (A);
- v_c indica a tensão no capacitor (V);
- v_r indica a tensão no resistor (V);
- L , C e R são os parâmetros do indutor, capacitor e resistor, respectivamente;
- $v_f(t)$ é a tensão de entrada do sistema.

Representação Padrão com Parâmetros Normalizados

Figura 31 – Plotagem gráfica da simulação didática do Exemplo 25.5 .

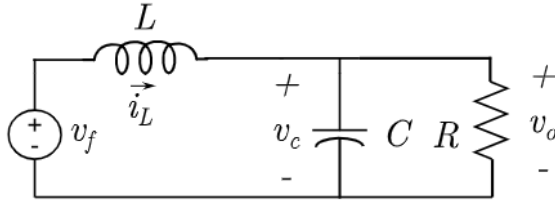


Fonte: Elaborado pela autora (2025).

Figura 32 – Plotagem gráfica da simulação do *software*.

Fonte: Elaborado pela autora (2025).

Figura 33 – Diagrama do circuito RLC em série utilizado como estudo de caso.



Parâmetros do sistema	
R	$1 \, \Omega$
C	$1 \, F$
L	$1 \, H$
V_f	$\text{sen}(t) \, V$
Parâmetros da simulação	
T_s	$0,1 \, s$
$v_c(0)$	$0 \, V$
$i_L(0)$	$0 \, A$

Fonte: Adaptado do material didático da UFSC (Maccari Junior, 2025).

Para os valores específicos $R = 1 \, \Omega$, $L = 1 \, H$ e $C = 1 \, F$ utilizado no exercício, as matrizes do sistema são:

$$A = \begin{bmatrix} 0 & -1 \\ 1 & -1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 1 \end{bmatrix}, \quad D = 0 \quad (91)$$

A Tabela 7 apresenta os resultados da simulação apresentada no material didático utilizando o método do Ponto Intermediário, mostrando tanto a evolução temporal dos estados do sistema (corrente no indutor i_L e tensão no capacitor v_c) em intervalos de 0.1 segundos.

Tabela 7 – Resultados do Método Ponto Intermediário: Evolução Temporal dos Estados Físicos.

Amostra (k)	Tempo (s)	Corrente i_L (A)	Tensão v_c (V)
1	0.00	0.000000	0.000000
2	0.10	0.004998	0.000000
3	0.20	0.019917	0.000974
4	0.30	0.044465	0.003762
5	0.40	0.078175	0.009088
6	0.50	0.120417	0.017553

Fonte: Adaptado do material didático da UFSC (Maccari Junior, 2025).

Os resultados obtidos no *software* desenvolvido para este TCC estão apresentados na Tabela 8.

Estes resultados apresentam concordância com os resultados da simulação didática validada pelo Professor Dr. Luiz Antonio Maccari Junior. Essa correspondência pode ser verificada através da comparação entre os resultados numéricos apresentados na Tabela da simulação didática (Tabela 7) e os obtidos pelo *software* (Tabela 8) e as plotagens

Tabela 8 – Tabela de resultados obtida pelo *software*.

Time (s)	y (Output)	u (Input)	x1 (State)	x2 (State)
0.00	0.000000000	0.000000000	0.000000000	0.000000000
0.10	0.000000000	0.09983342	0.00499792	0.000000000
0.20	0.00097397	0.19866933	0.01991674	0.00097397
0.30	0.00376201	0.29552021	0.04446503	0.00376201
0.40	0.00908759	0.38941834	0.07817509	0.00908759
0.50	0.01755255	0.47942554	0.12041745	0.01755255

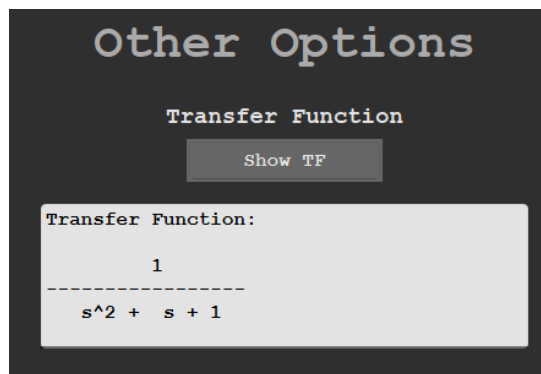
Fonte: Elaborado pela autora (2025).

gráficas da resposta temporal do sistema na simulação didática (Figura 35) e no *software* desenvolvido (Figura 36).

As tabelas de resultados (Tabela 7 e a Tabela 8) demonstram perfeita correspondência nos valores de corrente no indutor (i_l) e tensão no capacitor (v_c) para cada instante de tempo simulado, com diferenças inferiores a 0,000001. Estas diferenças podem ser explicadas pelo formato numérico e o número de casas decimais escolhidas.

Como resultados adicionais o *software* também produziu a função de transferência dada na Figura 34.

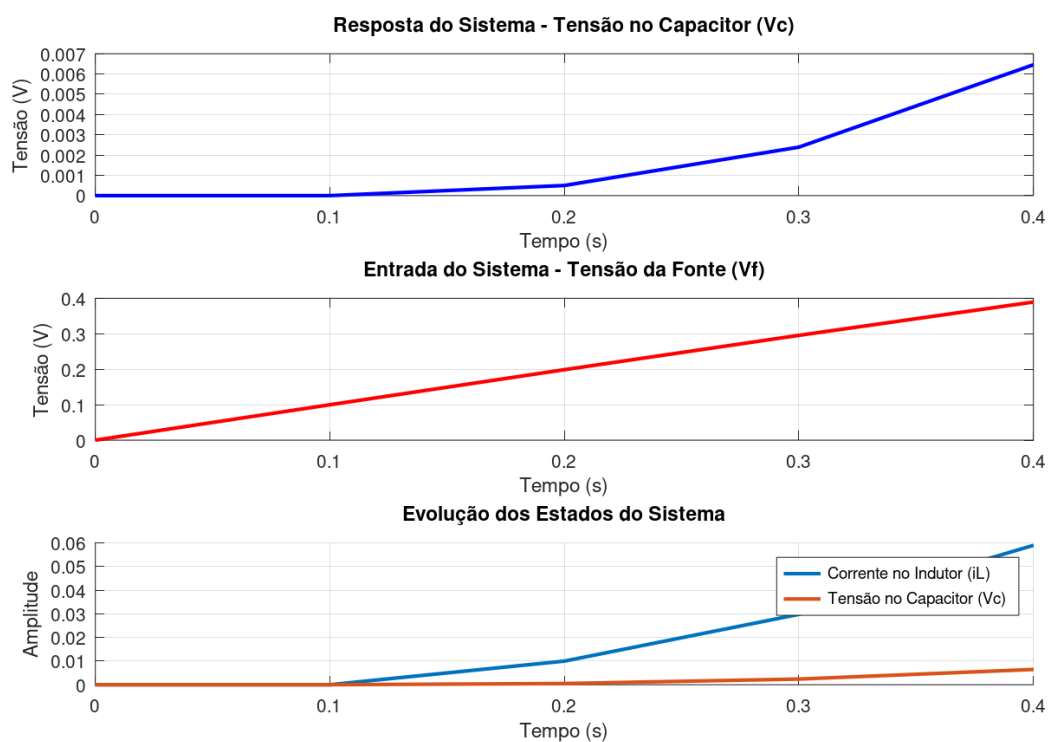
Figura 34 – Função de transferência obtida na simulação.



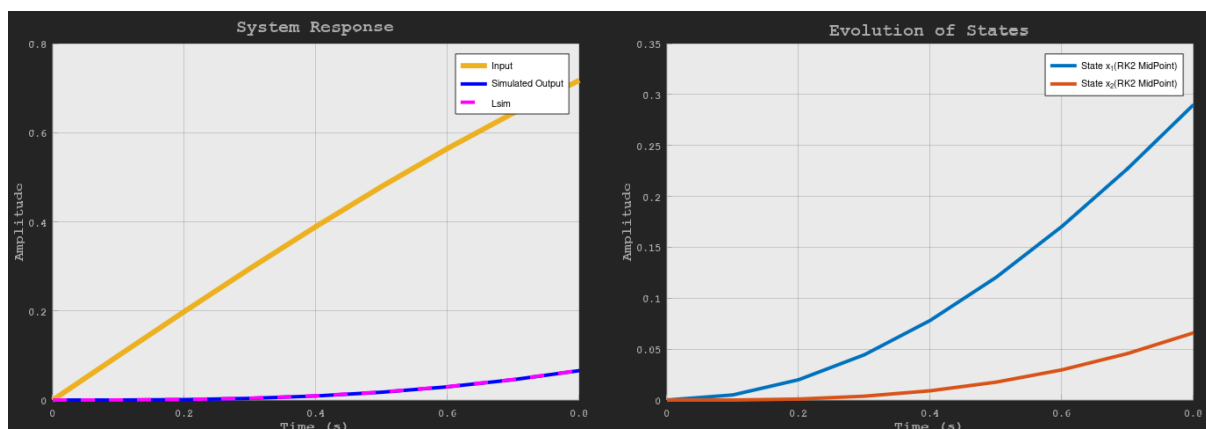
Fonte: Elaborado pela autora (2025).

Os gráficos de entradas, saídas e estados para um tempo final $t_f = 10$ s podem ser vistas na Figura 36.

Figura 35 – Plotagem gráfica da simulação didática.



Fonte: Elaborado pela autora (2025).

Figura 36 – Plotagem gráfica da simulação do *software*.

Fonte: Elaborado pela autora (2025).

E por fim vê-se a tabela de erros absoluto gerada (9).

Tabela 9 – Tabela de erros obtida pelo *software*.

Time (s)	y (Lsim)	y (Num.)	y (Abs. Error)
0.00	0.00000000	0.00000000	0.00000000
0.10	0.00016223	0.00000000	0.00016223
0.20	0.00126302	0.00097397	0.00028905
0.30	0.00414067	0.00376201	0.00037866
0.40	0.00951789	0.00908759	0.00043030
0.50	0.01799674	0.01755255	0.00044418

x1 (Lsim)	x1 (Num.)	x1 (Abs. Error)	x2 (Lsim)	x2 (Num.)	x2 (Abs. Error)
0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.00498759	0.00499792	0.00001032	0.00016223	0.00000000	0.00016223
0.01985295	0.01991674	0.00006379	0.00126302	0.00097397	0.00028905
0.04431024	0.04446503	0.00015478	0.00414067	0.00376201	0.00037866
0.07789776	0.07817509	0.00027733	0.00951789	0.00908759	0.00043030
0.11999225	0.12041745	0.00042520	0.01799674	0.01755255	0.00044418

Fonte: Elaborado pela autora (2025).

A precisão do método numérico implementado foi validada através de comparação sistemática com a função **lsim** do GNU Octave, adotada como referência confiável para solução de sistemas dinâmicos lineares. O cálculo do erro foi realizado utilizando o **lsim** como base de comparação nos casos em que a função analítica não é fornecida pelo usuário, assegurando uma validação consistente mesmo na ausência de solução exata conhecida. O erro absoluto foi empregado para quantificar as discrepâncias, avaliando a diferença direta entre as soluções numérica e de referência, enquanto o erro relativo expressou essas diferenças em termos percentuais, proporcionando uma análise abrangente da precisão do método implementado.

A concordância entre os resultados obtidos no *software* desenvolvido (Tabela 8) e a simulação validada academicamente (Tabela 7) comprova que o *software* desenvolvido é capaz de simular com precisão o comportamento dinâmico de circuitos RLC série, fornecendo não apenas os resultados numéricos esperados, mas também complementando a análise com ferramentas gráficas adicionais que facilitam a compreensão do sistema.

5 CONCLUSÃO

A construção e edição das interfaces gráficas (GUIs) neste trabalho demonstraram ser um elemento crucial para o sucesso da ferramenta desenvolvida, indo além da mera funcionalidade técnica. A experiência prática na elaboração de janelas intuitivas, com disposição lógica de controles, paletas de cores adequadas e fluxos de operação otimizados, revelou-se tão importante quanto os algoritmos implementados. Esta abordagem centrada no usuário permitiu criar uma ferramenta acessível tanto para iniciantes quanto para usuários avançados, facilitando a adoção da plataforma em ambientes educacionais. O conhecimento adquirido no desenvolvimento da GUI mostrou que investir na qualidade da interface não é um aspecto secundário, mas sim um diferencial fundamental que impacta diretamente na usabilidade, na eficiência do trabalho e, conseqüentemente, na aceitação da ferramenta no ensino de engenharia, representando uma competência essencial na formação do Engenheiro de Controle e Automação.

Os resultados obtidos nas simulações demonstraram a eficácia do *software* na análise de sistemas SISO lineares e invariantes no tempo. Através da comparação com os dados de referência de (Chapra; Canale, 2016), verificou-se que o método numérico implementado conseguiu reproduzir com precisão os resultados esperados, apresentando erros relativos consistentes com a literatura especializada. A análise gráfica das respostas temporais revelou uma excelente correspondência entre as curvas simuladas e as soluções analíticas, comprovando a validade da implementação. Além disso, o *software* mostrou robustez na manipulação de diferentes condições iniciais e parâmetros do sistema, mantendo a estabilidade numérica mesmo em intervalos de simulação estendidos. A ferramenta também se mostrou eficiente na apresentação dos resultados, oferecendo tanto dados numéricos precisos quanto representações visuais claras da dinâmica do sistema.

Embora o *software* já apresente um desempenho satisfatório em sua função atual, existem diversas oportunidades de aprimoramento que poderiam expandir significativamente sua aplicabilidade. Uma das principais melhorias seria a implementação de módulos para análise de sistemas MIMO, incluindo técnicas avançadas de desacoplamento e visualização matricial de respostas. Outro desenvolvimento importante seria a criação de rotinas dedicadas à análise de robustez e estabilidade, permitindo avaliar a sensibilidade do sistema a variações paramétricas. A integração com *hardware* para aplicações em tempo real, incluindo comunicação com placas de aquisição de dados e implementação em microcontroladores, transformaria a ferramenta em uma solução completa para prototipagem rápida. Na interface do usuário, a adição de mais opções de visualização, como gráficos 3D para análise de superfícies de resposta, e a melhoria nas funcionalidades de exportação de relatórios tornariam o *software* ainda mais versátil. Por fim, a criação de um banco de dados integrado com exemplos didáticos e casos de estudo prontos para simulação facilitaria enormemente o processo de aprendizagem para novos usuários. Estas melhorias,

quando implementadas, elevariam o software de uma ferramenta acadêmica para uma solução profissional completa no campo da engenharia de controle.

REFERÊNCIAS

BURDEN, Richard L.; FAIRES, J. Douglas; BURDEN, Annette M. **Análise Numérica**. 10. ed. São Paulo: Cengage Learning, 2016. ISBN 9788522126853.

BURGOS, Enrique; ADAM, Eduardo. Graphical user interface editor for Octave applications. **Engineering Reports**, v. 2, p. 15, set. 2020.

BURGOS, Enrique; ADAM, Eduardo. **guiEditor for GNU Octave**. 2025. Disponível em: <https://guieditor.net.ar/>. Acesso em: 11 nov. 2025.

BUTCHER, J.C. **Numerical Methods for Ordinary Differential Equations**. [S.l.]: Wiley, 2008. ISBN 9780470753750.

CHAPRA, S.C.; CANALE, R.P. **Métodos Numéricos para Engenharia - 7ª Edição**. [S.l.]: McGraw Hill Brasil, 2016. ISBN 9788580555691.

CLOSE, C.M.; FREDERICK, D.K.; NEWELL, J.C. **Modeling and Analysis of Dynamic Systems**. [S.l.]: Wiley, 2001. ISBN 9780471394426.

EATON, J.W.; BATEMAN, D.; HAUBERG, S. **GNU Octave: A High-level Interactive Language for Numerical Computations**. [S.l.]: Network Theory Limited, 2008. (GNU manual). ISBN 9780954612061.

ESFANDIARI, R.S. **Numerical Methods for Engineers and Scientists Using MATLAB®**. [S.l.]: CRC Press, 2017. ISBN 9781498777452.

INMAN, D.J.; SINGH, R.C. **Engineering Vibration**. [S.l.]: Pearson, 2014. (Always Learning). ISBN 9780273768449.

JOHNSON, Jeff. **GUI Bloopers 3.0: Common User Interface Design Don'ts and Dos**. 3. ed. [S.l.]: Morgan Kaufmann, 2020. Discute princípios de design visual, escolha de cores e tipografia para interfaces profissionais.

KREYSZIG, E. **Advanced Engineering Mathematics**. [S.l.]: Wiley, 2020. ISBN 9781119455929.

KULAKOWSKI, B.T.; GARDNER, J.F.; SHEARER, J.L. **Dynamic Modeling and Control of Engineering Systems**. [S.l.]: Cambridge University Press, 2007. ISBN 9780521864350.

MACCARI JUNIOR, Luiz Antonio. Lista de Exercícios de Simulação de Sistemas. Material didático da disciplina de Modelagem e Simulação de Processos. [S.l.], 2025.

MATHWORKS. **Control System Toolbox**. Official MATLAB documentation. 2023. Disponível em: <https://www.mathworks.com/help/control/>. Acesso em: 8 jun. 2025.

MATHWORKS. **MATLAB - MathWorks**. Official MATLAB website. 2023. Disponível em: <https://www.mathworks.com/products/matlab.html>. Acesso em: 8 jun. 2025.

MOLER, C.B. **Numerical Computing with MATLAB: Revised Reprint**. [S.l.]: Society for Industrial e Applied Mathematics, 2010. (Other Titles in Applied Mathematics). ISBN 9780898716603.

OCTAVE COMMUNITY. **GNU Octave**. Official Octave website. 2023. Disponível em: <https://octave.org/>. Acesso em: 8 jun. 2025.

OCTAVE CONTRIBUTORS. **Control Package for GNU Octave**. Control package for linear systems in Octave. 2023. Disponível em: <https://octave.sourceforge.io/control/>. Acesso em: 8 jun. 2025.

OGATA, K. **Modern Control Engineering**. [S.l.]: Pearson Education, Limited, 2014. ISBN 9781292025834.

PENG, R. D. Reproducible Research in Computational Science. **Science**, v. 334, n. 6060, p. 1226–1227, 2011.

PÉREZ, Fernando; GRANGER, Brian E.; HUNTER, John D. Python: An Ecosystem for Scientific Computing. **Computing in Science Engineering**, v. 13, n. 2, p. 13–21, 2011.

QUARTERONI, A.; SACCO, R.; SALERI, F. **Numerical Mathematics**. [S.l.]: Springer Berlin Heidelberg, 2010. (Texts in Applied Mathematics). ISBN 9783642071010.

RAO, S.S. **Mechanical Vibrations**. [S.l.]: Prentice Hall, 2011. (Mechanical Vibrations, v. 978, N° 0-212813). ISBN 9780132128193.

SCILAB ENTERPRISES. **Scilab: Open Source Software for Numerical Computing**. Official Scilab website. 2023. Disponível em: <https://www.scilab.org/>. Acesso em: 8 jun. 2025.

APÊNDICE A – Código main_v9.m

A seguir, apresenta-se a implementação completa do código principal **main_v9.m** desenvolvido no *software* GNU Octave.

Código A.1 – Implementação do código main_v9.

```

1 close all; clc; clear all;
2 %-----
3 % DEFINICAO DO SISTEMA
4 %-----
5 A = [-1 -2 -3;
6      3 -4 -5;
7      0 4 -2];
8 B = [1; 1; 0];
9 C = [1 0 0];
10 D = [0];
11
12 sys = ss(A, B, C, D);
13
14 %-----
15 % PARAMETROS, ENTRADA E CONDICAOES INICIAIS DO SISTEMA
16 %-----
17
18 t0 = 0;
19 tf = 4;
20 dt = 0.01;
21 t = t0:dt:tf;
22
23 na = length(t);
24 u = @(t) double(t >= 0);
25 x0 = [0; 0; 0];
26
27 %-----
28 % CHAMADA DAS FUNCOES DOS METODOS DE EULER E RK
29 %-----
30
31 x_explicit = euler_explicit_v9(sys, u, x0, dt, t);
32 x_implicit = euler_implicit_v9(sys, u, x0, dt, t);
33 x_rk_ponto_medio = rk_v9(sys, u, x0, dt, t, 'ponto_medio');
34 x_rk_ralston = rk_v9(sys, u, x0, dt, t, 'ralston');
35 x_rk_heun = rk_v9(sys, u, x0, dt, t, 'heun');
36 x_rk3_classico = rk_v9(sys, u, x0, dt, t, 'rk3_classico');
37 x_rk3_ralston = rk_v9(sys, u, x0, dt, t, 'rk3_ralston');
38 x_rk3_heun = rk_v9(sys, u, x0, dt, t, 'rk3_heun');
39 x_rk4_classico = rk_v9(sys, u, x0, dt, t, 'rk4_classico');
40 x_rk4_ralston = rk_v9(sys, u, x0, dt, t, 'rk4_ralston');
41 x_rk4_38rule = rk_v9(sys, u, x0, dt, t, 'rk4_38rule');

```

```

42
43 % Simulacao dos Metodos
44
45 fprintf('\nMetodos disponiveis: explicit, implicit, rk2_ponto_medio,
      rk2_ralston, rk2_heun, rk3_classico, rk3_ralston, rk3_heun,
      rk4_classico, rk4_ralston e rk4_38rule\n\n');
46 [y_explicit,x_explicit] = metodos_v9(sys, u, x0, dt, t, 'explicit');
47 [y_implicit,x_implicit] = metodos_v9(sys, u, x0, dt, t, 'implicit');
48 [y_rk_ponto_medio, x_rk_ponto_medio] = metodos_v9(sys, u, x0, dt, t, '
      rk2_ponto_medio');
49 [y_rk_ralston, x_rk_ralston] = metodos_v9(sys, u, x0, dt, t, '
      rk2_ralston');
50 [y_rk_heun, x_rk_heun] = metodos_v9(sys, u, x0, dt, t, 'rk2_heun');
51 [y_rk3_classico, x_rk3_classico] = metodos_v9(sys, u, x0, dt, t, '
      rk3_classico');
52 [y_rk3_ralston, x_rk3_ralston] = metodos_v9(sys, u, x0, dt, t, '
      rk3_ralston');
53 [y_rk3_heun, x_rk3_heun] = metodos_v9(sys, u, x0, dt, t, 'rk3_heun');
54 [y_rk4_classico, x_rk4_classico] = metodos_v9(sys, u, x0, dt, t, '
      rk4_classico');
55 [y_rk4_ralston, x_rk4_ralston] = metodos_v9(sys, u, x0, dt, t, '
      rk4_ralston');
56 [y_rk4_38rule, x_rk4_38rule] = metodos_v9(sys, u, x0, dt, t, 'rk4_38rule
      ');
57
58 %-----
59 % IMPLEMENTACAO LSIM
60 %-----
61
62 in_lsim = u(t);
63 [y_lsim, t_lsim, x_lsim] = lsim(sys, in_lsim, t);
64
65 %-----
66 % FORMATACAO DA TABELA DE RESULTADOS
67 %-----
68 x_explicit = x_explicit(:, 1:na);
69 x_implicit = x_implicit(:, 1:na);
70 x_rk_ponto_medio = x_rk_ponto_medio(:, 1:na);
71 x_rk_ralston = x_rk_ralston(:, 1:na);
72 x_rk_heun = x_rk_heun(:, 1:na);
73 x_rk3_classico = x_rk3_classico(:, 1:na);
74 x_rk3_ralston = x_rk3_ralston(:, 1:na);
75 x_rk3_heun = x_rk3_heun(:, 1:na);
76 x_rk4_classico = x_rk4_classico(:, 1:na);
77 x_rk4_ralston = x_rk4_ralston(:, 1:na);
78 x_rk4_38rule = x_rk4_38rule(:, 1:na);
79

```

```

80 % Arredondamento dos resultados para 8 casas decimais
81
82 n = 8;
83 x_explicit_rounded = round(x_explicit * 10^n)/10^n;
84 x_implicit_rounded = round(x_implicit * 10^n)/10^n;
85 x_rk_ponto_medio_rounded = round(x_rk_ponto_medio * 10^n) / 10^n;
86 x_rk_ralston_rounded = round(x_rk_ralston * 10^n) / 10^n;
87 x_rk_heun_rounded = round(x_rk_heun * 10^n) / 10^n;
88 x_rk3_classico_rounded = round(x_rk3_classico * 10^n) / 10^n;
89 x_rk3_ralston_rounded = round(x_rk3_ralston * 10^n) / 10^n;
90 x_rk3_heun_rounded = round(x_rk3_heun * 10^n) / 10^n;
91 x_rk4_classico_rounded = round(x_rk4_classico * 10^n) / 10^n;
92 x_rk4_ralston_rounded = round(x_rk4_ralston * 10^n) / 10^n;
93 x_rk4_38rule_rounded = round(x_rk4_38rule * 10^n) / 10^n;
94
95 % Formatacao da tabela
96
97 results_table = table(t', x_explicit_rounded(1,:), x_implicit_rounded(1,
    ,:)', x_rk_ponto_medio_rounded(1,:),...
98 x_rk_ralston_rounded(1,:), x_rk_heun_rounded(1,:),
    x_rk3_classico_rounded(1,:), x_rk3_ralston_rounded(1,:),...
99 x_rk3_heun_rounded(1,:), x_rk4_classico_rounded(1,:),
    x_rk4_ralston_rounded(1,:), x_rk4_38rule_rounded(1,:));
100
101 results_table.Properties.VariableNames = {'Tempo', 'Euler_Explicito', '
    Euler_Implicito', 'RK2_Ponto_Medio', 'RK2_Ralston', 'RK2_Heun', '
    RK3_Classico', 'RK3_Ralston', 'RK3_Heun', 'RK4_Classico', 'RK4_Ralston',
    'RK4_38rule'};
102
103 % Exibir a tabela com fprintf para 8 casas decimais
104 fprintf('\nResultados da Simulacao:\n');
105 fprintf('%-10s %-15s %-15s %-15s %-15s %-15s %-15s %-15s %-15s %-
    15s %-15s\n', ...
106         'Tempo', 'Euler_Expl', 'Euler_Impl', 'RK2_PMedio', 'RK2_Ralston'
    , 'RK2_Heun', 'RK3_Classico', 'RK3_Ralston', 'RK3_Heun', 'RK4_Classico
    ', 'RK4_Ralston', 'RK4_38rule');
107
108 for i = 1:size(results_table, 1)
109     fprintf('%-10.4f %-15.8f %-15.8f %-15.8f %-15.8f %-15.8f %-15.8f %-
    15.8f %-15.8f %-15.8f %-15.8f\n', ...
110         results_table.Tempo(i), ...
111         results_table.Euler_Explicito(i), ...
112         results_table.Euler_Implicito(i), ...
113         results_table.RK2_Ponto_Medio(i), ...
114         results_table.RK2_Ralston(i), ...
115         results_table.RK2_Heun(i), ...
116         results_table.RK3_Classico(i), ...

```

```

117         results_table.RK3_Ralston(i), ...
118         results_table.RK3_Heun(i));
119         results_table.RK4_Classico(i), ...
120         results_table.RK4_Ralston(i), ...
121         results_table.RK4_38rule(i);
122     end
123
124 %-----
125 % BLOCO DE PLOTAGEM E CONFIGURACOES DA PLOTAGEM
126 %-----
127
128 figure;
129
130 % Plotagem das saidas
131
132 subplot(2, 1, 1);
133 hold on;
134 plot(t, u(t), 'LineWidth', 3, 'Color', [0.8500 0.3250 0.0980]);
135 plot(t, y_explicit, 'c--', 'LineWidth', 2);
136 plot(t, y_implicit, 'm--', 'LineWidth', 2);
137 plot(t, y_rk_ponto_medio, 'g--', 'LineWidth', 2);
138 plot(t, y_rk_ralston, 'r--', 'LineWidth', 2);
139 plot(t, y_rk_heun, 'b--', 'LineWidth', 2);
140 plot(t, y_rk3_classico, ':', 'LineWidth', 2);
141 plot(t, y_rk3_ralston, '-.', 'LineWidth', 2);
142 plot(t, y_rk3_heun, '--', 'LineWidth', 2);
143 plot(t, y_rk4_classico, ':', 'LineWidth', 2);
144 plot(t, y_rk4_ralston, '-.', 'LineWidth', 2);
145 plot(t, y_rk4_38rule, '--', 'LineWidth', 2);
146 plot(t_lsim, y_lsim, '--', 'LineWidth', 2);
147
148 hold off;
149 title('Comparacao da Entrada ao Degrau e das Saidas Y dos metodos e Lsim
        ');
150 xlabel('Tempo (k)');
151 ylabel('y(k)');
152 legend('Entrada ao degrau','Euler Explícito', 'Euler Implícito', 'RK2
        Ponto Medio', 'RK2 Ralston', 'RK2 Heun', 'RK3 Classico', 'RK3 Ralston
        ', 'RK3 Heun', 'RK4 Classico', 'RK4 Ralston', 'RK4 3/8 Rule', 'Lsim')
        ;
153 grid on;
154
155 % Plotagem Curvas dos estados x1, x2, x3
156 subplot(2, 1, 2);
157 hold on;
158 plot(t, x_explicit(1, :), 'y', 'LineWidth', 2);
159 plot(t, x_implicit(1, :), 'm--', 'LineWidth', 2);

```

```

160 plot(t, x_rk_ponto_medio(1, :), 'g--', 'LineWidth', 2);
161 plot(t, x_rk_ralston(1, :), 'r--', 'LineWidth', 2);
162 plot(t, x_rk_heun(1, :), 'b--', 'LineWidth', 2);
163 plot(t, x_rk3_classico(1, :), ':', 'LineWidth', 2);
164 plot(t, x_rk3_ralston(1, :), '-.', 'LineWidth', 2);
165 plot(t, x_rk3_heun(1, :), '--', 'LineWidth', 2);
166 plot(t, x_rk4_classico(1, :), 'b--', 'LineWidth', 2);
167 plot(t, x_rk4_ralston(1, :), 'c--', 'LineWidth', 2);
168 plot(t, x_rk4_38rule(1, :), 'm--', 'LineWidth', 2);
169 plot(t, x_explicit(2, :), 'g', 'LineWidth', 2);
170 plot(t, x_implicit(2, :), 'k--', 'LineWidth', 2);
171 plot(t, x_rk_ponto_medio(2, :), 'c--', 'LineWidth', 2);
172 plot(t, x_rk_ralston(2, :), 'm--', 'LineWidth', 2);
173 plot(t, x_rk_heun(2, :), 'LineWidth', 2);
174 plot(t, x_rk3_classico(2, :), ':', 'LineWidth', 2);
175 plot(t, x_rk3_ralston(2, :), '-.', 'LineWidth', 2);
176 plot(t, x_rk3_heun(2, :), '--', 'LineWidth', 2);
177 plot(t, x_rk4_classico(2, :), 'g--', 'LineWidth', 2);
178 plot(t, x_rk4_ralston(2, :), 'k--', 'LineWidth', 2);
179 plot(t, x_rk4_38rule(2, :), '--', 'LineWidth', 2);
180 plot(t, x_explicit(3, :), 'b', 'LineWidth', 2);
181 plot(t, x_implicit(3, :), 'c--', 'LineWidth', 2);
182 plot(t, x_rk_ponto_medio(3, :), 'r--', 'LineWidth', 2);
183 plot(t, x_rk_ralston(3, :), 'g--', 'LineWidth', 2);
184 plot(t, x_rk_heun(3, :), 'k--', 'LineWidth', 2);
185 plot(t, x_rk3_classico(3, :), ':', 'LineWidth', 2);
186 plot(t, x_rk3_ralston(3, :), '-.', 'LineWidth', 2);
187 plot(t, x_rk3_heun(3, :), '--', 'LineWidth', 2);
188 plot(t, x_rk4_classico(3, :), 'y', 'LineWidth', 2);
189 plot(t, x_rk4_ralston(3, :), 'r--', 'LineWidth', 2);
190 plot(t, x_rk4_38rule(3, :), '-.', 'LineWidth', 2);
191 hold off;
192 title('Estados x1, x2, x3 de Euler Explícito, Euler Implícito, RK2 e RK3
      ');
193 xlabel('Tempo (k)');
194 ylabel('Estado x(k)');
195 legend('Euler Explícito x1', 'Euler Implícito x1', 'RK2 Ponto Medio x1',
      'RK2 Ralston x1', 'RK2 Heun x1', 'RK3 Classico x1', 'RK3 Ralston x1',
      'RK3 Heun x1', 'RK4 Classico x1', 'RK4 Ralston x1', 'RK4 3/8 Rule
      x1',...
196      'Euler Explícito x2', 'Euler Implícito x2', 'RK2 Ponto Medio x2',
      'RK2 Ralston x2', 'RK2 Heun x2', 'RK3 Classico x2', 'RK3 Ralston x2',
      'RK3 Heun x2', 'RK4 Classico x2', 'RK4 Ralston x2', 'RK4 3/8 Rule
      x2',...
197      'Euler Explícito x3', 'Euler Implícito x3', 'RK2 Ponto Medio x3',
      'RK2 Ralston x3', 'RK2 Heun x3', 'RK3 Classico x3', 'RK3 Ralston x3',
      'RK3 Heun x3', 'RK4 Classico x3', 'RK4 Ralston x3', 'RK4 3/8 Rule

```

```
    x3');  
198 grid on;  
199  
200 %-----  
201 % PLOTAGEM DOS DIAGRAMAS DE ANALISE  
202 %-----  
203 figure('Name', 'Diagramas de Analise do Sistema', 'Position', [100 100  
    1200 800]);  
204  
205 bode(sys);grid on;title('Bode');  
206  
207 subplot(2,2,1);nyquist(sys);grid on;title('Nyquist');axis equal;  
208  
209 subplot(2,2,2);rlocus(sys);grid on;title('Root Locus');  
210  
211 subplot(2,2,3);pzmap(sys);grid on;title('Pole-Zero Map');
```

Fonte: Elaborado pela autora (2025).

APÊNDICE B – Código rk_v9.m

A seguir, apresenta-se a implementação completa dos métodos de Runge-Kutta no *software* GNU Octave.

Código B.1 – Implementação matricial dos métodos de Runge-Kutta.

```

1 function [x_rk] = rk_v9(sys, u, x0, dt, t, metodo_rk)
2
3 [A, B, C, D] = ssdata(sys);
4
5 na = length(t);
6
7 x_rk = zeros(size(A, 1), na + 1);
8
9 x_rk(:, 1) = x0;
10
11 %-----
12 % TABELA DE BUTCHER PARA RK2 (Ponto Medio, Ralston, Heun)
13 %-----
14 butcher_table = struct();
15
16 butcher_table.ponto_medio = struct(...
17     'c', [0;
18         1/2], ...
19     'a', [0 0;
20         1/2 0], ...
21     'b', [0 1]);
22
23 butcher_table.ralston = struct('c', [0; 3/4], 'a', [0 0; 3/4 0], 'b',
24     [1/3 2/3]);
25
26 butcher_table.heun = struct('c', [0; 1], 'a', [0 0; 1 0], 'b', [1/
27     2 1/2]);
28
29 %-----
30 % TABELA DE BUTCHER PARA RK3 (Classico, Ralston e Heun)
31 %-----
32
33 butcher_table_rk3 = struct();
34
35 butcher_table_rk3.rk3_classico = struct(...
36     'c', [0; 1/2; 1], ...
37     'a', [0 0 0;
38         1/2 0 0;
39         -1 2 0], ...
40     'b', [1/6 2/3 1/6] ...
41 );

```

```

40
41  butcher_table_rk3.rk3_ralston = struct(...)
42      'c', [0; 1/2; 3/4], ...
43      'a', [0      0      0;
44            1/2    0      0;
45            0      3/4    0], ...
46      'b', [2/9    1/3    4/9] ...
47  );
48
49  butcher_table_rk3.rk3_heun = struct(...)
50      'c', [0; 1/3; 2/3], ...
51      'a', [0      0      0;
52            1/3    0      0;
53            0      2/3    0], ...
54      'b', [1/4    0      3/4] ...
55  );
56
57  %-----
58  % TABELA DE BUTCHER PARA RK4 (Classico, Ralston e 3/8)
59  %-----
60
61  butcher_table_rk4 = struct();
62
63  butcher_table_rk4.rk4_classico = struct(...)
64      'c', [0; 1/2; 1/2; 1], ...
65      'a', [0      0      0      0;
66            1/2    0      0      0;
67            0      1/2    0      0;
68            0      0      1      0], ...
69      'b', [1/6    1/3    1/3    1/6] ...
70  );
71
72  butcher_table_rk4.rk4_ralston = struct(...)
73      'c', [0; 0.4; 0.45573725; 1], ...
74      'a', [0      0      0      0;
75            0.4      0      0      0;
76            0.29697761  0.15875964  0      0;
77            0.21810040  -3.05096516  3.83286476  0], ...
78      'b', [0.17476028  -0.55148066  1.20553560  0.17118478] ...
79  );
80
81  butcher_table_rk4.rk4_38rule = struct(...)
82      'c', [0; 1/3; 2/3; 1], ...
83      'a', [0      0      0      0;
84            1/3    0      0      0;
85            -1/3   1      0      0;
86            1      -1     1      0], ...

```

```

87     'b', [1/8  3/8  3/8  1/8] ...
88 );
89
90 %-----
91 % VERIFICACAO DO METODO E CALCULO - RK2
92 %-----
93     if isfield(butcher_table, metodo_rk)
94
95         % Extracao dos coeficientes da tabela de Butcher
96         c = butcher_table.(metodo_rk).c;
97         a = butcher_table.(metodo_rk).a;
98         b = butcher_table.(metodo_rk).b;
99
100        for k = 1:na
101            % Calculo dos estagios (k1, k2, ...)
102            m1 = A * x_rk(:, k) + B * u(t(k));
103            m2 = A * (x_rk(:, k) + a(2,1) * m1 * dt) + B * u(t(k) + c(2)
104            * dt);
105
106            % Atualizacao do estado
107            x_rk(:, k+1) = x_rk(:, k) + (b(1) * m1 + b(2) * m2) * dt;
108        end
109
110 %-----
111 % VERIFICACAO DO METODO E CALCULO - RK3
112 %-----
113
114        elseif isfield(butcher_table_rk3, metodo_rk)
115
116            c = butcher_table_rk3.(metodo_rk).c;
117            a = butcher_table_rk3.(metodo_rk).a;
118            b = butcher_table_rk3.(metodo_rk).b;
119
120        for k = 1:na
121            m1 = A * x_rk(:, k) + B * u(t(k));
122            m2 = A * (x_rk(:, k) + a(2,1) * m1 * dt) + B * u(t(k) + c(2) *
123            dt);
124            m3 = A * (x_rk(:, k) + a(3,1) * m1 * dt + a(3,2) * m2 * dt) + B
125            * u(t(k) + c(3) * dt);
126
127            x_rk(:, k+1) = x_rk(:, k) + (b(1) * m1 + b(2) * m2 + b(3) * m3)
128            * dt;
129        end
130
131 %-----
132 % VERIFICACAO DO METODO E CALCULO - RK4
133 %-----

```

```

130
131     elseif isfield(butcher_table_rk4, metodo_rk)
132
133         c = butcher_table_rk4.(metodo_rk).c;
134         a = butcher_table_rk4.(metodo_rk).a;
135         b = butcher_table_rk4.(metodo_rk).b;
136
137         for k = 1:na
138             m1 = A * x_rk(:, k) + B * u(t(k));
139             m2 = A * (x_rk(:, k) + a(2,1) * m1 * dt) + B * u(t(k) + c(2)
140 * dt);
141             m3 = A * (x_rk(:, k) + a(3,1) * m1 * dt + a(3,2) * m2 * dt)
142 + B * u(t(k) + c(3) * dt);
143             m4 = A * (x_rk(:, k) + a(4,1) * m1 * dt + a(4,2) * m2 * dt +
144 a(4,3) * m3 * dt) + B * u(t(k) + c(4) * dt);
145
146             x_rk(:, k+1) = x_rk(:, k) + (b(1) * m1 + b(2) * m2 + b(3) *
147 m3 + b(4) * m4) * dt;
148         end
149
150     else
151         error(['Metodo nao reconhecido. Use: \n' ...
152             'RK2: "ponto_medio", "ralston" ou "heun" \n' ...
153             'RK3: "rk3_classico", "rk3_ralston" ou "rk3_heun" \n' ...
154             'RK4: "rk4_classico", "rk4_ralston" ou "rk4_38rule".']);
155     end
156 end

```

Fonte: Elaborado pela autora (2025).

APÊNDICE C – Código metodos_v9.m

A seguir, apresenta-se a implementação completa do código **metodos_v9.m** no *software* GNU Octave.

Código C.1 – Implementação matricial dos métodos de Runge-Kutta.

```

1 function [y, x] = metodos_v9(sys, u, x0, dt, t, metodo)
2
3     switch metodo
4         %-----
5         % METODOS DE EULER
6         %-----
7         case 'explicit'
8             fprintf('Metodo escolhido: Euler Explicito\n');
9             x = euler_explicit_v9(sys, u, x0, dt, t);
10
11        case 'implicit'
12            fprintf('Metodo escolhido: Euler Implicito\n');
13            x = euler_implicit_v9(sys, u, x0, dt, t);
14
15        %-----
16        % METODOS DE RK2
17        %-----
18        case 'rk2_ponto_medio'
19            fprintf('Metodo escolhido: RK2 Ponto Medio\n');
20            x = rk_v9(sys, u, x0, dt, t, 'ponto_medio');
21
22        case 'rk2_ralston'
23            fprintf('Metodo escolhido: RK2 Ralston\n');
24            x = rk_v9(sys, u, x0, dt, t, 'ralston');
25
26        case 'rk2_heun'
27            fprintf('Metodo escolhido: RK2 Heun\n');
28            x = rk_v9(sys, u, x0, dt, t, 'heun');
29
30        %-----
31        % METODOS DE RK3
32        %-----
33        case 'rk3_classico'
34            fprintf('Metodo escolhido: RK3 Classico\n');
35            x = rk_v9(sys, u, x0, dt, t, 'rk3_classico');
36
37        case 'rk3_ralston'
38            fprintf('Metodo escolhido: RK3 Ralston\n');
39            x = rk_v9(sys, u, x0, dt, t, 'rk3_ralston');
40
41        case 'rk3_heun'

```

```

42     fprintf('Metodo escolhido: RK3 Heun\n');
43     x = rk_v9(sys, u, x0, dt, t, 'rk3_heun');
44
45 %-----
46 % METODOS DE RK4
47 %-----
48     case 'rk4_classico'
49         fprintf('Metodo escolhido: RK4 Classico\n');
50         x = rk_v9(sys, u, x0, dt, t, 'rk4_classico');
51
52     case 'rk4_ralston'
53         fprintf('Metodo escolhido: RK4 Ralston\n');
54         x = rk_v9(sys, u, x0, dt, t, 'rk4_classico');
55
56     case 'rk4_38rule'
57         fprintf('Metodo escolhido: RK4 Regra dos 3/8\n');
58         x = rk_v9(sys, u, x0, dt, t, 'rk4_38rule');
59
60     otherwise
61         error(['Metodo desconhecido. Escolha entre: \n'...
62             'Euler: "explicit", "implicit" \n'...
63             'RK2: "rk2_ponto_medio", "rk2_ralston", "rk2_heun" \n'...
64             'RK3: "rk3_classico", "rk3_ralston", "rk3_heun" \n'...
65             'RK4: "rk4_classico", "rk4_ralston" ou "rk4_38rule".']);
66 end
67
68 %-----
69 % BLOCO DE CONFIGURACOES PARA CALCULO DAS SAIDAS
70 %-----
71     in = u(t);
72
73     [A, B, C, D] = ssdata(sys);
74
75     na = length(t);
76
77     x = x(:, 1:na);
78
79     y = C * x + D * in;
80
81 end

```

Fonte: Elaborado pela autora (2025).

ANEXO A – Tabelas de Butcher

A seguir, apresentam-se todas as tabelas de Butcher dos métodos de Runge-Kutta de ordem superior (RK2, RK3, RK4) que foram utilizadas na implementação do *software Numerical Methods Interface*.

MÉTODOS DE RK2:

- Ponto Médio (*Midpoint*):

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array} \quad (92)$$

- Ralston:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{3}{4} & \frac{3}{4} & 0 \\ \hline & \frac{1}{3} & \frac{2}{3} \end{array} \quad (93)$$

- Heun:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \quad (94)$$

MÉTODOS DE RK3:

- Clássico:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 1 & -1 & 2 & 0 \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \quad (95)$$

- Ralston:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{3}{4} & 0 & \frac{3}{4} & 0 \\ \hline & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} \end{array} \quad (96)$$

- Heun:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{2}{3} & 0 & \frac{2}{3} & 0 \\ \hline & \frac{1}{4} & 0 & \frac{3}{4} \end{array} \quad (97)$$

MÉTODOS DE RK4:• **Clássico:**

$$\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\
\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
\hline
& \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array} \tag{98}$$

• **Ralston:**

$$\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
0.4 & 0.4 & 0 & 0 & 0 \\
0.45573725 & 0.29697761 & 0.15875964 & 0 & 0 \\
1 & 0.21810040 & -3.05096516 & 3.83286476 & 0 \\
\hline
& 0.17476028 & -0.55148066 & 1.20553560 & 0.17118478
\end{array} \tag{99}$$

• **3/8-Rule:**

$$\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
\frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\
\frac{2}{3} & -\frac{1}{3} & 1 & 0 & 0 \\
1 & 1 & -1 & 1 & 0 \\
\hline
& \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8}
\end{array} \tag{100}$$