UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS

FILIPE ANTUNES GARCEZ

Projeto de Fim de Curso

REFORMULAÇÃO DE UMA FERRAMENTA VOLTADA À MODELAGEM E SIMULAÇÃO DE SISTEMAS DINÂMICOS

Florianópolis

2019

FILIPE ANTUNES GARCEZ

REFORMULAÇÃO DE UMA FERRAMENTA VOLTADA À MODELAGEM E SIMULAÇÃO DE SISTEMAS DINÂMICOS

Relatório submetido à Universidade Federal de Santa Catarina como requisito para a aprovação na disciplina **DAS 5511 - Projeto de Fim de Curso** do curso de Graduação em Engenharia de Controle e Automação.
Orientador: Prof. Julio Elias Normey Rico

Florianópolis
2019

FILIPE ANTUNES GARCEZ

REFORMULAÇÃO DE UMA FERRAMENTA VOLTADA À MODELAGEM E SIMULAÇÃO DE SISTEMAS DINÂMICOS

Esta monografia foi julgada no contexto da disciplina DAS5511: Projeto de Fim de Curso e aprovada na sua forma final pelo Curso de Engenharia de Controle e Automação.

Florianópolis, 01 de agosto de 2019

Prof. Julio Elias Normey Rico
Orientador
Universidade Federal de Santa Catarina

Me. Rafael Sartori
Supervisor no Grupo de Pesquisa
SCoPI - UFSC

FILIPE ANTUNES GARCEZ

REFORMULAÇÃO DE UMA FERRAMENTA VOLTADA À MODELAGEM E SIMULAÇÃO DE SISTEMAS DINÂMICOS

Banca Examinadora:

Prof. Hector Bessa Silveira
Responsável pela disciplina
Universidade Federal de Santa Catarina

Me. Mauricio Pereira Dal Pont
Avaliador
Universidade Federal de Santa Catarina

Eng. Gabriel Bruzaca Cavalcante
Debatedor
Universidade Federal de Santa Catarina

Eng. José Silvan Batista Mota Júnior
Debatedor
Universidade Federal de Santa Catarina

Resumo

Um simulador de processos dinâmicos é, certamente, uma ferramenta valiosa para a solução de problemas em engenharia. O uso de ferramentas deste tipo já acumula mais de 50 anos de história, surgindo como artifícios eletrônicos e se estabelecendo atualmente como ferramentas de modelagem e simulação orientadas a equações. Este trabalho visa apresentar os desafios e soluções associadas ao processo de retificação e aprimoramento de uma ferramenta de simulação de sistemas dinâmicos baseados em equações. Apesar de tal tecnologia se mostrar eficiente na caracterização de sistemas, esta traz consigo algumas complexidades associadas ao seu carácter estrutural. Para tanto, foram estudadas a fundo técnicas de depuração de sistemas que visam contornar estes problemas. A reformulação da ferramenta também conta com um pacote próprio de objetos e métodos para caracterização e manipulação simbólica de conceitos matemáticos como variáveis, expressões e equações. Este pacote é de fundamental importância para a definição e ajuste de modelos dinâmicos para simulação. A validação da ferramenta reformulada foi feita através da execução e avaliação de ensaios numéricos, apresentando resultados consistentes em todos eles.

Palavras-chave: Modelagem de equipamentos. Simulação numérica. Modelos modulares. Indústria de petróleo e gás. Depuração de sistemas de equações. Sistemas *DAE*. Sistemas *NLA*. Sistemas booleanos. Sistemas discretos. Linguagem de programação *Python*. Código aberto.

Abstract

A dynamic process simulator is certainly a valuable tool for engineering problem solving. The use of these tools has accumulated over 50 years of history, emerging from electronic devices and establishing itself today as equation-oriented modeling and simulation tools. This report aims to present the challenges and solutions associated with the rectification and improvement process of an equation-oriented dynamic systems simulation tool. Although such technology proves to be efficient in characterizing dynamic systems, it brings with it some complexities associated with its structural aspect. For this purpose, systems debugging techniques that seek to solve these problems have been studied. The redesigned tool also has its own package of objects and methods for the characterization and symbolic manipulation of mathematical concepts such as variables, expressions and equations. This package is of fundamental importance for defining and adjusting dynamic models for simulation. The validation of the reformulated tool was done through the execution and evaluation of numerical tests, presenting consistent results in all of them.

Key-words: Equipment modeling. Numerical simulation. Modular models. Oil and gas industry. Equation systems debugging. DAE systems. NLA systems. Boolean systems. Discrete systems. Python programming language. Open-source.

Lista de ilustrações

Lista de tabelas

Lista de algoritmos

Sumário

1 Introdução

A medida que o tempo passa, cada vez mais se utiliza de artifícios computacionais para solucionar problemas em engenharia. Especialmente em áreas ligadas a modelagem e simulação de sistemas dinâmicos, o uso de ferramentas computacionais para obtenção de resultados numéricos é essencial e tem evoluído consideravelmente. Ao longo dos anos, é visível uma tendência de migração das tão extensivamente utilizadas ferramentas de modelagem modulares sequenciais para ferramentas baseadas em equações.

Grande parte das ferramentas modulares sequenciais surgiram por meio de grandes companhias do ramo químico e petroquímico durante a década de 60, somando mais de 200 pacotes diferentes (WESTERBERG, 1998). Com o passar do tempo, tais ferramentas foram sendo abandonadas, sedendo espaço a uma nova tecnologia de modelagem que vinha se desenvolvendo de forma paralela por diversos pesquisadores do meio acadêmico. Esta nova tecnologia é o que hoje se chamada de modelagem orientada a equações.

O paradigma baseado em equações estabelece uma forma consideravelmente mais simples de modelagem em detrimento da simplicidade de simulação. Um modelo de processo orientado a equação se resume, basicamente, no conjunto de suas equações descritivas, que podem estar todas reunidas ou espalhadas em diferentes modelos de equipamentos. Para que um modelo orientado a equações possa ser numericamente simulado, não só é necessário concatenar todas as equações em um único modelo, mas também ajusta-lo para que este produza resultados confiáveis.

Este trabalho visa a reformulação de uma ferramenta orientada a equações desenvolvida como um projeto de estágio em Garcez (2019). Tal ferramenta foi desenvolvida como uma biblioteca para a linguagem de programação *python* e tem sua aplicação voltada a modelagem e simulação de processos no contexto da indústria de petróleo e gás.

1.1 Problemática

Os motivos que levaram a necessidade de reformulação da ferramenta produzida em Garcez (2019) variam desde pequenas limitações a problemas sérios de consistência. A fragilidade mais séria com relação à ferramenta é que esta considerava que modelos de processos compostos por equações diferenciais e equações algébricas não-lineares poderiam ser simulados através da solução sequencial de uma sistema de equações ordinárias e um sistema de equações não lineares, o que não é verdade para a maioria dos sistemas. Tal consideração compromete a confiabilidade dos resultados, o que é um requisito básico de ferramentas de simulação.

Outra fragilidade da ferramenta em questão diz respeito a sua dificuldade de inicialização das variáveis do modelo de processo. Ao contrário do que se espera de uma boa ferramenta de simulação, os algoritmos de solução numérica desta eram fortemente dependentes de uma boa estimativa inicial de solução, uma vez que a ferramenta desenvolvida não provia suporte a derivação simbólica para a determinação da matriz jacobiana do modelo, informação essa que é essencial para guiar a convergência destes algoritmos.

Por fim, existe um aspecto que pode até parecer irrelevante em casos de sistemas simples, mas que é de grande valor em sistemas complexos é a presença de suporte ao usuário quanto a determinação de problemas estruturais em modelos. Suporte este que estava ausente na ferramenta alvo desta reformulação.

1.2 Objetivo

Uma vez elencadas as fragilidades e limitações da ferramenta a ser reformulada, estabelece-se como objetivo geral deste trabalho a retificação e aprimoramento desta para que se obtenha uma ferramenta de modelagem e simulação verdadeiramente confiável, com alta capacidade de representação e ajustada de forma a oferecer uma experiencia de modelagem simples de processos e equipamentos dinâmicos.

Para que se resolvam os problemas citados na seção 1.1, será necessária uma reformulação quase que total da ferramenta, preservando apenas algumas características como, por exemplo, sua estruturação modular e parametrizada de equipamentos. Além disso, a reformulação visa proporcionar uma expansão em termos de funcionalidades e poder de representação. Considerando tudo o que foi citado até aqui, pode-se dizer que os objetivos específicos deste trabalho são:

1. Aumentar o poder de representação e simulação de sistemas por meio de suporte à modelagem de fenômenos contínuos, discretos e booleanos;
2. Aplicar metodologias de análise e depuração de sistemas de equações matemáticas afim de auxiliar na modelagem e definições de processos e equipamentos;
3. Desenvolver todo um ferramental de matemática simbólica para não só melhorar a inserção de modelos na ferramenta, mas também prover métodos de derivação simbólica para solucionar problemas ligados a simulação;
4. Ajustar detalhes de uso da ferramenta conforme a necessidade dos usuários.

1.3 Estrutura do documento

Todo o conteúdo deste relatório encontra-se dividido e estruturado em cinco capítulos, sendo o primeiro deles este capítulo de introdução. O conteúdo dos capítulos que se sucedem foram concebidos e devidamente sequenciados de forma a estabelecer uma linha de raciocínio contínua a respeito do processo de reformulação da ferramenta. Tudo começa por uma fundamentação teórica, conteúdo do Capítulo 2, que busca apresentar os conhecimentos de engenharia de software e de simulação de sistemas dinâmicos que serão necessários para entender os detalhes da ferramenta. Tal detalhamento é feito no capítulo seguinte, Capítulo 3, que dividirá a ferramenta em quatro partes e explicará cada uma delas. Na sequência, Capítulo 4, será efetuada a validação dos resultados produzidos pela ferramenta. Esta validação acontecerá por meio do estudo de alguns exemplos. Por fim, tem-se no Capítulo 5 as conclusões a respeito do trabalho e suas perspectivas futuras.

2 Fundamentação teórica

Como mencionado no capítulo introdutório, para que se possa entender todos os aspectos e nuances a respeito da ferramenta reformulada, faz-se necessário que, primeiro, os conhecimentos utilizados em tal reformulação sejam apresentados e devidamente estudados. Estes conceitos se dividem em duas categorias: os que dizem respeito à engenharia de software e os tratam de simulação de sistemas dinâmicos.

2.1 Engenharia de software

Na computação, a engenharia de software é a área que abrange todo o conhecimento voltado a especificação, desenvolvimento, manutenção e criação de software (ENGENHARIA..., 2019). Alguns conceitos de engenharia de software foram utilizados neste trabalho e serão apresentados nas subseções 2.1.1 e 2.1.2.

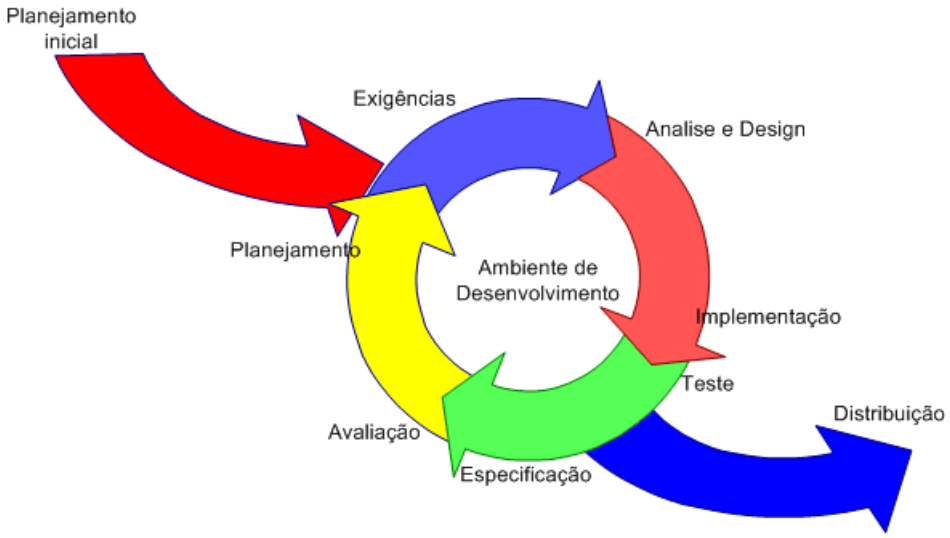
2.1.1 Desenvolvimento iterativo

Durante o processo de produção de ferramentas de software, é comum que se encontrem problemas estruturais ou de funcionalidade que acarretem em reprogramação parcial da ferramenta. Tais problemas são especialmente frequentes no caso de produção de softwares verdadeiramente complexos. O impacto gerado por estes problemas varia com relação ao tamanho e complexidade da porção de software a ser retrabalhada. Em busca de minimizar o impacto destes eventos e conceber um produto de software de qualidade, é recomendado que se estabeleça um processo de desenvolvimento de software.

Uma vez que a ferramenta desenvolvida é uma reformulação quase total de uma ferramenta anterior, uma grande parcela do software é nova e certamente levará a muitos problemas de reprogramação. Devido a certeza destes problemas, considerou-se adequada a utilização de um processo de desenvolvimento de software iterativo.

O desenvolvimento iterativo consiste em obter um produto final de software a partir de iterações sucessivas de trabalho em um produto inicial. Em cada iteração, se efetua um refinamento de todo o produto inicial, concebendo assim versões funcionais de software que, ao longo das iterações, irão convergir para o produto final. Todo o processo é subdividido em etapas as quais são apresentadas na Figura 1. Em casos de criação, o produto inicial de software é inexistente, sendo, portanto, necessária a construção do software deste o seu princípio. No caso deste projeto, o produto inicial é a mencionada ferramenta produzida como projeto de estágio em Garcez (2019).

Figura 1 – Etapas do processo de desenvolvimento iterativo



Fonte: (MOCELIN, 2011)

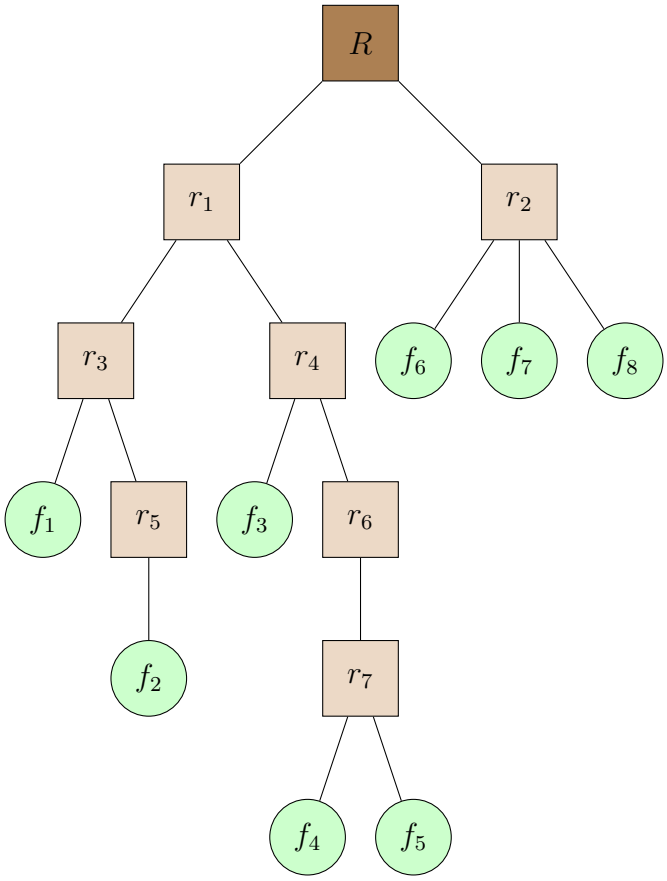
2.1.2 Estrutura de dados: árvores n-árias

Os elementos denominados árvores são estruturas de dados bastante importantes no contexto de programação e engenharia de software. Consiste de uma estrutura não linear de dados organizados de forma hierárquica, estabelecendo relações do tipo pai e filho entre seus elementos. Os conhecimento acerca desta estrutura encontram-se referenciados em Árvore... (2019). A Figura 2 apresenta uma das possíveis representações gráficas para árvores, a forma hierárquica.

Na Figura 2, os elementos R, r_n e f_n são chamados nodos e representam os dados contidos na árvore. Estes nodos podem ser divididos em três subclasses: raiz, ramo e folha. Toda árvore tem sua origem em um nodo do tipo raiz, representado pelo nodo R na Figura 2, e termina em nodos do tipo folha, representado pelos nodos de f_1 a f_8. Os elementos que não são nem a origem da árvore e nem suas terminações são chamados ramos, representados pelos nodos de r_1 a r_7.

Na Figura 2, as linhas que conectam os nodos são chamadas relações e a terminologia mais utilizada para descrevê-las é a mesma para árvores genealógicas. Tomando como exemplo as relações entre os nodos R, r_1 e r_2, o nodo R seria classificado como o nodo pai de r_1 e r_2. Da mesma forma, r_1 e r_2 seriam os nodos filhos de R e, por consequência, seriam também chamados de nodos irmãos. Esta terminologia se estende a todas as relações conferindo status de pai, filho e irmãos a todos os nodos da árvore. Vale observar que, devido às suas definições, um nodo raiz jamais será filho de outro nodo, bem como um

Figura 2 – Representação gráfica de uma árvore de dados



nodo folha jamais será pai de outrem. Os únicos nodos que podem desempenhar tanto o papel de pai quanto o papel de filho são os nodos do tipo ramo.

Um aspecto notável com relação aos ramos é que, caso sejam interpretados como uma raiz, eles caracterizam sub-árvores dentro da árvore original. Desta forma, uma árvore pode ser vista como uma composição de outras árvores menores. Observa-se que esta definição é de natureza recursiva e, por conta disso, dá margem a implementação de diversas funções e rotinas recursivas que caminham através dos ramos da árvore para obter seus resultados.

Existem variados tipos de árvores que surgem quando se impõe restrições quanto a organização dos dados ou a natureza deles. Como exemplo tem-se o caso das árvores binárias, que limita para dois o número máximo de filhos que um nodo pode possuir. Outros exemplos são as árvores B, AVL, 2-3 e muitas outras com suas determinadas características e aplicações. Quando uma árvore não impõe limites ao número máximo de nodos filhos que um nodo pode possuir, esta árvore é dita n-ária. O exemplo da Figura 2 pode muito bem ser uma árvore n-ária, como também pode ser uma árvore ternária, limitando o número total de filhos a, no máximo, três.

As árvores, em um contexto geral, possuem diversas aplicações e geralmente estão associadas a ganhos de performance em sistemas computacionais que lidam com um grande volume de dados. Outras aplicações mais simples se resumem a apenas modelar relações de hierarquia ou de pertencimento. Este é o caso deste trabalho, que utilizará desta estruturas para modelar conceitos matemáticos e operá-los afim de se desenvolver todo um ferramental de matemática simbólica para a ferramenta de modelagem e simulação.

2.2 Simulação de sistemas dinâmicos

Existem muitos conceitos matemáticos e noções a respeito de simulação de sistemas dinâmicos que serão requeridos ao longo deste trabalho para que se entenda a complexidade dos problemas e as ideias de solução por trás dos algoritmos implementados na reformulação da ferramenta de modelagem e simulação. Todos os conhecimentos apresentados nesta seção, serão utilizados nas seções 3.1, 3.3 e 3.4, onde se detalhará as camadas da ferramenta desenvolvida.

2.2.1 Comparação entre sistemas *DAE* e sistemas *ODE*

As equações diferenciais são amplamente utilizadas em áreas como a engenharia ou a física para modelar relações entre grandezas quantitativas que são afetadas não só pelas próprias grandezas, mas também por suas derivadas (EQUAÇÃO. . . , 2019). Durante a modelagem de fenômenos físicos e químicos, por exemplo, é comum que se utilizem de conceitos como a conservação da energia ou a conservação da massa para obter um conjunto de equações diferenciais ordinárias e um conjunto de equações algébricas que descrevem o comportamento desses sistemas (GARCIA, 2005). Estes dois conjuntos caracterizam um sistema de equações diferenciais algébricas, ou sistema DAE, do inglês *Differential-Algebraic Equations*. Muitas vezes, como etapa seguinte do processo de modelagem, todo o conjunto de equações é manipulado matematicamente de modo a formar um sistema unicamente composto por equações diferencias ordinárias, ou sistema ODE, do inglês *Ordinary Differential Equations*.

Sistemas ODE podem ser facilmente convertidos em sistemas DAE, porém, o caso inverso nem sempre é fácil, ou mesmo possível. Devido a isto, pode-se dizer que todo o conjunto de sistemas ODE é apenas um subconjunto de um domínio ainda mais abrangente de sistemas, que são os sistemas DAE. O exemplo 2.1 apresenta um modelo ODE e um modelo DAE de um tanque para líquidos de formato qualquer.

$$\frac{dA}{dt}\ h\ +\ A\ \frac{dh}{dt} = \sum F_{in}\ -\ \sum F_{out} \qquad \begin{aligned} \frac{dV}{dt} &= \sum F_{in}\ -\ \sum F_{out} \\ V &= A.h \end{aligned} \tag{2.1}$$

Sistema ODE não-linear | Sistema DAE

No exemplo 2.1, h representa a altura da coluna de líquido no tanque, A representa a área da seção transversal do tanque na altura h, V representa o volume de líquido dentro do tanque, F_{in} é o fluxo volumétrico de entrada de líquido e F_{out} é o fluxo volumétrico de saída de líquido. É notável que a representação DAE é mais simplificada e legível que a representação ODE. Estas características ficam ainda mais evidentes a medida em que se aumenta a complexidade dos sistemas. Para ilustrar, supõe-se o exemplo 2.2 em que o tanque genérico do exemplo 2.1 passa a ser um tanque cônico equilátero.

$$\pi.h^2\ \frac{dh}{dt} = \sum F_{in}\ -\ \sum F_{out} \qquad \begin{aligned} \frac{dV}{dt} &= \sum F_{in}\ -\ \sum F_{out} \\ V &= A.h \\ A &= \frac{\pi.h^2}{3} \end{aligned} \tag{2.2}$$

Sistema ODE não-linear | Sistema DAE

Somente olhando para o sistema ODE do exemplo 2.2, é difícil dizer que este modelo se trata de um tanque cônico equilátero, já olhando para o sistema DAE, tal conclusão é facilitada. A lista 2.1 elenca algumas vantagens da modelagem em sistemas DAE.

2.1 – Vantagens dos modelos DAE quando comparados a modelos ODE (PELEGRINI, 2007)

- A modelagem é rápida;
- As equações são facilmente compreendidas;
- As variáveis possuem uma interpretação física;
- É fácil incluir ou remover fenômenos físicos ou geométricos dos modelos;
- É capaz de representar fenômenos que os modelos ODE não representam;
- A conversão para um modelo ODE pode comprometer a esparsidade do problema original.

Apesar da modelagem de sistemas dinâmicos ser razoavelmente facilitada ao se utilizar modelos DAE, a obtenção de uma solução consistente para tais modelos é geralmente mais complexa e desafiadora quando comparada ao caso ODE. Devido à linearidade da operação de derivação, a solução de qualquer sistema de equações de variáveis reais deve respeitar, não somente as equações do sistema, como também as derivadas destas equações. Para o caso de um sistema ODE, a derivação de suas equações jamais irão impor restrições adicionais ao problema (PELEGRINI, 2007), logo, os graus de liberdade de um sistema ODE são mantidos mesmo ao se efetuar derivações. Tal característica pode não ser válida para o caso de sistemas DAE. Quando alguma derivada de um sistema de equações passa a impor restrições sobre a solução do problema, diz-se que o sistema possui restrições escondidas. Utilizando a notação de Newton para a derivada, os exemplos 2.3 e 2.4 ilustram casos de restrição escondida.

$$\begin{array}{ll} & eq_1: \quad \dot{x_1} = x_2 \\ & eq_2: \quad x_1 = 3 \\ eq_1: \quad \dot{x_1} = x_2 & e\dot{q}_1: \quad \ddot{x_1} = \dot{x_2} = 0 \\ eq_2: \quad x_1 = 3 & e\dot{q}_2: \quad \dot{x_1} = 0 \\ \text{Sistema DAE original} & \text{Sistema com restrição revelada} \end{array} \tag{2.3}$$

$$\begin{array}{ll} & eq_1: \quad \dot{x_1} = x_2 \\ & eq_2: \quad \dot{x_2} = x_3 \\ & eq_3: \quad x_1 = 3 \\ eq_1: \quad \dot{x_1} = x_2 & e\dot{q}_2: \quad \ddot{x_2} = \dot{x_3} = 0 \\ eq_2: \quad \dot{x_2} = x_3 & e\dot{q}_3: \quad \dot{x_1} = 0 \\ eq_3: \quad x_1 = 3 & e\ddot{q}_3: \quad \ddot{x_1} = \dot{x_2} = 0 \\ \text{Sistema DAE original} & \text{Sistema com restrições reveladas} \end{array} \tag{2.4}$$

No exemplo 2.3, mesmo com o sistema original possuindo três variáveis e duas equações, vê-se que ele é totalmente definido, ou seja, ao contrário do que se imagina, o sistema não possui nenhum grau de liberdade com relação às suas variáveis. Isso se deve ao fato de que o sistema apresenta uma restrição escondida, ou seja, uma nova equação que pode ser acrescentada ao sistema sem que se acrescente novas variáveis. Esta restrição escondida surge ao se derivar a equação eq_2. É importante notar também que, apesar do sistema original possuir uma equação diferencial, não é possível inserir condições iniciais a ele, visto que a equação eq_2 já define toda a solução para x_1. Analisando agora o exemplo

2.4, vê-se que este é análogo ao exemplo anterior. Ele possui cinco variáveis e apenas três equações, porém, ainda sim, é bem determinado. Isso acontece devido ao surgimento de não uma, mas duas restrições escondidas: $e\dot{q}_3$ e $e\ddot{q}_3$. Este exemplo também não permitiria a adição de condições iniciais, mesmo possuindo duas equações diferenciais.

Uma propriedade muito importante que caracteriza os sistemas DAE é o seu índice diferencial, que nada mais é do que um número inteiro e positivo que serve para medir o quão distante um sistema de equações DAE está de se tornar um sistema de equações ODE. Nos exemplos 2.3 e 2.4, caso as equações que foram derivadas fossem retiradas do modelo, o sistema de equações remanescente seria um sistema ODE e as equações removidas poderiam ser utilizadas para se obter um conjunto de condições iniciais consistentes. Para ilustrar melhor, apresenta-se o exemplo 2.5, onde foram feitas algumas simplificações algébricas nas equações dos exemplos 2.3 e 2.4 para facilitar a visualização do subconjunto ODE e suas condições iniciais (abreviação C.I.).

$$
\begin{array}{ll}
ODE \begin{cases} e\dot{q}_2: & \dot{x_1} = 0 \\ e\dot{q}_1: & \dot{x_2} = 0 \end{cases} & ODE \begin{cases} eq_1: & \dot{x_1} = x_2 \\ e\ddot{q}_3: & \dot{x_2} = 0 \\ e\dot{q}_2: & \dot{x_3} = 0 \end{cases} \\
C.I. \begin{cases} eq_2: & x_1 = 3 \\ eq_1: & x_2 = 0 \end{cases} & C.I. \begin{cases} eq_3: & x_1 = 3 \\ e\dot{q}_3: & x_2 = 0 \\ eq_2: & x_3 = 0 \end{cases}
\end{array}
\tag{2.5}
$$

Exemplo 2.3 com restrição revelada — Exemplo 2.4 com restrições reveladas

O valor do índice diferencial de um sistema DAE é o número total de restrições escondidas que este possui. As restrições aparecem a medida em que se vai derivando as equações do sistema. Uma vez que todas as restrições são reveladas, o sistema resultante será um sistema ODE. Esta definição está de acordo com a definição de Brenan, Definição 2.1, e consiste numa forma intuitiva de se enxergar o índice diferencial.

Definição 2.1. (BRENAN et al., 1989) *O índice diferencial ν_d é o número mínimo de vezes que todo ou um subgrupo de equações de um sistema DAE $F(y, y', t) = 0$ precisa ser diferenciado, em relação à variável independente t, até ser transformado em um sistema ODE.*

É importante notar que o número de condições iniciais requeridas por um sistema DAE pode ser menor que a soma das ordens de todas as suas equações diferenciais, como é feito para o caso de sistemas ODE. Observa-se no exemplo 2.5 que, apesar dos sistemas apresentarem respectivamente duas e três equações diferenciais, estes também possuem duas e três condições iniciais naturais do sistema, sendo assim, nenhum desses sistemas

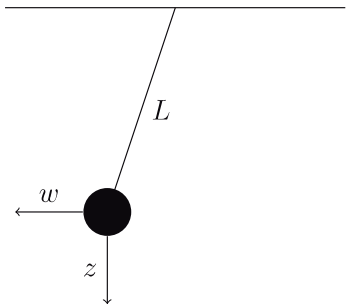
necessitam de condições iniciais adicionais. Nem sempre o número de condições iniciais naturais do sistema é suficiente para determiná-lo completamente, como acontece no exemplo 2.5. Na subseção 3.3.5, será estudado um sistema DAE que, ainda sim, necessita de condições iniciais adicionais, evidenciando o quão complexo pode ser a determinação de um conjunto de condições inicias consistente para um sistema DAE.

Voltando ao tema do índice diferencial, diz-se que sistemas DAE com índice maior ou igual a dois, $\nu_d \geq 2$, são sistemas de índice elevado, já os demais sistemas são tidos como de baixo índice (BRENAN et al., 1989). Isso inclui os sistemas DAE de índice zero, que na realidade são sistemas ODE. Os sistemas DAE são categorizados desta maneira para assim constituir duas classes de problemas com diferentes níveis de complexidade quanto a obtenção de uma solução numérica consistente. A literatura é, atualmente, rica em métodos de solução numérica de sistemas DAE de baixo índice, alcançando, inclusive, a mesma ordem de convergência obtida na solução de sistemas ODE. Alguns exemplos são DASSL (PETZOLD, 1982), LSODI (HINDMARSH, 1980), LIMEX (DEUFLHARD et al., 1987), RADAU5 (HAIRER; WANNER, 1999) e PSIDE (LIOEN et al., 1998). Já para o caso dos sistemas de índice elevado, encontram-se disponíveis na literatura métodos de integração capazes de resolver sistemas DAE de, no máximo, índice diferencial $\nu_d = 3$. Estes são os métodos MEBDF (ABDULLA et al., 2001), do inglês *Modified Extended Backward Differentiation Formulae*.

Embora seja possível se efetuar uma redução de índice por meio de derivações sucessivas em um sistema DAE, tal procedimento pode inviabilizar a obtenção de uma solução confiável (PELEGRINI, 2007). Isso acontece porque este processo de substituição de equações algébricas por equações diferenciais gera sistemas com graus de liberdade maiores que o sistemas original. Tais graus de liberdade deverão ser cobertos por condições iniciais, o que não acarretaria nenhum problema para uma solução analítica, porém, para o caso de uma solução numérica, geraria o problema de *drift-off*. Para exemplificar o problema, apresenta-se o modelo DAE da Figura 3, que representa um pêndulo de índice diferencial $\nu_d = 3$ em notação de Lagrange.

Figura 3 – Modelo DAE de um pêndulo de índice diferencial $\nu_d = 3$

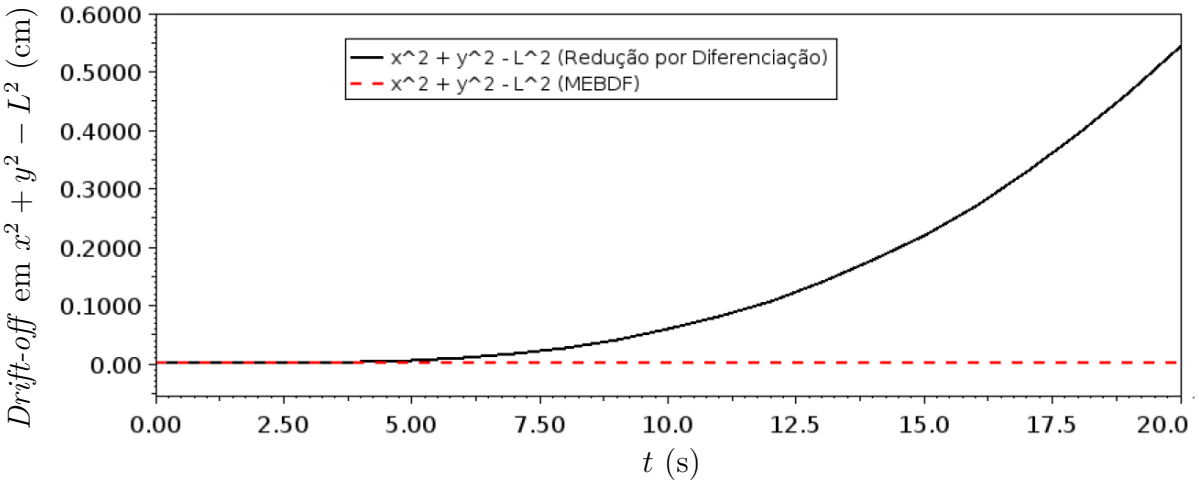
Modelo DAE:	Modelo ODE:
$eq_1: \quad x' = w$	$eq_1'': \quad x''' = w''$
$eq_2: \quad y' = z$	$eq_2'': \quad y''' = z''$
$eq_3: \quad w' = T.x$	$eq_3': \quad w'' = T'.x + T.x'$
$eq_4: \quad z' = T.y - g$	$eq_4': \quad z'' = T'.y + T.y'$
$eq_5: \quad x^2 + y^2 = L^2$	$eq_5''': \quad x'''.x + x''.x' + x''.x' + x'.x'' + y'''.y + y''.y' + y''.y' + y'.y'' = 0$



Fonte: (PELEGRINI, 2007)

Para expor a existência do problema de *drift-off*, é efetuada a simulação numérica do sistema da Figura 3 de duas maneiras diferentes: integração do sistema original utilizando o método MEBDF, adequado para sistemas de índice $\nu_d \leq 3$, e integração do sistema de índice reduzido utilizando o método DASSL, do inglês *Differential-Algebraic System Solver*, adequado para sistemas de índice $\nu_d \leq 1$. A Figura 4 exibe o problema de *drift-off* que surge ao se transformar a equação eq_5 em uma equação diferencial utilizando o processo de redução de índice.

Figura 4 – Distorção na solução numérica do pêndulo de índice diferencial 3 com redução de índice por diferenciação e utilizando o método MEBDF



Fonte: (PELEGRINI, 2007)

Como visto na Figura 4, a integração utilizando o modelo de índice reduzido

apresenta inconsistência quanto ao valor da expressão $x^2 + y^2 - L^2$, que deveria ser constante e igual a zero para respeitar a equação eq_5 do modelo original. Para este ensaio foi utilizada uma tolerância de 10^{-6} e, mesmo assim, o erro ocasionado pelo efeito de *drift-off* foi da ordem 1, o que é inadmissível. O simples fato da equação eq_5 não fazer parte do modelo de índice reduzido faz com que os valores tanto de x quanto de y precisem ser calculados através de integrações numéricas sucessivas, já que o modelo reduzido apresenta somente as derivadas de terceira ordem destas variáveis. Ao longo da simulação, os erros numéricos de integração vão se sobrepondo e gerando o erro visto na Figura 4.

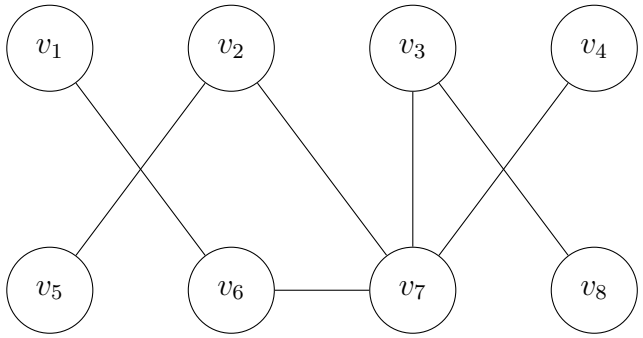
2.2.2 Teoria de grafos

A teoria de grafos é um ramo da matemática que estuda as relações entre objetos de um ou mais conjuntos. As definições, propriedades e técnicas desenvolvidas neste ramo possuem diversas aplicações em problemas práticos nas áreas de engenharia, logística, otimização, banco de dados, economia e diversas outras.

2.2.2.1 Definição básica

Os elementos chamados grafos são definidos como estruturas constituídas por um par de conjuntos, frequentemente chamados V e E, onde V representa o conjunto dos vértices (nós, pontos ou *vertex*) e E representa o conjunto das arestas (relações, linhas ou *edges*). Enquanto os elementos contidos no conjunto V devem respeitar uma determinada natureza, os elementos contidos no conjunto E respeitam a forma $e \in E$ tal que $e = [V]^2$, sendo assim, os elementos de E nada mais são do que pares de elementos de V representando uma associação. A notação matemática de um grafo se dá na forma $G(V, E)$ sendo G o símbolo que o representa. Encontra-se na Figura 5 uma representação gráfica de um grafo.

Figura 5 – Grafo $G(V, E)$ com V = $\{v_1...v_8\}$ e E = $\{\{v_1, v_6\}, \{v_2, v_5\}, \{v_2, v_7\}, \{v_3, v_7\}, \{v_3, v_8\}, \{v_4, v_7\}, \{v_6, v_7\}\}$.



No grafo da Figura 5 vê-se círculos que representam os vértices do conjunto V e linhas que representam as arestas do conjunto E. O número de elementos de V e de E são representados por $|V|$ e $|E|$, respectivamente. Outra definição importante dentro da teoria de grafos é a de caminho.

2.2.2.2 Caminhos

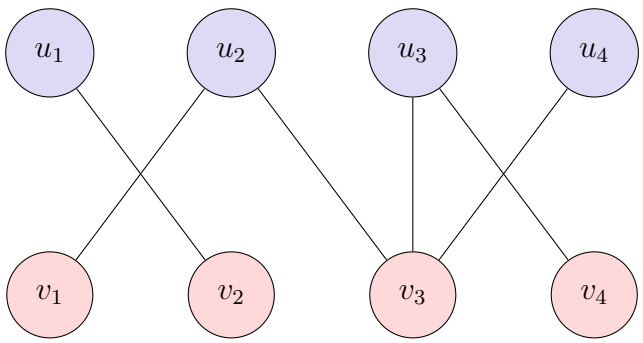
Um caminho P, segundo a teoria de grafos, é definido como sendo uma sequência de vértices iniciada em um vértice v_i e terminada em um vértice final v_f onde, para cada par sequencial de vértices deste caminho, existe uma aresta que os conecta. Em outras palavras, o caminho P representa uma maneira de se chegar ao vértice v_f partindo de um vértice v_i. Dentre as propriedades de um caminho P está seu comprimento, $|P|$, que nada mais é do que o número de arestas utilizadas para se realizar a conexão. Um grafo pode possuir infinitos caminhos, umas vez que não é vetada a repetição de vértices. Como exemplo temos na Figura 5 o caminho $P1_{v_5 \to v_7} = \{v_5 - v_2 - v_7\}$, de comprimento $|P1| = 2$, e $P2_{v_5 \to v_7} = \{v_5 - v_2 - v_5 - v_2 - v_7\}$, de comprimento $|P2| = 4$. Os caminhos pertinentes a este estudo são aqueles ditos hamiltonianos, onde não é permitida a repetição de vértices. Na Figura 5 temos três caminhos hamiltonianos distintos saindo do vértice v_1, são eles: $P_{v_1 \to v_4} = \{v_1 - v_6 - v_7 - v_4\}$, $P_{v_1 \to v_5} = \{v_1 - v_6 - v_7 - v_2 - v_5\}$ e $P_{v_1 \to v_8} = \{v_1 - v_6 - v_7 - v_3 - v_8\}$.

O conjunto de todos os grafos possíveis pode ser dividido em subconjuntos que compartilham determinadas características, recebendo assim uma classificação. Um grafo pode ser classificado como simples, completo, nulo, trivial, planar, entre outras. O subconjunto de grafos que para este trabalho possui especial importância são os chamados grafos bipartidos.

2.2.2.3 Grafos bipartidos

Os grafos bipartidos, ou bigrafos, são aqueles cujos vértices podem ser divididos em duas classes distintas resultando assim em dois conjunto de vértices: U e V. Além disso, no conjunto de arestas E, não podem existir arestas que unem vértices de uma mesma classe. A representação matemática de um grafo bipartido se da na forma $G(U, V, E)$ e sua representação gráfica ocorre conforme o exemplo apresentado na Figura 6.

Figura 6 – Grafo $G(U, V, E)$ com U = $\{u_1 ... u_4\}$, V = $\{v_1 ... v_4\}$ e E = $\{\{u_1, v_2\}, \{u_2, v_1\}, \{u_2, v_3\}, \{u_3, v_3\}, \{u_3, v_4\}, \{u_4, v_3\}\}$.



O grafo bipartido da Figura 6 é propositalmente semelhante ao grafo da Figura 5 para que se note, não só a distinção entre as classes de vértices, mas também a ausência da aresta $\{v_2, v_3\}$, correspondente à aresta $\{v_6, v_7\}$ da Figura 5, por violar a restrição de não unir vértices de mesma classe.

Os grafos bipartidos possuem aplicações voltadas a solução de problemas de associação de elementos de classes distintas, como por exemplo: trabalhadores e tarefas, tarefas e horários, objetos e características, dentre outras. As associações de maior interesse para este trabalho são as das equações e suas variáveis. Uma vez construído um grafo que represente tais associações, pode-se utilizar dos conhecimentos e métodos da teoria de grafos para encontrar e estudar propriedades de sistemas de equações afim de concluir sobre a sua solucionabilidade.

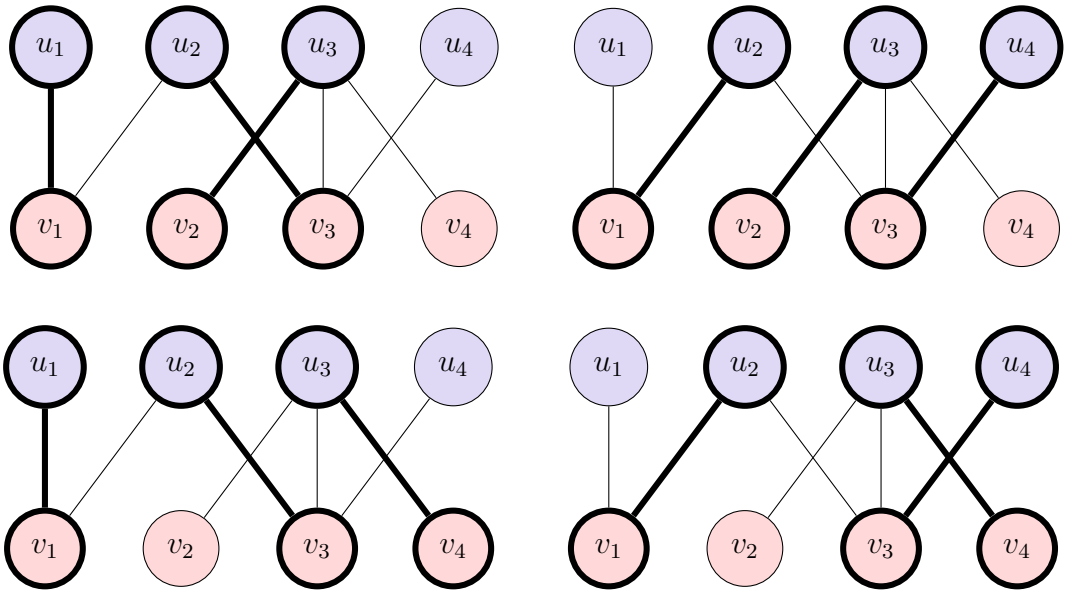
2.2.2.4 Acoplamentos

Dentro da teoria de grafos, existe uma propriedade de grafos bastante interessante chamada acoplamento ou emparelhamento. Um acoplamento M (do inglês *matching*) em um grafo $G(V, E)$ é definido como qualquer subconjunto do conjunto E cujas arestas não apresentam vértices em comum. Em outras palavras, se fossem listados os vértices que compõe cada aresta de um acoplamento M, certamente não haveriam vértices repetidos nesta lista.

O tamanho de um acoplamento M, também chamado de cardinalidade, possui notação na forma $|M|$ e é definida como sendo o número de arestas contidas em M. Em um grafo G podem haver inúmeros acoplamentos M com diferentes cardinalidades. Dentre eles, os tipos de acoplamentos que serão importantes para este trabalho são os chamados acoplamentos máximos e acoplamentos perfeitos.

Um acoplamento é dito máximo, M_{max}, quando este é o maior acoplamento possível de se formar em um grafo G, ou seja, um acoplamento de máxima cardinalidade. A quantidade de acoplamentos máximos possíveis em um grafo G pode não ser única. Na Figura 7 aparecem destacados todos os acoplamentos máximos possíveis em um exemplo de grafo bipartido $G(U, V, E)$.

Figura 7 – Grafo $G(U, V, E)$ com seus acoplamentos $M1_{max}$, $M2_{max}$, $M3_{max}$ e $M4_{max}$ destacados.

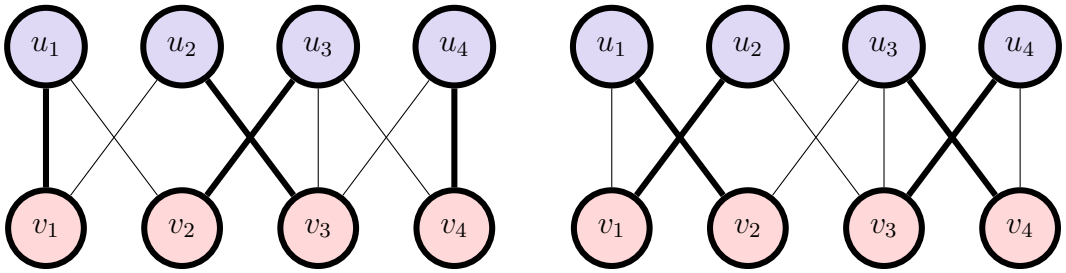


Vê-se na Figura 7 que o grafo G possui quatro possibilidades de acoplamentos máximos, sendo elas:

- $M1_{max} = \{\{u_1, v_1\}, \{u_2, v_3\}, \{u_3, v_2\}\}$
- $M2_{max} = \{\{u_2, v_1\}, \{u_3, v_2\}, \{u_4, v_3\}\}$
- $M3_{max} = \{\{u_1, v_1\}, \{u_2, v_3\}, \{u_3, v_4\}\}$
- $M4_{max} = \{\{u_2, v_1\}, \{u_3, v_4\}, \{u_4, v_3\}\}$

Pode-se dizer que o grafo $G(U, V, E)$ possui cardinalidade máxima $|M_{max}| = 3$. É interessante observar o que aconteceria com a propriedade de cardinalidade máxima caso fossem adicionadas ao grafo G as arestas $\{u_1, v_2\}$ e $\{u_4, v_4\}$. A Figura 8 apresenta os acoplamentos máximos possíveis para o novo grafo $G(U, V, E)$.

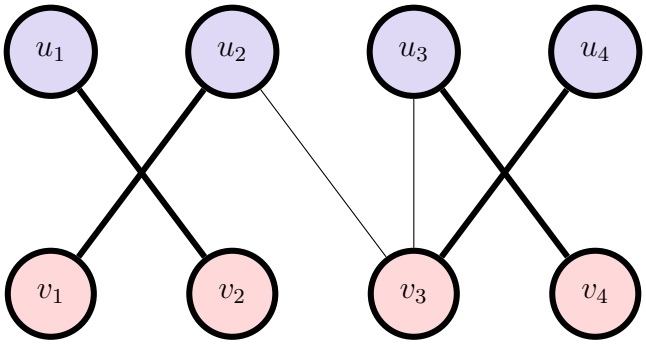
Figura 8 – Novo grafo $G(U, V, E)$ com seus acoplamentos $M1_{max}$ e $M2_{max}$ destacados.



Na Figura 8, observa-se que a adição das novas arestas fez com que a cardinalidade máxima do grafo G aumentasse para $|M_{max}| = 4$. O real intuito por trás deste exemplo não é de apenas falar sobre cardinalidade, mas também mostrar bons exemplos daquilo que se chama acoplamento perfeito ou completo.

Os acoplamentos $M1_{max}$ e $M2_{max}$ não são apenas máximos mas também perfeitos. Um acoplamento perfeito é definido como sendo o acoplamento M capaz de envolver todos os vértices de um grafo. É importante ressaltar que, assim como os acoplamentos máximos, os acoplamentos perfeitos também podem não ser únicos, tudo depende da estrutura do grafo. A exemplo disso, apresenta-se na Figura 9 o grafo da Figura 6 e seu acoplamento máximo que além de perfeito é também único.

Figura 9 – Grafo $G(U, V, E)$ com seu acoplamento perfeito M_{per}.



2.2.2.5 Acoplamentos e caminhos

Ao se analisar a natureza das arestas de um acoplamento, é fácil perceber que, por suas arestas não apresentarem vértices em comum, é impossível se formar caminhos de comprimento maior que 1 utilizando somente arestas de um acoplamento. Apesar disso, existem duas classes de caminhos que apresentam uma relação especial com acoplamentos, são eles os caminhos alternantes e os caminhos de aumento.

Os caminhos alternantes, ou alternado, são definidos em relação à um acoplamento M e são chamados desta maneira pois suas arestas alternam entre pertencentes ao acoplamento M e não pertencentes ao acoplamento M. Utilizando-se da representação gráfica de acoplamentos, é possível dizer que os caminhos alternantes são aqueles cuja sequência de arestas alterna entre aresta em negrito e aresta normal. Dentre os caminhos alternantes possíveis com relação a M_{per} no grafo da Figura 9 está, por exemplo, $P_{v_1 \to u_4} = \{v_1 - u_2 - v_3 - u_4\}$.

Aquilo que se define como caminho de aumento nada mais é do que um tipo particular de caminho alternante. Um caminho de aumento é todo caminho alternante com relação a um acoplamento M que se inicia e termina em vértices que não são tocados pelas arestas de M. O grafo da Figura 9 não apresenta nenhum caminho de aumento com relação a M_{per}. Isso se deve ao fato de não existir sequer um vértice do grafo que não seja tocado pelas arestas de M_{per}. Tal constatação é obvia, pois a própria definição de acoplamento perfeito exige que todos os vértices do grafo sejam alcançados. Diferente desta, uma constatação não tão óbvia é que o mesmo acontece com relação a qualquer acoplamento máximo. Apesar de os acoplamentos máximos não necessariamente cobrir todos os vértices de um grafo G, jamais será possível encontrar um caminho de aumento com relação à um acoplamento máximo. Tal fato, inclusive, é utilizado por Berge para definir acoplamentos máximos, Teorema 2.1.

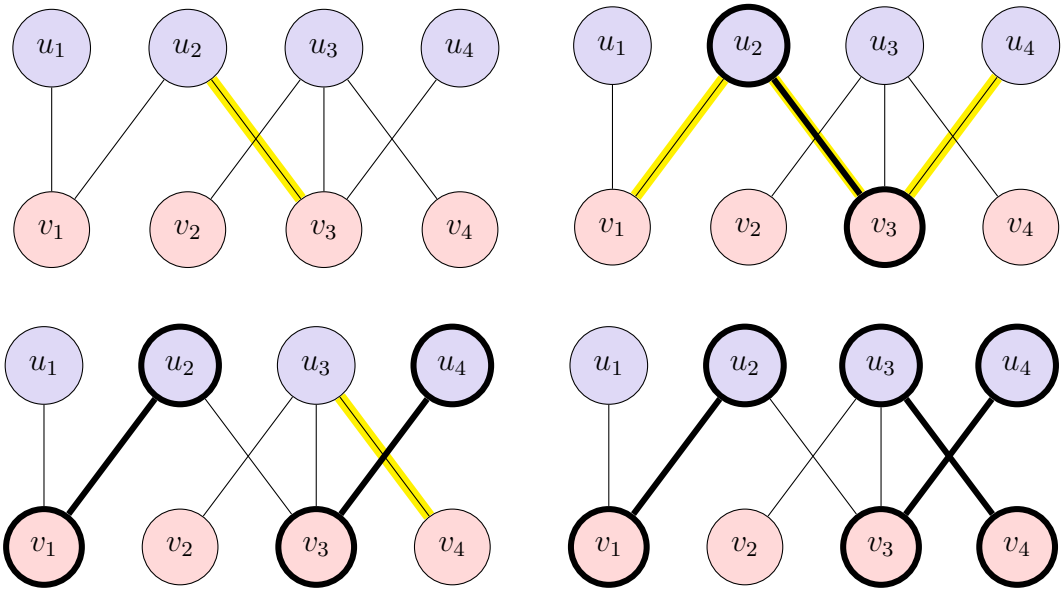
Teorema 2.1. (BERGE, 1957) Um acoplamento M em um grafo bipartido G é máximo se e somente se G não apresenta nenhum caminho de aumento em relação a M.

Existe um motivo interessante para os caminhos de aumento serem chamados desta forma. Basicamente, os caminhos de aumento em um acoplamento M possuem o poder de aumentar a cardinalidade deste acoplamento em uma unidade. Sendo assim, tais caminhos podem ser utilizados para se encontrar um acoplamento de máxima cardinalidade de um grafo G partindo de um acoplamento M qualquer. O procedimento para este fim seria o seguinte:

- **Passo 1:** Encontra-se um acoplamento M qualquer em G (pode até ser um acoplamento vazio, sem qualquer aresta)
- **Passo 2:** Encontra-se um caminho de aumento P com relação a M no grafo G
- **Passo 3:** Troca-se as arestas de M que foram utilizadas para a formação do caminho de aumento P pelas arestas utilizadas em P que não pertenciam ao acoplamento.
- **Passo 4:** Se for possível encontrar um caminho de aumento com relação ao novo acoplamento M, voltar ao Passo 2.

O Passo 3 deste procedimento é o verdadeiro responsável por aumentar a cardinalidade do acoplamento M. Ao realizá-lo, sempre será adicionado uma aresta a mais em M do que será removido dele, fazendo, assim, sua cardinalidade aumentar em uma unidade. Quando não for mais possível encontrar caminhos de aumento, o acoplamento M será máximo conforme dita o Teorema 2.1. Um exemplo deste procedimento encontra-se ilustrado na Figura 10.

Figura 10 – Grafo $G(U, V, E)$ com seus acoplamentos $M1_{max}$, $M2_{max}$, $M3_{max}$ e $M4_{max}$ destacados.



Cada imagem da Figura 10 representa a execução do Passo 2 do procedimento tendo os caminhos de aumento marcados na cor amarela. Todos os conhecimentos sobre teoria de grafos aqui apresentados serão utilizados como base para a seção 3.3.

2.2.3 Depuração de sistemas de sistemas de equações

É bem conhecido que simulações de processos utilizando simuladores baseados em equações podem apresentar diversos problemas de natureza estrutural (PELEGRINI, 2007). Para tais, existem técnicas de detecção, análise e ajuste que serão abordadas nas subseções 2.2.3.1, 2.2.3.2 e 2.2.3.3, respectivamente.

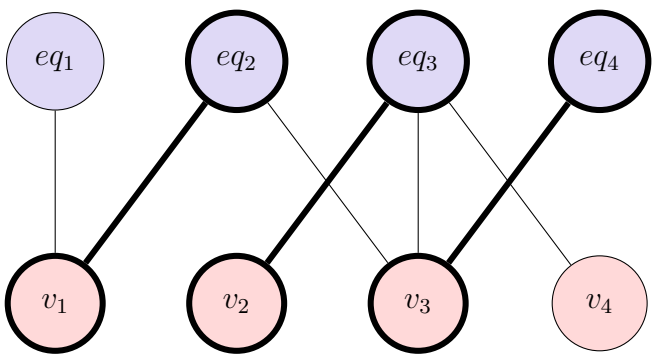
2.2.3.1 Detecção de singularidades estruturais

Em um sistema de equações, uma singularidade é dita estrutural quando tal singularidade diz respeito a forma do sistema ou como ele está posto. A sobredeterminação e subdeterminação em sistemas de equações são exemplos de singularidades estruturais e podem estar presentes até mesmo em sistemas com mesmo número de variáveis e de equações. Os motivos que levam um sistema à singularidade estrutural são:

- Diferença na quantidade de variáveis e equações;
- Existência de equações que não envolvem variáveis;
- Existência de variáveis que não aparecem em nenhuma equação;
- Existência de equações ou variáveis linearmente dependentes;
- Existência de um subsistema de equações sobredeterminado;
- Existência de um subsistema de equações subdeterminado.

Os conhecimentos apresentados na subseção 2.2.2 aplicados a sistemas de equações aparecem como um caminho muito eficiente de se encontrar suas singularidades estruturais. Vê-se como exemplo o grafo da Figura 11 onde encontram-se representadas as associações e um dos acoplamentos máximos de um sistema com quatro equações e quatro variáveis.

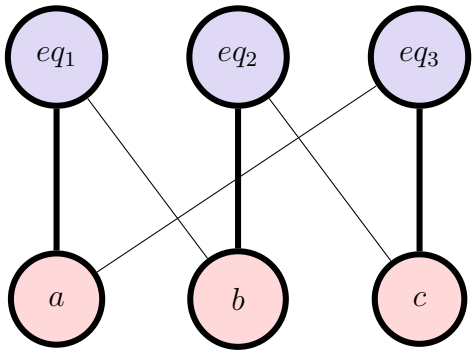
Figura 11 – Grafo $G(U, V, E)$ com um de seus acoplamentos máximos possíveis destacados.



Como é ilustrado na Figura 11, temos um acoplamento máximo de cardinalidade $|M_{max}| = 3$. O acoplamento máximo de um sistema de equações tem a característica de evidenciar a parte do sistema que é bem determinada. Os vértices não pertencentes ao acoplamento M_{max} evidenciam que o sistema de equações possui singularidade estrutural, sendo assim, ou sua solução numérica não é possível ou ela não é única. Vértices da classe equação não pertencentes ao acoplamento denunciam que o sistema possui sobredeterminação, enquanto vértices da classe variável apontam para subdeterminação. O sistema representado na Figura 11 possui ambas as singularidades devido aos vértices eq_1 e v_4. Caso um sistema de equações apresente um acoplamento perfeito, pode-se dizer que este não apresenta singularidade estrutural. Isso por si só não quer dizer que o sistema é solucionável, pois ele, ainda sim, pode apresentar singularidades numéricas. Um exemplo de sistema bem determinado com singularidade numérica encontra-se na Figura 12, onde a solução deste levaria a $0 = 3$.

Figura 12 – Sistema de equações e seu grafo com um de seus acoplamentos perfeitos possíveis destacados.

$$
\begin{aligned}
eq_1: \quad a &= b + 1 \\
eq_2: \quad b &= c + 1 \\
eq_3: \quad c &= a + 1
\end{aligned}
$$



2.2.3.2 Decomposição de sistemas de equações

Na intenção de expor as singularidades estruturais de um sistema de equações, apenas detectá-las não é o suficiente, é preciso evidenciar também os elementos do sistema que são os reais causadores da singularidades. A depuração de um sistema de equações seria muito mais interessante se, além de detectar a existência de singularidades, o depurador fosse capaz de dizer quais equações sobredeterminam quais variáveis e quais variáveis encontram-se subdeterminadas. Esta é a utilidade das técnicas de decomposição de sistemas de equações.

Dentro da teoria de grafos, existe uma técnica de decomposição para grafos bipartidos que, dado um acoplamento máximo M_{max} de um grafo $G(U, V, E)$, é capaz de particionar o grafo em três subgrafos de características específicas, $G^+(U^+, V^+, E^+)$, $G^w(U^w, V^w, E^w)$ e $G^-(U^-, V^-, E^-)$. Esta técnica é chamada Decomposição Dulmage-Mendelsohn e quando aplicada a sistemas de equações, os subgrafos revelados por ela

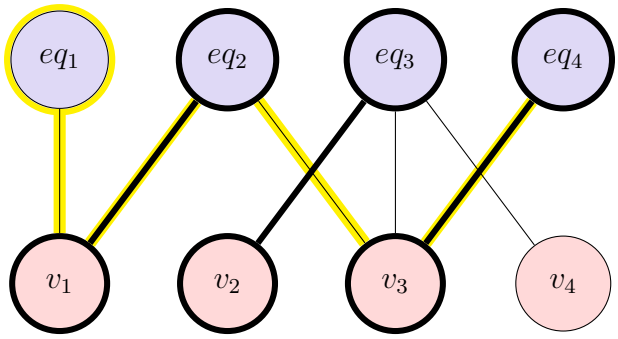
são, justamente, os grafos dos subsistemas sobredeterminados (G^+), bem determinado (G^w) e subdeterminados (G^-). Basicamente, dado um grafo de um sistema de equações $G(U, V, E)$ e um acoplamento M_{max}, a obtenção destes subsistemas acontece conforme explica a lista 2.2.

2.2 – Lista de subsistemas obtidos através da Decomposição Dulmage-Mendelsohn

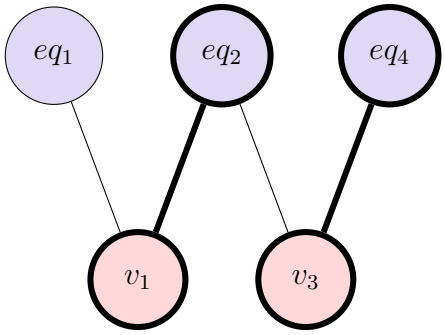
- Subsistemas sobredeterminados: subgrafos $G^+(U^+, V^+, E^+)$
  - ▷ Os elementos dos conjuntos U^+ e V^+ são todos os vértices tocados por caminhos alternantes em G com relação ao acoplamento M_{max} iniciados em vértices de equações v_{eq} não pertencentes ao acoplamento. Para cada vértice de equação v_{eq} haverá um subgrafo G^+;
  - ▷ Os elementos do conjunto E^+ são todas as arestas de G que unem os vértices contidos em U^+ e V^+.
- Subsistemas subdeterminados: subgrafos $G^-(U^-, V^-, E^-)$
  - ▷ Os elementos dos conjuntos U^- e V^- são todos os vértices tocados por caminhos alternantes em G com relação ao acoplamento M_{max} iniciados em vértices de variáveis v_{var} não pertencentes ao acoplamento. Para cada vértice de variável v_{var} haverá um subgrafo G^-;
  - ▷ Os elementos do conjunto E^- são todas as arestas de G que unem os vértices contidos em U^- e V^-.
- Subsistema bem determinado: subgrafo $G^w(U^w, V^w, E^w)$
  - ▷ Os elementos dos conjuntos U^w são todos os vértices de U não contidos nos conjuntos U^+ e U^- de todos os subgrafos G^+ e G^- do grafo G. Sua expressão matemática seria $U^w = U \setminus (U^+ \cup U^-)$ para todo U^+ e U^- em U;
  - ▷ Da mesma forma, os elementos de V^w são os vértices de V não contidos em V^+ e V^-. Sua expressão matemática seria: $V^w = V \setminus (V^+ \cup V^-)$ para todo V^+ e V^- em V;
  - ▷ Os elementos do conjunto E^w são todas as arestas de G que unem os vértices contidos em U^w e V^w.

Para exemplificar a obtenção destes subgrafos através da decomposição Dulmage-Mendelsohn apresenta-se a Figura 13.

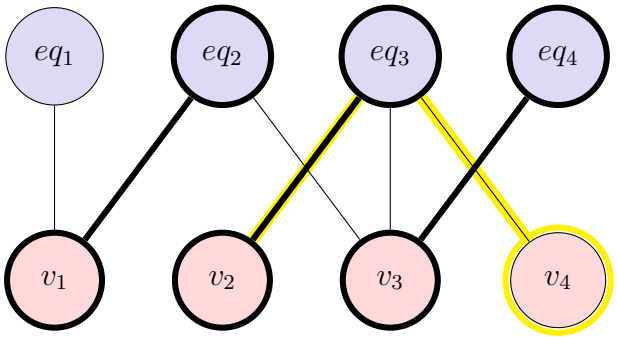
Figura 13 – Decomposição Dulmage-Mendelsohn aplicada a um grafo $G(U, V, E)$



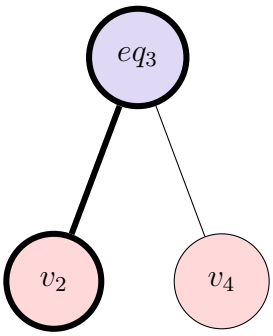
(a) Caminho alternante em G com relação a M_{max} e iniciado no vértice de equação eq_1 não pertencente a M_{max}



(b) Subgrafo $G^+(U^+, V^+, E^+)$ com um de seus acoplamentos M_{max} destacado



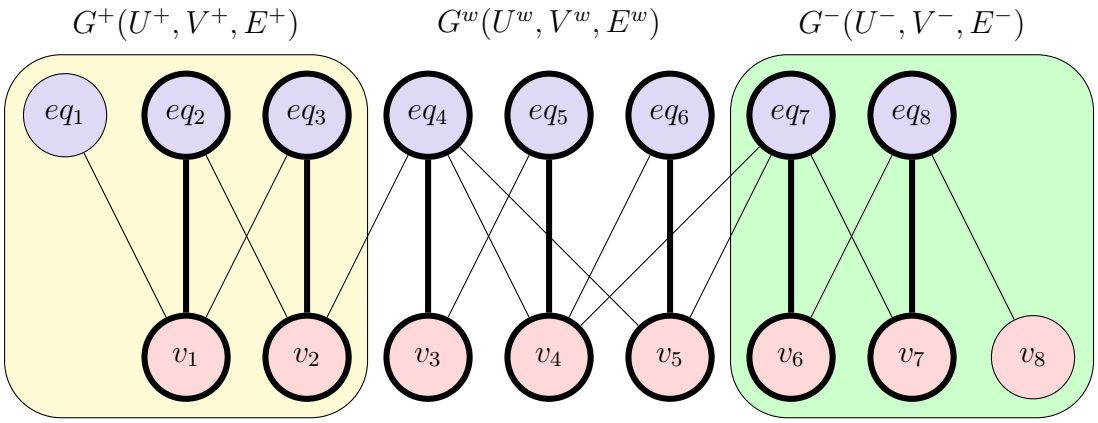
(c) Caminho alternante em G com relação a M_{max} e iniciado no vértice de variável v_4 não pertencente a M_{max}



(d) Subgrafo $G^-(U^-, V^-, E^-)$ com um de seus acoplamentos M_{max} destacado

Vê-se na Figura 13 que o sistema de equações representado, mesmo tendo igual número de variáveis e equações, apresenta tanto sobredeterminação quanto subdeterminação. O subgrafo G^+, que representa a parcela sobredeterminada do sistema, diz que as equações eq_1, eq_2 e eq_3 sobredeterminam as variáveis v_1 e v_3. Para corrigir esta singularidade, basta que se remova do sistema qualquer uma das equações citadas. Fazendo esta remoção e atualizando o grafo G, não haveriam mais vértices de equações não pertencentes ao acoplamento máximo M_{max}, sendo assim, não haveria mais sobredeterminação. Falando agora do subgrafo G^-, que representa a parcela subdeterminada do sistema, vê-se que as variáveis v_2 e v_4 estão subdeterminadas. Para resolver esta singularidade do sistema seria preciso ou remover uma das variáveis, o que muitas vezes não é possível, ou adicionar uma nova equação que utilize ao menos uma das variáveis subdeterminadas. Esta equação também não pode adicionar novas variáveis ao sistema, pois, caso assim fizesse, estas novas variáveis passariam a ser subdeterminadas. No exemplo apresentado na Figura 13, não há parcela bem determinada no sistema de equações. A Figura 14 ilustra um grafo de sistema de equações com os três subsistemas destacados.

Figura 14 – Grafo $G(U, V, E)$ com um de seus acoplamentos máximos possíveis destacados.



Na Figura 14, $G^w(U^w, V^w, E^w)$ representa o subsistema bem-determinado do sistema de equações. É interessante perceber que nenhuma equação de G^w depende de variáveis de G^-. Este é o fator que separa a parte bem determinada da subdeterminada, já que impossibilita os caminhos alternados de G^- alcançar os vértices de G^w. Por outro lado, as equações de G^w podem depender de variáveis de G^+, como mostra a relação $\{eq_4, v_2\}$. Isso significa que, apesar de a parte bem determinada apresentar um acoplamento perfeito, o sistema de equações que ela representa não necessariamente é bem posto.

2.2.3.3 Algoritmos para redução de índice de sistemas DAE

A técnica de redução de índice apresentada nesta subseção foi proposta por Pelegrini (2007) em sua tese de doutorado. A técnica de Pelegrini é uma modificação de um algoritmo bastante difundido para a redução de índice de sistemas DAE, proposto por Pantelides (1988).

Como visto na subseção 2.2.1, o índice diferencial é uma propriedade de sistemas DAE que está ligada à sua dificuldade de solução. O interesse por trás de uma redução de índice é, geralmente, possibilitar que sistemas de índice elevado possam ser simulados. Existem muitos algoritmos na literatura voltados à redução de índice, porém, a maioria deles aborda o problema por meio de uma análise estrutural do sistema de equações DAE. Estes algoritmos estudam o sistema, não numericamente, mas sim olhando o modo como ele está montado, utilizando para isso uma representação matricial ou em grafos para as equações do sistema.

A análise estrutural de um sistema DAE introduz uma propriedade similar ao índice diferencial chamada índice estrutural. Esta propriedade é calculada conforme a Definição 2.2.

Definição 2.2. (PELEGRINI, 2007) *O índice estrutural ν_s é o número mínimo de vezes que todo ou um subgrupo de equações de um sistema DAE $F(y, y', t) = 0$ precisa ser diferenciado, em relação à variável independente t, até que $F_{y'}$ seja estruturalmente não-singular.*

Singularidades estruturais são abordadas na subseção 2.2.3.1. Na Definição 2.2, o grupo de equações $F_{y'}$ se trata da derivada parcial de todas as equações do sistema F em relação as variáveis diferenciadas y'. O índice estrutural aparece para os algoritmos de redução de índice como um substituto palpável ao índice diferencial, porém, é bem conhecido na literatura que o índice estrutural ν_s pode, por vezes, ser diferente do índice diferencial ν_d de problemas DAE.

Até meados do ano 2000, pensava-se que o valor do índice estrutural seria limitado pela relação $\nu_s \leq \nu_d$, muito devido ao trabalho de Unger et al. (1995), porém, com o trabalho de Reißig et al. (2000), percebeu-se a existência de uma certa categoria de sistemas cujo índice estrutural supera o índice diferencial. Embora o trabalho de Reißig sugira que a solução numérica de sistemas com $\nu_s \geq \nu_d$ possa ser ainda mais difícil do que sugere seu índice ν_d, Pelegrini (2007) verificou em sua tese de doutorado que, ao menos para sistema exemplo utilizado por Reißig, tal condição não gera impacto algum ao processo de integração numérica.

O algoritmo de redução de índice de sistemas DAE proposto por Pelegrini em sua tese de doutorado é capaz de reduzir o índice estrutural de um sistema DAE até $\nu_s = 0$. A execução deste algoritmo revela informações preciosas sobre um sistema DAE, possibilitando sua solução numérica por meio de códigos clássicos de integração. Tal algoritmo é dividido em duas funções, uma auxiliar chamada *augmentMatching*, Algoritmo 1, e outra principal chamada *indexReducer*, Algoritmo 2. Basicamente, a função principal é aquela responsável por efetuar as reduções de índice e a função auxiliar é apenas uma função que busca aumentar a cardinalidade de um acoplamento M em um grafo G, semelhante ao apresentado na subseção 2.2.2.5. A primeira função a ser apresentada será a função auxiliar, *augmentMatching*.

Algoritmo 1: Pseudocódigo do algoritmo proposto por Pelegrini para aumentar a cardinalidade de um acoplamento M de um grafo G

Entradas: ***G(U,V,E)***: grafo bipartido do sistema DAE sendo U o conjunto de equações, V o conjunto de variáveis e E o conjunto de relações entre vértices de U e V. ***M***: um acoplamento do grafo G. ***v_{eq}***: vértice de equação analisado, sendo que $v_{eq} \in U$. ***coloridas***: conjunto de vértices de equação já analisados durante a execução do algoritmo. ***alg***: parâmetro que determina se o caminho de aumento pode terminar em um vértice de variável algébrica.

Saída: Retorna verdadeiro caso se consiga aumentar a cardinalidade do acoplamento M, caso contrário, retorna falso.

```
1  Função augmentMatching(G(U,V,E), M, v_eq, coloridas, alg):
2      coloridas ← coloridas ∪ v_eq
3      para cada variável v_var da equação v_eq faça
4          elegível ← v_var é uma variável diferenciada ou alg é verdadeiro
5          se elegível for verdadeiro e v_var ∉ M então
6              M ← M ∪ {v_eq, v_var}
7              retorna verdadeiro
8          fim
9      fim
10     para cada variável v_var da equação v_eq faça
11         elegível ← v_var é um variável diferenciada ou alg é verdadeiro
12         v_eq2 ← equação associada a v_var em M
13         se elegível for verdadeiro e v_eq2 ∉ coloridas então
14             se augmentMatching(G(U,V,E), M, v_eq2, coloridas, alg) retornar
                verdadeiro então
15                 M ← M ∪ {v_eq, v_var}
16                 retorna verdadeiro
17             fim
18         fim
19     fim
20     retorna falso
21 fim
```

Analisando o Algoritmo 1, é visto que ele procura aumentar a cardinalidade de um acoplamento M de duas maneiras diferentes. A primeira delas, que acontece no laço da linha 3, é procurar por alguma variável elegível da equação v_eq que não pertença a M, ou seja, que ainda não esteja acoplada a nenhuma outra equação. Caso não se encontre nenhuma variável elegível e disponível para associação, é dado início à segunda maneira através do laço da linha 10.

A segunda maneira de se aumentar a cardinalidade é procurar por um caminho de aumento, ou seja, um caminho alternante no grafo G com relação ao acoplamento M que se inicie no vértice v_{eq} e termine em algum vértice de variável elegível e ainda não pertencente ao acoplamento M. Através do laço da linha 10, este caminho de aumento vai se formando no grafo G e crescendo até que se encontre um vértice de variável que satisfaça a linha 5, retornando verdadeiro na linha 7, ou até que não se encontre mais opções de rota e acabe retornando falso na linha 20. As opções de rota vão se acabando a medida em que os vértices v_{eq} são coloridos na linha 2, indicando que tal vértice já foi analisado durante a formação do caminho de aumento. É importante notar que o caminho de aumento só se forma e cresce devido ao fato do Algoritmo 1 ser recursivo, linha 14.

Um aspecto que não pode passar despercebido é o efeito que a elegibilidade produz nos resultados do algoritmo. Nas linhas 4 e 11 é definida a elegibilidade do vértice de variável v_{var}. A variável v_{var} sempre será elegível se esta for uma variável diferenciada. Caso não seja uma variável diferenciada, a elegibilidade dependerá do parâmetro alg. Quando falso, o parâmetro alg fará com que somente as variáveis diferenciadas sejam elegíveis, quando verdadeiro, qualquer variável passa a ser elegível. Este aspecto é importante para garantir os resultados desejados da função principal, *indexReducer*, reproduzida pelo Algoritmo 2.

Algoritmo 2: Pseudocódigo do algoritmo proposto por Pelegrini para reduzir o índice estrutural de sistemas DAE

Entrada: ***G(U,V,E)***: grafo bipartido do sistema DAE sendo U o conjunto de equações, V o conjunto de variáveis e E o conjunto de relações entre vértices de U e V.

Saída: Retorna verdadeiro caso se consiga obter uma redução de índice até $\nu_s = 0$, em caso de falha, retorna falso.

```
1  Função indexReducer(G(U, V, E)):
2      M ← ∅
3      para cada equação v_eq ∈ U faça
4          coloridas ← ∅
5          se augmentMatching(G(U, V, E), M, v_eq, coloridas, falso) retornar falso
               então
6              marcadas ← coloridas
7              coloridas ← ∅
8              se augmentMatching(G(U, V, E), M, v_eq, coloridas, verdadeiro) retornar
                   falso então
9                  retorna falso
10             fim
11             U' ← derivada de todas as equações do conjunto marcadas
12             V' ← variáveis de todas as equações do conjunto U'
13             U ← U ∪ U'
14             V ← V ∪ V'
15             E ← E ∪ relações entre os elementos de U' e V'
16         fim
17     fim
18     retorna verdadeiro
19 fim
```

Vê-se no Algoritmo 2 que o processo de redução de índice parte de um acoplamento $M = \emptyset$, linha 2. É esperado que, ao final da execução deste algoritmo, o acoplamento M corresponda a um dos acoplamentos máximos possíveis para o grafo G. Isso somente ocorrerá caso algoritmo não retorne precocemente na linha 9, o que indicaria que a equação v_{eq} faz parte de um subconjunto sobredeterminado de equações em G. Ao retornar verdadeiro na linha 17, o acoplamento M certamente será um acoplamento máximo do grafo G.

Seguindo para a linha 3, tem-se a iteração principal do Algoritmo 2. Esta avaliará todas as equações do grafo G pegando uma a uma do conjunto U. É importante perceber que o conjunto U pode receber novas equações ao longo das iterações, o que acontece por meio da linha 13, fazendo com que o número total de iterações seja, a priori, desconhecido. As linha 4 e 7 é responsável por esvaziar o conjunto de equações que foram, ou serão, coloridas pela execução da função *augmentMatching*, Algoritmo 1. Esta operação sempre acontece logo antes da função *augmentMatching* ser chamada, seja na linha 5 ou na linha 8. A diferença entre as duas chamadas da função *augmentMatching* está em seu último parâmetro, alg (ver Algoritmo 1), que define se a função enxergará a existência das variáveis algébricas do grafo G ou se apenas enxergará as variáveis diferenciadas. Sabendo disso, a execução de *augmentMatching* na linha 5 primeiro tentará encontrar uma maneira de aumentar a cardinalidade de M considerando apenas as associações com variáveis diferenciadas. Em caso de sucesso, o algoritmo seguirá para a próxima iteração voltando a linha 3, caso contrário, segue-se para a linha 6.

Uma vez na linha 6, todas as equações que foram coloridas pela a função *augment-Matching* durante sua tentativa falha de aumentar a cardinalidade do acoplamento M são guardadas no conjunto *marcadas*. Todas as equações do conjunto *marcadas* serão, mais tarde, derivadas na linha 11, promovendo assim o efeito de redução de índice. Antes que isso aconteça, o algoritmo tenta novamente aumentar a cardinalidade do acoplamento M, linha 8, porém agora considera-se a possibilidade de associação com variáveis algébricas. O que se sucede nas linhas de 11 a 14 é a derivação das equações marcadas e a atualização do grafo com as novas equações, variáveis e relações.

A operação de derivada que utilizada na linha 11 do Algoritmo 2 não é a operação matemática convencional, mas sim uma simplificação para análises estruturais de sistemas de equações DAE. Tal operação é chamada derivada estrutural e respeita a Definição 2.3.

Definição 2.3. *Utilizando a notação de Lagrange, a derivada estrutural de uma função $f(x, y, y', t)$ em relação à variável independente t é a função $f'(x', y', y'')$, onde f' depende unicamente das derivadas das variáveis de f.*

É evidente que a derivada estrutural se trata de uma simplificação da derivada convencional, uma vez que, dependendo a estrutura de uma função $f(x, y, y', t)$, sua derivada em relação a t pode envolver não somente as variáveis derivadas x', y' e y'', mas também as variáveis algébricas x, y, y' e t. Por exemplo, a derivada convencional da função $f(x, y, y', t) = x^2 + yy' + t^2$ é $f'(x, x', y, y', y'', t) = 2xx' + y'y' + yy'' + 2t$ e envolve mais variáveis do que apenas x', y' e y''. É importante esclarecer que o intuito por trás da derivada estrutural não é encontrar uma expressão simplificada para f', mas sim revelar o conjunto de variáveis de f' com potencial de aumentar os graus de liberdade dinâmico do sistema.

3 Detalhamento técnico da ferramenta

Em engenharia de software, é comum a prática de subdividir projetos muito grandes em partes menores de propósito bem definido. Quando há relações de trocas de dados entre as partes subdividas, pode-se considerá-las camadas de software. Cada camada é responsável por desempenhar papéis específicos dentro do contexto geral, sendo que a combinação ou sequenciamento de suas funções é o que caracteriza o funcionamento do software.

Neste trabalho, todo o projeto de software foi subdividido em quatro camadas distintas chamadas: camada matemática, camada de modelagem, camada de síntese de sistemas e camada de simulação numérica. As seções deste capítulo tratarão especificamente de cada uma destas camadas, explicando seu propósito como componente da ferramenta e suas relações com as demais camadas.

3.1 Camada matemática

A camada matemática é a camada que implementa todas as estruturas e métodos de matemática simbólica para a ferramenta de modelagem e simulação. Seu objetivo é estabelecer a forma como usuário lida com os elementos matemáticos e a forma como estas informações matemáticas são armazenadas e tratadas. Para isso, são definidas estruturas como variáveis, vetores, matrizes, operações, expressões, equações e sistemas de equações.

3.1.1 Natureza de elementos

Os elementos matemáticos desta camada podem ser de três naturezas distintas: contínuos, discretos ou booleanos. Desta forma, existirão variáveis contínuas, variáveis discretas e variáveis booleanas. Além de definir o tipo de dado que cada variável carrega, a natureza também determina as operações que podem ser feitas entre as variáveis e qual seria a natureza do resultado das operações.

3.1.2 Elementos básicos

Os elementos básicos da camada matemática são suas variáveis, vetores e matrizes. Cada um destes elementos também é, necessariamente, de uma das citadas naturezas. Desta forma, existem nove tipos de elementos, são eles: variável contínua, variável discreta, variável booleana, vetor contínuo, vetor discreto, vetor booleano, matriz contínua, matriz discreta e matriz booleana. O Algoritmo 3 apresenta a forma como se instanciam cada um destes elementos na ferramenta de modelagem e simulação.

Algoritmo 3: Exemplo de instanciação de elementos matemáticos na ferramenta

```
# Basic Elements
x  = continuousVariable("x")
y  = continuousVariable("y")
z  = continuousVariable("z")
u  = continuousVariable("u", limits=(-12, 12))
d1 = discreteVariable("d1", dt = 1)
d2 = discreteVariable("d2", dt = 0.5, limits=(0, None))
d3 = discreteVariable("d3", dt = 3)
b1 = booleanVariable("b1")
b2 = booleanVariable("b2")
b3 = booleanVariable("b3")
X  = continuousVector("X", 2, limits=([-10, u], [x, 100]))
D1 = discreteVector("D1", 3, dt = 0.2)
Z1 = booleanVector("Z", 2)
B  = Vector([5, 7], "B")
D2 = Vector([d3, 4, d2], "D2")
Z2 = Vector([b1, b3], "Z3")
M  = continuousMatrix("M", (2, 2))
N  = discreteMatrix("N", (1, 3), dt = 12.4, limits=([(-3,-2,-1)], [(7,8,9)]))
O  = booleanMatrix("O", (2, 2))
I  = Matrix([(1, 0), (0, 1)], "I")
C  = Matrix([(5), (6)], "C")
R  = Matrix((1, d1[k], d3[k-2]), "R")
A  = Matrix([(x, y), (3, z)], "A")
```

Linguagem de programação: *python 3.7*

Todos os elementos básicos da camada matemática foram implementados segundo o paradigma da orientação a objetos, sendo assim, todos esses elementos são objetos com atributos e métodos próprios. Um atributo que é comum a todos eles é o atributo *name* que corresponde ao símbolo matemático que os nomeia. Os objetos dos elementos de natureza contínua e discreta possuem também o atributo *limits* que restringem o domínio das variáveis estabelecendo um valor máximo e um valor mínimo alcançável. Tal atributo não cabe aos objetos booleanos, pois estes, por definição, possuem apenas dois valores possíveis: *true* e *false*, com seus equivalentes quantitativos 1 e 0, respectivamente. Um atributo que é exclusivo dos elementos de natureza discreta e absolutamente necessário durante a instanciação destes é o período de tempo *dt* equivalente ao intervalo de tempo entre os instantes discretos k e $k-1$.

Uma matriz de uma determinada natureza nada mais é do que uma sequência estruturada e indexada de variáveis desta mesma natureza. Sua definição acontece conforme dita seu conceito matemático, possuindo m linhas e n colunas. Os índices de suas variáveis são definidos conforme suas posições nestas linhas e colunas. Vetores, dentro da camada matemática, são definidos como matrizes coluna, ou seja, matrizes com m linhas porém com uma única coluna. Devido a esta particularidade, a indexação de suas variáveis componentes é feita com apenas um índice: a posição da variável dentro da coluna. As definições de nomes e limites também valem para os objetos matriz e vetor.

Um aspecto importante com respeito aos objetos discretos é que estes não guardam apenas um valor mas todo um histórico de valores já assumidos pela variável discreta. A ideia de guardar informações com respeito aos valores passados é permitir que se escrevam equações a diferenças. Ao armazenar e sequenciar o histórico de valores assumidos, uma variável discreta pode ser vista como um vetor de valores. O tamanho deste vetor dependerá do acesso ao valor mais atrasado da variável discreta, por exemplo, caso o valor mais atrasado acessado esteja a cinco instantes no passado, o tamanho do vetor será seis já que, além dos cinco instantes passados, a variável discreta também precisa armazenar seu valor atual. Estendendo esta ideia aos vetores discretos, percebe-se que estes, na realidade, armazenam uma matriz de valores, uma vez que cada uma das variáveis discretas que o compõe são também vetores. Da mesma forma, uma matriz discreta armazena um bloco de valores.

Além dos elementos variáveis, existem também os elementos constantes e os elementos indefinidos. Os elementos constantes representam valores que não variam ao longo do tempo sendo categorizados em duas classes distintas: as constantes contínuas e as constantes booleanas. Já os elementos indefinidos são elementos que não possuem nem natureza nem dimensão. Estes são elementos utilizados internamente na ferramenta para representar algumas grandezas cuja natureza e dimensão ainda não foram definidas. Não é esperado que um usuário comum tenha qualquer contato com este tipo de elemento, seu real intuito é voltado à auxiliar os desenvolvedores da ferramenta de modelagem e simulação quando se faz necessário prover ao usuário um objeto que represente dados ainda indisponíveis sobre o processo ou sistema.

3.1.3 Operações matemáticas

As operações são a forma de relacionar matematicamente os elementos básicos. Estas possuem restrições quanto à natureza de seu domínio e apresenta uma natureza para sua imagem. O resultado de uma operação é geralmente um objeto do tipo expressão, porém, dependendo do caso, o resultado pode ser uma constante ou até mesmo uma das variáveis de seu domínio.

3.1.3.1 Operações nativas

As operações nativas são aquelas que constituem o corpo original de operações da ferramenta de modelagem e simulação. Elas são utilizadas para relacionar os elementos básicos da camada matemática criando, assim, expressões (ver subseção 3.1.4). A lista 3.1 apresenta todas as operações nativas da ferramenta.

3.1 – Lista de operações nativas da ferramenta de modelagem e simulação

- Soma: $x + y$
- Subtração: $x - y$
- Multiplicação: $x * y$
- Divisão: x / y
- Divisão inteira: x // y
- Resto da divisão: x % y
- Exponenciação: x ** y
- Radiciação: $root(x, y)$
- Logaritmação: $log(x, y)$
- Função gamma: $gamma(x)$
- Valor absoluto: $abs(x)$
- Valor oposto: - x
- Teste de igualdade: $x == y$
- Teste de desigualdade: x != y
- Teste de maior: $x > y$
- Teste de menor: $x < y$
- Teste de maior ou igual: $x >= y$
- Teste de menor ou igual: $x <= y$
- Lógica "ou" : $x \mid y$, ou, $x + y$
- Lógica "e" : x & y, ou, x * y
- Lógica "não" : $\sim x$
- Lógica "ou exclusivo" : $x \wedge y$
- Seno: $sin(x)$
- Cosseno: $cos(x)$
- Tangente: $tan(x)$
- Arco seno: $asin(x)$
- Arco cosseno: $acos(x)$
- Arco tangente: $atan(x, y)$
- Arco tangente restrito: $atan2(x, y)$
- Seno hiperbólico: $sinh(x)$
- Cosseno hiperbólico: $cosh(x)$
- Tangente hiperbólica: $tanh(x)$
- Arco seno hiperbólico: $asinh(x)$
- Arco cosseno hiperbólico: $acosh(x)$
- Arco tangente hiperbólica: $atanh(x)$
- Somatório: $summation(x, n)$
- Normalização: $normalize(x, n)$
- Transposição: $transpose(x)$
- Inversão: $inverse(x)$
- Derivação: $diff(x, limit)$[1]
- Atribuição: $new(x)$[2]
- Interpolação de ordem zero: $zoh(x)$
- Interpolação de ordem n: $noh(x)$
- Amostragem: $sample(x, st)$

[1]: Mais informações na subseção 3.1.7

[2]: Mais informações na subseção 3.1.5

Como mencionado na subseção 3.1.3.1, as operações possuem restrições quanto a natureza de seu domínio. Basicamente, as características dos parâmetros de entrada irão definir as características do resultado da operação. Por exemplo: a soma entre duas variáveis contínuas resultará em um número contínuo, ao passo que a soma de duas

grandezas discretas resultará em um valor discreto. Como exemplo, as tabelas 1, 2 e 3 apresentam as características dos resultados possíveis para a operação de multiplicação de acordo com as características dos parâmetros de entrada.

Tabela 1 – Definições de entrada e saída para o operador de multiplicação - Parte 1

$A * B$		**Constante B** (unidimensional)		**Grandeza B** (unidimensional)		
		Contínua	**Booleana**	**Contínua**	**Discreta**	**Booleana**
Constante A (unidimensional)	**Contínua**	Constante contínua	Constante contínua	Grandeza contínua[1]	Grandeza discreta[1]	Grandeza contínua[1]
	Booleana	Constante contínua	Constante booleana	Grandeza contínua[1]	Grandeza discreta[1]	Grandeza booleana[2]
Grandeza A (unidimensional)	**Contínua**	Grandeza contínua[1]	Grandeza contínua[1]	Grandeza contínua	Não se aplica	Grandeza contínua
	Discreta	Grandeza discreta[1]	Grandeza discreta[1]	Não se aplica	Grandeza discreta	Grandeza discreta
	Booleana	Grandeza contínua[1]	Grandeza booleana[2]	Grandeza contínua	Grandeza discreta	Grandeza booleana
Grandeza vetorial A (n x 1)	**Contínuo**	Grandeza vetorial contínua (n x 1)	Grandeza vetorial contínua (n x 1)	Grandeza matricial contínua (n x 1)	Não se aplica	Grandeza vetorial contínua (n x 1)
	Discreto	Grandeza vetorial discreta (n x 1)	Grandeza vetorial discreta (n x 1)	Não se aplica	Grandeza vetorial discreta (n x 1)	Grandeza vetorial discreta (n x 1)
	Booleano	Grandeza vetorial contínua (n x 1)	Grandeza vetorial booleana (n x 1)	Grandeza vetorial contínua (n x 1)	Grandeza vetorial discreta (n x 1)	Grandeza vetorial booleana (n x 1)
Grandeza matricial A (n x m)	**Contínua**	Grandeza matricial contínua (n x m)	Grandeza matricial contínua (n x m)	Grandeza matricial contínua (n x m)	Não se aplica	Grandeza matricial contínua (n x m)
	Discreta	Grandeza matricial discreta (n x m)	Grandeza matricial discreta (n x m)	Não se aplica	Grandeza matricial discreta (n x m)	Grandeza matricial discreta (n x m)
	Booleana	Grandeza matricial contínua (n x m)	Grandeza matricial booleana (n x m)	Grandeza matricial contínua (n x m)	Grandeza matricial discreta (n x m)	Grandeza matricial booleana (n x m)

[1]: O resultado pode ser uma Constante Contínua (zero) caso o operando constante for zero ou *false*.

[2]: O resultado pode ser uma Constante Booleana (*false*) caso o operando constante for *false*.

Tabela 2 – Definições de entrada e saída para o operador de multiplicação - Parte 2

$A * B$		**Grandeza vetorial B** (p x 1)		
		Contínuo	**Discreto**	**Booleano**
Constante A (unidimensional)	**Contínua**	Grandeza vetorial contínua (p x 1)	Grandeza vetorial discreta (p x 1)	Grandeza vetorial contínua (p x 1)
	Booleana	Grandeza vetorial contínua (p x 1)	Grandeza vetorial discreta (p x 1)	Grandeza vetorial booleana (p x 1)
Grandeza A (unidimensional)	**Contínua**	Grandeza vetorial contínua (p x 1)	Não se aplica	Grandeza vetorial contínua (p x 1)
	Discreta	Não se aplica	Grandeza vetorial discreta (p x 1)	Grandeza vetorial discreta (p x 1)
	Booleana	Grandeza vetorial contínua (p x 1)	Grandeza vetorial discreta (p x 1)	Grandeza vetorial booleana (p x 1)
Grandeza vetorial A (n x 1)	**Contínuo**	Não se aplica, já que não existem vetores (1 x 1)		
	Discreto			
	Booleano			
Grandeza matricial A (n x m)	**Contínua**	Se $m = p$, Grandeza vetorial contínua (n x 1)[1]	Não se aplica	Se $m = p$, Grandeza vetorial contínua (n x 1)[1]
	Discreta	Não se aplica	Se $m = p$, Grandeza vetorial discreta (n x 1)[1]	Se $m = p$, Grandeza vetorial discreta (n x 1)[1]
	Booleana	Se $m = p$, Grandeza vetorial contínua (n x 1)[1]	Se $m = p$, Grandeza vetorial discreta (n x 1)[1]	Se $m = p$, Grandeza vetorial booleana (n x 1)[1]

[1]: Caso $n = 1$, o resultado deixa de ser uma grandeza vetorial e passa a ser uma grandeza unidimensional.

Tabela 3 – Definições de entrada e saída para o operador de multiplicação - Parte 3

$A * B$		**Grandeza matricial B** (p x q)		
		Contínua	**Discreta**	**Booleana**
Constante A (unidimensional)	**Contínua**	Grandeza matricial contínua (p x q)	Grandeza matricial discreta (p x q)	Grandeza matricial contínua (p x q)
	Booleana	Grandeza matricial contínua (p x q)	Grandeza matricial discreta (p x q)	Grandeza matricial booleana (p x q)
Grandeza A (unidimensional)	**Contínua**	Grandeza matricial contínua (p x q)	Não se aplica	Grandeza matricial contínua (p x q)
	Discreta	Não se aplica	Grandeza matricial discreta (p x q)	Grandeza matricial discreta (p x q)
	Booleana	Grandeza matricial contínua (p x q)	Grandeza matricial discreta (p x q)	Grandeza matricial booleana (p x q)
Grandeza vetorial A (n x 1)	**Contínuo**	Se $p = 1$, Grandeza matricial contínua (n x q)	Não se aplica	Se $p = 1$, Grandeza matricial contínua (n x q)
	Discreto	Não se aplica	Se $p = 1$, Grandeza matricial discreta (n x q)	Se $p = 1$, Grandeza matricial discreta (n x q)
	Booleano	Se $p = 1$, Grandeza matricial contínua (n x q)	Se $p = 1$, Grandeza matricial discreta (n x q)	Se $p = 1$, Grandeza matricial booleana (n x q)
Grandeza matricial A (n x m)	**Contínua**	Se $m = p$, Grandeza matricial contínua (n x q)	Não se aplica	Se $m = p$, Grandeza matricial contínua (n x q)
	Discreta	Não se aplica	Se $m = p$, Grandeza matricial discreta (n x q)	Se $m = p$, Grandeza matricial discreta (n x q)
	Booleana	Se $m = p$, Grandeza matricial contínua (n x q)	Se $m = p$, Grandeza matricial discreta (n x q)	Se $m = p$, Grandeza matricial booleana (n x q)

Por motivos de viabilidade, as tabelas 1, 2 e 3 apresentam uma versão simplificada das definições de entrada e saída, já que além das constantes unidimensionais, existem também as contantes vetoriais e matriciais.

Analisando os dados das tabelas, uma observação importante a ser feita é que o resultado de uma operação se dá na forma de uma grandeza ou uma constante, assim como suas entradas. Isso significa que o resultado de uma operação pode ser utilizado como entrada para outras operações. Esta ideia é chave para a compreensão das estruturas chamadas expressões da subseção 3.1.4. Outro aspecto importante a se ressaltar é a possibilidade de operar sobre grandezas de naturezas diferentes, como por exemplo multiplicar uma grandeza booleana por uma contínua. As informações da tabelas apresentadas somente dizem respeito ao caso da operação de multiplicação, porém, existem conjuntos de definições e restrições para cada operação disponível na ferramenta.

3.1.3.2 Operações definidas pelo usuário

Além do conjunto de operações nativas, a ferramenta desenvolvida também oferece suporte para a definição de novas operações. Conforme a necessidade do usuário, este pode desenvolver suas próprias funções matemáticas em *python* e encapsulá-las como uma operação simbólica da ferramenta a fim de utilizá-las na modelagem matemática de fenômenos físicos. Para isso, basta o usuário definir um nome para a operação, a natureza dos parâmetros de entrada, a natureza da saída e a função propriamente dita, aquela que executa os cálculos. Um exemplo de como ficaria o código de uma nova operação encontra-se no Algoritmo 4.

Algoritmo 4: Exemplo de função definida pelo usuário - média ponderada

```
1  def weightedMean(*args):
2      # Domain and codomain: f[domain] = codomain
3      f = dict()
4      f[contVector(n), contVector(n)] = contVar
5      f[discVector(n), discVector(n)] = discVar
6      # Numerical function
7      def fun(vector, weightVector):
8          den = sum(weightVector)
9          num = 0
10         for i in range(len(vector)):
11             num += vector[i]*weightVector[i]
12         return num/den
13     return createFunction(mean, fun, args, f)
```

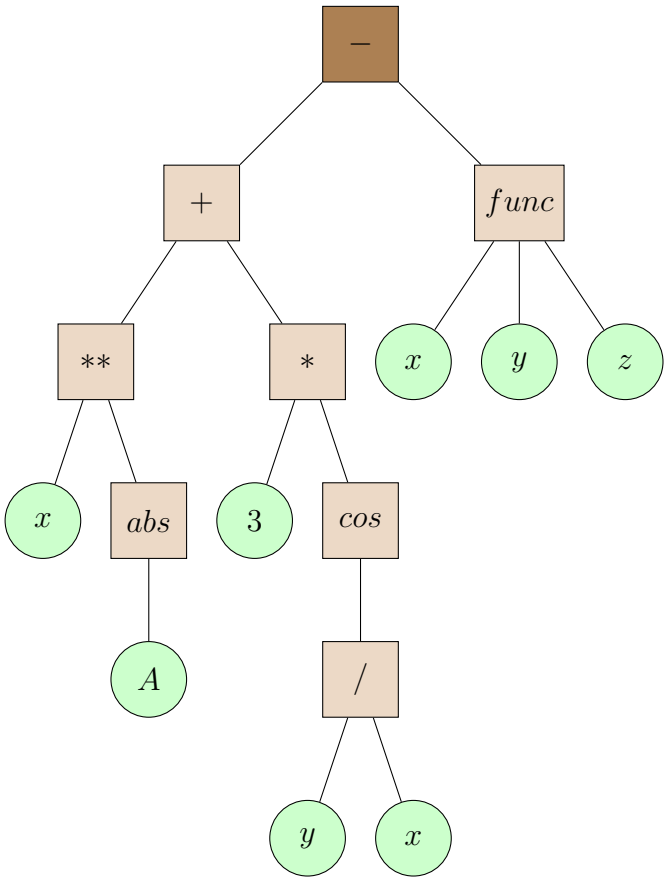
Linguagem de programação: *python 3.7*

Analisando o Algoritmo 4, na linha 1 é definido o operador *weightedMean* (média ponderada). Nas linhas de 3 a 5 são definidas as restrições de natureza dos parâmetros de entrada e a natureza da saída destas funções. A linha 4, por exemplo, diz que o operador *weightedMean* recebe duas grandezas vetoriais de tamanho n qualquer como parâmetros e seu resultado será uma grandeza unidimensional contínua. É importante salientar que definir duas grandezas multidimensionais com um mesmo tamanho n não só diz que as grandezas podem ser de qualquer tamanho como também diz que ambas as grandezas devem ser de mesma dimensão n. Assim como a linha 4, a linha 5 também adiciona a possibilidade do operador *weightedMean* receber como parâmetros duas grandezas vetoriais discretas e ter como resultado da operação uma grandeza unidimensional discreta. As linhas de 7 a 12 são as linhas que implementam o cálculo da média ponderada, recebendo duas listas como parâmetros e retornando, na linha 12, o valor numérico calculado. A função da linha 7 já pode supor que ambos os parâmetros serão listas de mesmo tamanho, uma vez que esta restrição já é imposta pelas linhas 4 e 5. Por fim, a linha 13 é a responsável por fazer com que a função *weightedMean* atue como se fosse um operador nativo da ferramenta. O resultado de uma operação, seja ela nativa ou não, é sempre um objeto do tipo expressão, apresentado na subseção 3.1.4.

3.1.4 Expressões matemáticas

As expressões surgem como resultado ao se efetuar operações envolvendo os elementos básicos da camada matemática. Os objetos do tipo expressão também são considerados grandezas, sejam elas unidimensionais, vetoriais ou matriciais. Assim como na matemática, uma expressão é um encadeamento organizado de operações e variáveis que geralmente resultam em algum valor de determinada dimensão quando são dados valores às suas variáveis, salvo em casos especiais como a indeterminação ou quando o resultado não existe. A implementação deste conceito matemático foi dado na forma de árvores n-árias, estruturas de dados abordadas na subseção 2.1.2. A Figura 15 apresenta como ficaria uma arvore n-ária para a expressão $x^{abs(A)} + 3.cos(y/x) - func(x, y, z)$.

Figura 15 – Árvore n-ária para a expressão $x^{abs(A)} + 3.cos(y/x) - func(x, y, z)$



Na Figura 15, os círculos na cor verde representam os elementos básicos da camada matemática, ou seja, variáveis, vetores, matrizes e constantes. Estes círculos são as folhas folhas da árvore n-ária. Já os retângulos em marrom são as operações da camada matemática e representam a origem de um ramo. Dentre os retângulos marrons encontra-se a raiz da árvore colorida em um tom mais escuro.

É importante salientar que a árvore n-ária da Figura 15 não é a única árvore capaz de representar a expressão $x^{abs(A)} + 3.cos(y/x) - func(x, y, z)$, já que a soma e a subtração podem ser efetuadas em qualquer ordem sem afetar o resultado. A árvore representada na Figura 15 é aquela resultante do caso em que a operação de soma é efetuada antes da operação de subtração, respeitando a ordem de notação das operações. Para auxiliar no entendimento, vale lembrar que as operações são efetuadas de baixo para cima, partindo das folhas da árvore e seguindo em direção a raiz.

Foi mencionado no início deste capítulo que as expressões são grandezas, assim como os elementos básicos da camada matemática, e por isso podem ser utilizadas em operações para compor expressões ainda maiores. Viu-se na subseção 3.1.3 que para funcionar corretamente, um operador precisa conhecer a natureza de suas entradas, sendo assim, uma expressão também precisa ter uma natureza definida. Sendo uma expressão estruturada na forma de árvore, a sua natureza é a natureza da resposta do operador que se encontra em sua raiz. No exemplo da Figura 15, a natureza da expressão representada é a natureza da resposta da operação de subtração, sua raiz. É claro que, como apresentado na subseção 3.1.3, a natureza de saída de uma operação depende da natureza de seus parâmetros, desta forma, a natureza de resposta da operação de subtração dependerá da natureza de resposta das operações soma e $func$. Este conceito se propaga até chegar as folhas da árvore que, por possuírem natureza bem definida, acabam determinando a natureza de resposta de todas as operações.

Não foram definidos quais elementos básicos representam cada símbolo utilizado na expressão $x^{abs(A)} + 3.cos(y/x) - func(x, y, z)$, porém, caso a definição fosse x, y e z representando variáveis contínuas e A representando uma matriz contínua, a resposta da operação de subtração (raiz da árvore) certamente seria uma grandeza contínua. Uma expressão que resulta em um valor deste tipo denominada expressão unidimensional contínua. Os tipos de expressões existentes são os que aparecem na lista 3.2.

3.2 – Lista dos tipos possíveis de expressões

- Expressão unidimensional contínua constante
- Expressão unidimensional booleana constante
- Expressão unidimensional contínua
- Expressão unidimensional booleana
- Expressão unidimensional discreta
- Expressão vetorial contínua constante
- Expressão vetorial booleana constante
- Expressão vetorial contínua
- Expressão vetorial booleana
- Expressão vetorial discreta
- Expressão matricial contínua constante
- Expressão matricial booleana constante
- Expressão matricial contínua
- Expressão matricial booleana
- Expressão matricial discreta

3.1.5 Equações

As equações são, possivelmente, os elementos matemáticos mais importantes da ferramenta de modelagem e simulação. É através delas que os fenômenos físicos acerca dos equipamentos são modelados, provendo assim, um meio para que tais equipamentos sejam simulados. Matematicamente falando, uma equação surge por meio da igualdade de duas grandezas as quais ao menos uma apresenta algum valor desconhecido (incógnita). Na camada matemática, estas grandezas podem ser elementos básicos ou expressões matemáticas, sendo assim, existem várias possibilidades de combinações de grandezas que podem resultar, ou não, em uma equação válida.

Além das equações, existe aquilo que foi denominado de atribuição e pode ser entendido como uma subclasse de equações. Estas são caracterizadas por se tratar de uma igualdade entre elementos básicos. Diferente do que ocorre para as demais equações, as atribuições servem exclusivamente para acrescentar informação aos elementos básicos. Supondo que exista uma variável a e uma variável b de determinada natureza, o resultado da atribuição $a = b$ é simplesmente a definição de que a e b não são mais variáveis

independentes, mas sim instâncias de uma nova variável unificada u_1 que carrega as características de ambas as variáveis. Para enriquecer o exemplo, supõe-se que existam também as variáveis c e d e que foi feita a atribuição $c = d$ gerando a variável unificada u_2. Ao se efetuar a atribuição $b = c$, por exemplo, será gerado uma nova variável unificada u_3 que unirá não apenas b e c, mas sim todas as variáveis a, b, c e d. Com o surgimento de u_3, as variáveis unificadas anteriores, u_1 e u_2, serão descartadas. Se posterior a isso for feita a atribuição $d = 2$, todas as mencionadas variáveis se tornaram constantes e iguais a 2.

Como mencionado, as variáveis unificadoras carregam as informações das variáveis às quais elas representam. Possíveis conflitos entre estas informações podem impedir que se efetue uma atribuição, como é o caso de se tentar igualar dois conjuntos de variáveis que são também constantes, porém de valores diferentes. Por exemplo: $a = 3$ e $b = 5$ impossibilita que seja feita a atribuição $a = b$. Outro conflito possível acontece ao se estabelecer limites de domínio diferentes para as variáveis que se deseja igualar, por exemplo $a < 0$ e $b > 0$.

Voltando a falar sobre a criação de equações, esta acontece por meio do operador *«equal»* e, caso sejam respeitadas as condições de igualdade, apresenta como resultado ou um objeto do tipo equação, ou uma atribuição. A tabelas 4, 5, 6, 7, 8 e 9 apresentam de igualdades possíveis de acordo com a natureza das grandezas igualadas.

Tabela 4 – Definições de entrada e saída para o operador de igualdade - Parte 1

$A << equal >> B$		**Constante B** (unidimensional)		**Variável B** (unidimensional)		
		Contínua	**Booleana**	**Contínua**	**Discreta**	**Booleana**
Constante A (unidimensional)	**Contínua**	Não se aplica	Não se aplica	Atribuição contínua	Não se aplica	Não se aplica
	Booleana	Não se aplica	Não se aplica	Não se aplica	Não se aplica	Atribuição booleana
Variável A (unidimensional)	**Contínua**	Atribuição contínua	Não se aplica	Atribuição contínua	Não se aplica	Não se aplica
	Discreta	Atribuição discreta	Não se aplica	Não se aplica	Atribuição discreta	Não se aplica
	Booleana	Não se aplica	Atribuição booleana	Não se aplica	Não se aplica	Atribuição booleana
Expressão A (unidimensional)	**Contínua**	Equação algébrica	Não se aplica	Equação algébrica	Não se aplica	Não se aplica
	Discreta	Equação a diferenças	Não se aplica	Não se aplica	Equação a diferenças	Não se aplica
	Booleana	Não se aplica	Equação booleana	Não se aplica	Não se aplica	Equação booleana

Tabela 5 – Definições de entrada e saída para o operador de igualdade - Parte 2

$A << equal >> B$		**Expressão B** (unidimensional)		
		Contínua	**Discreta**	**Booleana**
Constante A (unidimensional)	**Contínua**	Equação algébrica	Não se aplica	Não se aplica
	Booleana	Não se aplica	Não se aplica	Equação booleana
Variável A (unidimensional)	**Contínua**	Equação algébrica	Não se aplica	Não se aplica
	Discreta	Não se aplica	Equação a diferenças	Não se aplica
	Booleana	Não se aplica	Não se aplica	Equação booleana
Expressão A (unidimensional)	**Contínua**	Equação algébrica	Não se aplica	Não se aplica
	Discreta	Não se aplica	Equação a diferenças	Não se aplica
	Booleana	Não se aplica	Não se aplica	Equação booleana

Tabela 6 – Definições de entrada e saída para o operador de igualdade - Parte 3

A << *equal* >> B		**Constante vetorial B** (n x 1)		**Vetor B** (n x 1)		
		Contínua	**Booleana**	**Contínuo**	**Discreto**	**Booleano**
Constante vetorial A (n x 1)	**Contínua**	Não se aplica	Não se aplica	Atribuição vetorial contínua	Não se aplica	Não se aplica
	Booleana	Não se aplica	Não se aplica	Não se aplica	Não se aplica	Atribuição vetorial booleana
Vetor A (n x 1)	**Contínuo**	Atribuição vetorial contínua	Não se aplica	Atribuição vetorial contínua	Não se aplica	Não se aplica
	Discreto	Atribuição vetorial discreta	Não se aplica	Não se aplica	Atribuição vetorial discreta	Não se aplica
	Booleano	Não se aplica	Atribuição vetorial booleana	Não se aplica	Não se aplica	Atribuição vetorial booleana
Expressão vetorial A (n x 1)	**Contínua**	Equação algébrica vetorial	Não se aplica	Equação algébrica vetorial	Não se aplica	Não se aplica
	Discreta	Equação a diferenças vetorial	Não se aplica	Não se aplica	Equação a diferenças vetorial	Não se aplica
	Booleana	Não se aplica	Equação booleana vetorial	Não se aplica	Não se aplica	Equação booleana vetorial

Tabela 7 – Definições de entrada e saída para o operador de igualdade - Parte 4

$A << equal >> B$		**Expressão vetorial B** (n x 1)		
		Contínua	**Discreta**	**Booleana**
Constante vetorial A (n x 1)	**Contínua**	Equação algébrica vetorial	Não se aplica	Não se aplica
	Booleana	Não se aplica	Não se aplica	Equação booleana vetorial
Vetor A (n x 1)	**Contínua**	Equação algébrica vetorial	Não se aplica	Não se aplica
	Discreta	Não se aplica	Equação a diferenças vetorial	Não se aplica
	Booleana	Não se aplica	Não se aplica	Equação booleana vetorial
Expressão vetorial A (n x 1)	**Contínua**	Equação algébrica vetorial	Não se aplica	Não se aplica
	Discreta	Não se aplica	Equação a diferenças vetorial	Não se aplica
	Booleana	Não se aplica	Não se aplica	Equação booleana vetorial

Tabela 8 – Definições de entrada e saída para o operador de igualdade - Parte 5

A << *equal* >> B		**Constante matricial B** (n x m)		**Matriz B** (n x m)		
		Contínua	**Booleana**	**Contínuo**	**Discreto**	**Booleano**
Constante matricial A (n x m)	**Contínua**	Não se aplica	Não se aplica	Atribuição matricial contínua	Não se aplica	Não se aplica
	Booleana	Não se aplica	Não se aplica	Não se aplica	Não se aplica	Atribuição matricial booleana
Matriz A (n x m)	**Contínuo**	Atribuição matricial contínua	Não se aplica	Atribuição matricial contínua	Não se aplica	Não se aplica
	Discreto	Atribuição matricial discreta	Não se aplica	Não se aplica	Atribuição matricial discreta	Não se aplica
	Booleano	Não se aplica	Atribuição matricial booleana	Não se aplica	Não se aplica	Atribuição matricial booleana
Expressão matricial A (n x m)	**Contínua**	Equação algébrica matricial	Não se aplica	Equação algébrica matricial	Não se aplica	Não se aplica
	Discreta	Equação a diferenças matricial	Não se aplica	Não se aplica	Equação a diferenças matricial	Não se aplica
	Booleana	Não se aplica	Equação booleana matricial	Não se aplica	Não se aplica	Equação booleana matricial

Tabela 9 – Definições de entrada e saída para o operador de igualdade - Parte 6

A << *equal* >> B		**Expressão matricial B** (n x m)		
		Contínua	**Discreta**	**Booleana**
Constante matricial A (n x m)	**Contínua**	Equação algébrica matricial	Não se aplica	Não se aplica
	Booleana	Não se aplica	Não se aplica	Equação booleana matricial
Matriz A (n x m)	**Contínua**	Equação algébrica matricial	Não se aplica	Não se aplica
	Discreta	Não se aplica	Equação a diferenças matricial	Não se aplica
	Booleana	Não se aplica	Não se aplica	Equação booleana matricial
Expressão matricial A (n x 1)	**Contínua**	Equação algébrica matricial	Não se aplica	Não se aplica
	Discreta	Não se aplica	Equação a diferenças matricial	Não se aplica
	Booleana	Não se aplica	Não se aplica	Equação booleana vetorial

Além dos tipos de equações apresentados nas tabelas 4, 5, 6, 7, 8 e 9, existem tipos especiais de equações que nascem da utilização do operador *new*, apresentado pela primeira vez na lista 3.1 da subseção 3.1.3. Este operador é responsável por transformar uma equação em uma atribuição que funciona de modo um pouco particular. Ao se definir, por exemplo, $new(a) = a + 1$, o que acontece é que a variável a deixa de ser uma variável de fato para se tornar uma constante calculada a cada avanço temporal do processo de simulação. Para isso, é necessário que seja definido um valor inicial para a variável a. Explicando um pouco melhor, supõe-se que a possui condição inicial $a_0 = 0$. Durante o passo inicial de solução numérica, toda a equação que necessitar do valor de a receberá o valor $a = 0$. Já no passo seguinte, o novo valor de a passara a ser $a = 1$, pois o que faz a equação $new(a) = a + 1$ é estabelecer que o novo valor de a será o valor anterior de a somado um. Este processo ocorrerá a cada avanço temporal até que a simulação esteja terminada.

Diferente do exemplo teórico apresentado, existem aplicações realmente úteis para o operador *new*, especialmente na modelagem de sistemas com memória. Bons exemplos de sistemas com memória são as válvulas com duplo piloto. Uma válvula deste tipo possui a característica de abrir somente se lhe é enviado o comando para abrir, e fechar somente quando lhe é enviado o comando para fechar. Quando não é enviado comando algum, a válvula permanece da maneira em que está, seja aberta ou fechada. Supõe-se aqui que jamais são enviados os comandos para abrir e para fechar ao mesmo tempo, porém, caso aconteça, a válvula priorizará algum desses comandos. Válvulas com este comportamento podem ser facilmente modeladas utilizando o operador *new* da seguinte forma: $new(S) = O \mid (S \ \& \sim C)$ sendo S o estado da válvula, O o sinal de abertura e C o sinal de fechamento. Nesta equação, o sinal de abertura é priorizado quando é enviado o comando de abrir e de fechar ao mesmo tempo. A tabela verdade que representa a expressão dada é a Tabela 10.

Tabela 10 – Tabela verdade para a atribuição $new(S) = O \mid (S \ \& \sim C)$

S	O	C	$new(S)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

É importante ressaltar que o operador *new* se aplica somente aos elementos básicos do tipo contínuo e booleano. Ao se criar uma atribuição utilizando o operador *new*, faz-se necessário que este operador esteja isolado em um dos lados da igualdade, ou seja, $new(var) = f(var, ...)$ sendo impossível fazer, por exemplo, $3.new(var) = f(var, ...)$. Ao mesmo tempo, o outro lado da igualdade deve ser uma grandeza ou contínua ou booleana conforme a natureza do elemento ao qual se aplica o operador *new*.

Além das novidades trazidas pelo operador *new*, existem também as nuances advindas do uso do operador *diff* sob equações algébricas contínuas de qualquer dimensão. Serão apresentados na subseção 3.1.7 os aspectos relacionados ao operador *diff*, porém, cabe dizer aqui que tal operador é responsável por derivar equações de natureza contínua gerando, assim, um subtipo de equação algébrica chamada equação diferencial.

A criação de equações e atribuições na ferramenta de modelagem e simulação acontece de uma maneira simples e intuitiva. Esta consiste em apenas dois passos, sendo o primeiro a etapa de criação das variáveis que serão utilizadas para compor as equações e, por fim, a etapa de escrita das equações. Um exemplo deste processo aparece no Algoritmo 5.

Algoritmo 5: Exemplos de criação de equações

```
# Basic Elements
x  = continuousVariable("x")
y  = continuousVariable("y")
z  = continuousVariable("z")
u  = continuousVariable("u", limits=(-12, 12))
d1 = discreteVariable("d1", dt = 1)
d2 = discreteVariable("d2", dt = 0.5, limits=(0, None))
d3 = discreteVariable("d3", dt = 3)
b1 = booleanVariable("b1")
b2 = booleanVariable("b2")
b3 = booleanVariable("b3")
X  = continuousVector("X", 2, limits=([-10, u], [x, 100]))
D1 = discreteVector("D1", 3, dt = 0.2)
Z1 = booleanVector("Z", 2)
B  = Vector([5, 7], "B")
D2 = Vector([d3, 4, d2], "D2")
Z2 = Vector([b1, b3], "Z3")
M  = continuousMatrix("M", (2, 2))
N  = discreteMatrix("N", (1, 3), dt = 12.4, limits=([(-3,-2,-1)], [(7,8,9)]))
O  = booleanMatrix("O", (2, 2))
I  = Matrix([(1, 0), (0, 1)], "I")
C  = Matrix([(5), (6)], "C")
R  = Matrix((1, d1[k], d3[k-2]), "R")
A  = Matrix([(x, y), (3, z)], "A")

# Algebraic Equation
eq1 = x**2 + y <<equal>> sin(cos(x) + foh(d1[k-2], 3)*b1) - 2
# Differential Equation
eq2 = diff(x, dmin = y, dmax = y + 10) <<equal>> z*log(x*y) - zoh(d2, 1)
# Difference Equation
eq3 = d3[k]**2 <<equal>> d3[k-1]*2 - d3[k-2] + cos(d1[k-1])
# Boolean Equation
eq4 = b3 <<equal>> (b1 | (b2 & b3))
# Algebraic Vector Equation
eq5 = M*Z2 <<equal>> A*B + I*C
# Differential Vector Equation
eq6 = diff(X, dmin = [0, -3]) <<equal>> A*X + B*u
# Difference Vector Equation
eq7 = D1[k] <<equal>> (N[k]*transpose(N[k-1]))*D2[k-1] + [1, 2, 3]
# Boolean Vector Equation
eq8 = (Z2 ^ (X > [0, 0])) <<equal>> ((2*C == X) | (Z1 & Z2))
# Algebraic Matrix Equation
eq9 = A + I <<equal>> X*transpose(C) - M
# Differential Matrix Equation
eq10 = diff(A)*diff(M) <<equal>> C*[(x, zoh(d3, 1))] - I
# Difference Matrix Equation
eq11 = N[k] + [(5, 2.2, 8)] <<equal>> 3*R[k-3]
# Boolean Matrix Equation
eq12 = ((z*M != A) & O) <<equal>> [(False, b1), (b3, (b2 | b1))]
# NewBoolean Equation
eq13 = new(b1) <<equal>> (((x > y) ^ b2) & (d3[k] != d3[k-1]))
# NewContinuous Equation
eq14 = new(z) <<equal>> x + (z*b3)
```

Linguagem de programação: *python 3.7*

3.1.6 Sistemas de equações

As estruturas matemáticas denominadas sistemas de equações são, na realidade, implementadas na forma de uma única classe chamada *equationSystem*. Esta classe é uma generalização do que poderia ser uma sistema de equações DAE, de equações não-lineares, de equações a diferenças, entre outras. A importância desta classe está em agrupar as equações na forma de um conjunto e prover informações a respeito deste conjunto, como o número de variáveis do sistema, a quantidade de equações diferenciais, o atraso máximo de variáveis discretas e assim por diante. Também é responsável por estruturar e prover uma função de resíduos de equações, função essa que possibilita a simulação numérica de sistemas por meio de algoritmos numéricos, mais informações na seção 3.4.

3.1.7 Derivações

Uma das operações mais importantes do pacote de operações nativas da ferramenta de modelagem e simulação é a derivação. Apresentada pela primeira vez na lista 3.1 da subseção 3.1.3.1, a derivação pode ser aplicada às grandezas de natureza contínua em relação ao tempo, isto é, variáveis, vetores, matrizes, expressões e equações contínuas. Ao longo desta subseção serão apresentadas duas técnicas de derivação que serão bastante importantes para a ferramenta, são elas a derivação simbólica e a derivação simbólico-numérica. Ambas são performadas pelo operador *diff* que as escolhe dependendo do caso de derivação.

3.1.7.1 Derivação simbólica

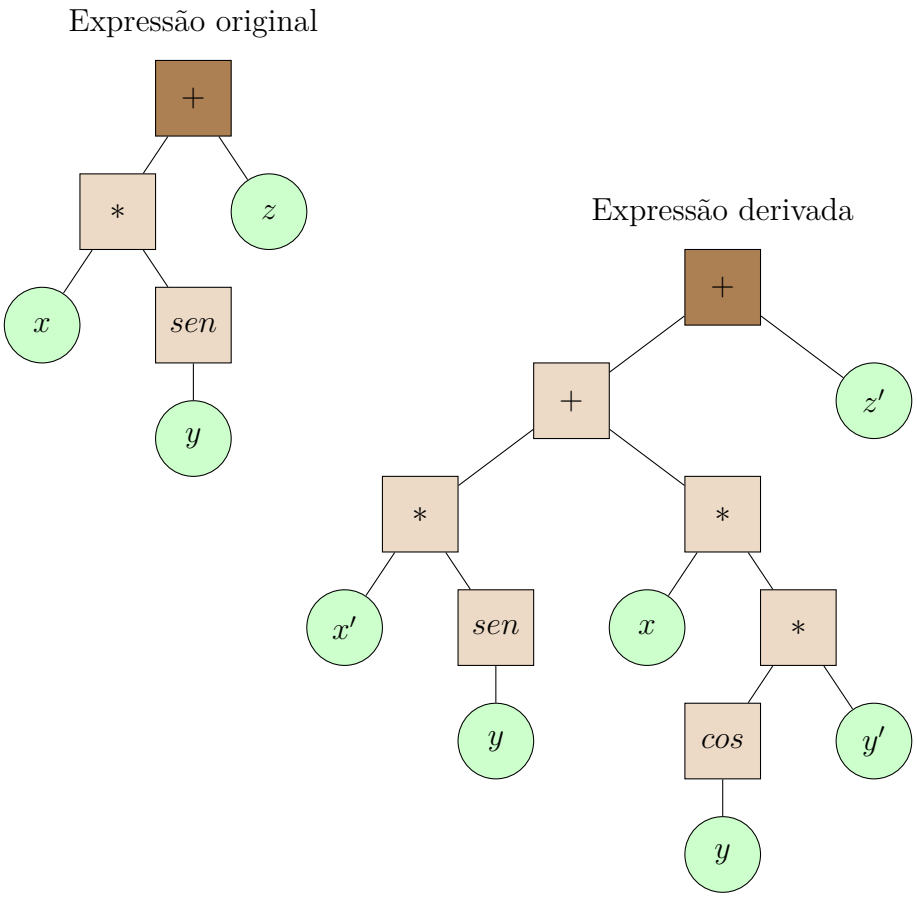
Dada uma grandeza contínua, a derivação simbólica é o procedimento na qual se deseja encontrar uma expressão simbólica que represente a derivada desta grandeza. Tal operação possui grande importância para ferramenta pois é necessária para a aplicação dos métodos apresentados na subseção 3.3.3. Seu uso mais comum se dará sobre objetos do tipo equação, podendo ser algébricas ou diferenciais e de qualquer dimensão. Como mencionado na subseção 3.1.5, as equações contínuas são responsáveis por igualar duas grandezas também contínuas sendo ao menos uma delas uma expressão. Sabendo que cada uma dessas grandezas representa um dos lados da equação, pode-se dizer que a derivada desta equação é definida como sendo a derivada de cada um de seus lados. Para entender como funciona a derivação desses lados, é preciso entender como funciona a derivação de uma expressão.

Para cada operação aplicável a grandezas contínuas é definida uma expressão para sua derivada com relação ao tempo. Tais operações partem da premissa de que seus parâmetros são, possivelmente, variáveis que dependem do tempo. Por exemplo, a função $f(x) = sen(x)$ é tratada como uma função composta em relação ao tempo já que existe a possibilidade de x ser variante no tempo, ou seja, $x = g(t)$ sendo g uma função ainda

desconhecida. Desta maneira, a derivada temporal de $f(x)$ em notação de Lagrange é $f'(x) = cos(x).x'$.

Viu-se na subseção 3.1.4 que os objetos do tipo expressão são implementados na forma de uma árvore n-ária, sendo as operações matemáticas a base de seus ramos, e os elementos básicos da camada como sendo as folhas. Esta estrutura de dados é especialmente conveniente para procedimentos de derivação simbólica, pois a implementação da derivação pode acontecer de modo recursivo, se originando na raiz da árvore e se propagando até as folhas. Para exemplificar a operação, toma-se como exemplo a expressão $x * sen(y) + z$ e sua derivada $x'.sen(y) + x.cos(y).y' + z'$ cujas árvores n-árias encontra-se ilustradas na Figura 16.

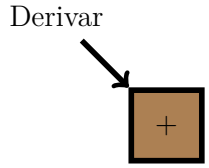
Figura 16 – Árvore n-ária da expressão $x.sen(y) + z$ e de sua derivada $x'.sen(y) + x.cos(y).y' + z'$



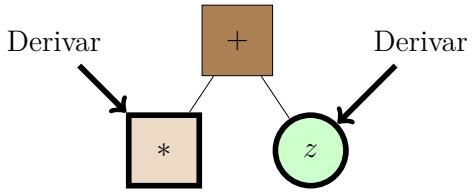
A Figura 17 mostra como funciona a propagação da operação de derivada, passo a passo, ao longo da árvore n-ária.

Figura 17 – Propagação da derivada simbólica na árvore n-ária da expressão $x*sen(y/5)+z$

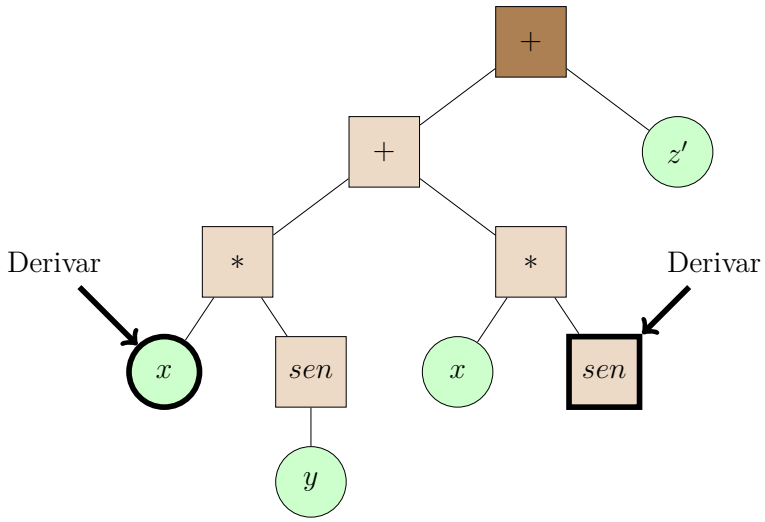
Passo 1: $f(a,b) = a + b \rightarrow f'(a,b) = a' + b'$



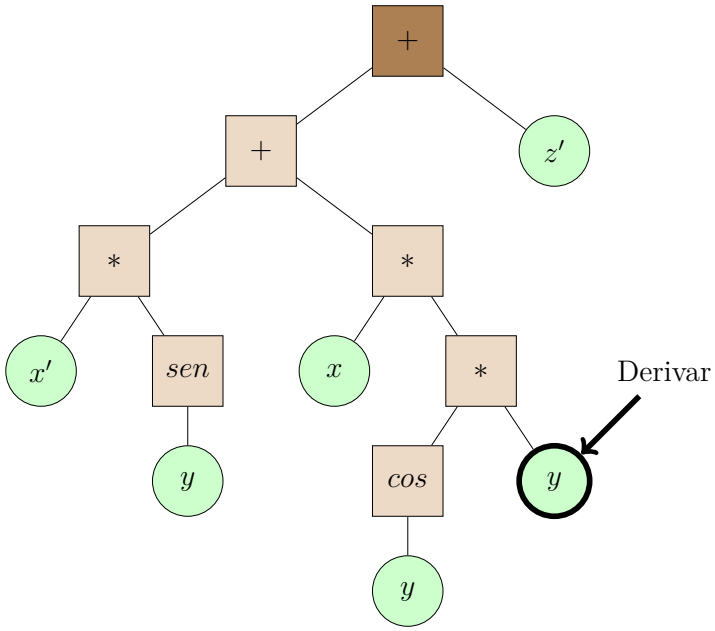
Passo 2: $f(a,b) = a.b \rightarrow f'(a,b) = a'.b + a.b'$



Passo 3: $f(a) = sen(a) \rightarrow f'(a) = cos(a).a'$



Passo 4: Derivar y e obter a árvore final ilustrada na Figura 16



3.1.7.2 Derivação simbólico-numérica

Assim como as operações nativas, as operações definidas pelo usuário também devem apresentar uma expressão para sua derivada com relação ao tempo. Dependendo da forma como funcionam tais operações, nem sempre é possível se obter uma expressão matemática para suas derivadas. A exemplo disto têm-se as operações que se utilizam de recursões ou iterações para obter seus resultados. Outro exemplo possível é o caso de operações que simplesmente recebem dados de ferramentas externas, servindo como meio de comunicação entre ferramentas, ao passo que não informa uma expressão para a derivada destes dados. Para tais operações é utilizada a técnica de derivação simbólico-numérica proposta por Costa Jr. (2003).

O termo derivação simbólico-numérica se dá por tratar de um artifício numérico que possibilita a obtenção de uma expressão simbólica para a derivada temporal de funções. A concepção desta técnica é simples e se baseia na aplicação da regra da cadeia, que é capaz de transformar um problema de derivação temporal em um problema de derivação parcial com relação a cada parâmetro de entrada da função. Dado o exemplo da função $f(a, b, c)$ e utilizando a regra da cadeia, sua derivada temporal pode ser escrita em notação de Leibniz conforme a Equação 3.1:

$$\frac{df(a,b,c)}{dt} = \frac{\partial f(a,b,c)}{\partial a}\frac{da}{dt} + \frac{\partial f(a,b,c)}{\partial b}\frac{db}{dt} + \frac{\partial f(a,b,c)}{\partial c}\frac{dc}{dt} \tag{3.1}$$

Para o caso genérico, seja $f(x)$ uma função tendo x como sendo o vetor que representa seus n parâmetros de entrada variantes no tempo, a expressão da derivada temporal de f após aplicada a regra da cadeia encontra-se na Equação 3.2.

$$\frac{df(x)}{dt} = \sum_{i=1}^{n} \frac{\partial f(x)}{\partial x_i}\frac{dx_i}{dt} \tag{3.2}$$

Avaliando as equações 3.1 e 3.2, vê-se que o problema de derivação temporal de f se transformou em um problema de derivação com relação às entradas de f. O artifício numérico por trás da técnica de derivação simbólico-numérica é substituir as derivadas parciais com relação às entradas por valores numéricos. Os valores numéricos em questão são obtidos utilizando a definição formal de derivadas através de limites, Equação 3.3, que pode ser uma boa aproximação desde que h seja suficientemente pequeno.

$$\frac{\partial f(x)}{\partial x_i} = \lim_{h \to 0} \frac{f(x_0, ..., x_i + h, ..., x_n) - f(x)}{h} \tag{3.3}$$

Detalhes sobre a obtenção de um valor adequado para h na Equação 3.3 serão tratados na subseção 3.4.2.

3.2 Camada de modelagem

Na camada de modelagem se encontram implementados os objetos e métodos utilizados na modelagem de processos. Esta é a camada que instaura a forma como o usuário comum irá interagir com a ferramenta, bem por isso, ela é a grande responsável por determinar a sensação que o usuário terá ao modelar sistemas. Em virtude disso, as diretrizes por trás desta camada é propiciar ao usuário um processo de modelagem intuitivo, rápido, organizado, enxuto e com alto poder de representação.

3.2.1 Objetos e métodos

O corpo de objetos da camada de modelagem é composto por inúmeros objetos. Estes objetos se enquadram em, basicamente, seis finalidades distintas apresentadas na lista 3.3

3.3 – Lista de finalidades e objetos da camada de modelagem

- Representar o próprio modelo de processo em si
  - Classe *ProcessModel*
- Representar equipamentos
  - Classes filhas da classe *Equipment*
- Representar subprocessos
  - Classes filhas da classe *Subprocess*
- Determinar os terminais do modelo
  - Classe *FluidSource*
  - Classe *FluidOutlet*
  - Classe *ExternalComponent*
- Descrever as propriedades e estados dos fluidos que circulam no processo
  - Classe *Stream*
  - Classe *Fluid*
  - Classes filhas da classe *Substance*
- Prover uma visualização gráfica dos resultados de simulação
  - Classe *Graph*

Uma vez que se conheça os objetivos por trás de cada classe da camada de modelagem, parece bem possível explicar cada uma dessas classes seguindo uma abordagem

top-down, onde primeiro se apresentará a classe *ProcessModel* e se prosseguirá pelas classes até finalizar com a apresentação da classe *Graph*. Desta maneira, segue-se a explicação da classe *ProcessModel*.

A classe *ProcessModel* é a principal classe por trás da camada de modelagem. Ela implementa praticamente todos os métodos que caracterizam o processo de construção de um modelo (ver Algoritmo 6 da subseção 3.2.2). Os métodos da qual dispõe a classe *ProcessModel* e suas descrições podem ser vistos na lista 3.4.

3.4 – Lista de métodos dos objetos da classe *ProcessModel*

- Construtor: *ProcessModel(name)*
  - Esta é a instrução utilizada para criar uma instância de modelo de processo. Seu único parâmetro é o nome que lhe será dado.
- *ProcessModel.addEquipments(*equips)*
  - Esta é a instrução que responsável por incorporar instâncias de equipamentos ao modelo de processo. Este método pode receber quantos parâmetros forem necessário, sendo estes as instâncias dos equipamentos.
- *ProcessModel.connect(terminal1, terminal2)*
  - Esta é a instrução que realiza conexões, tanto de terminais de entrada e saída de fluidos, quanto de sinais de atuação e sensores de equipamentos. Este método possui apenas dois parâmetros que são os terminais os quais se deseja conectar.
- *ProcessModel.createModel(showInfo)*
  - Esta é a instrução que trata de avaliar e sintetizar o modelo global do processo. Este método não só analisa todo o modelo em busca de singularidades estruturais, mas também retorna ao usuário as informações detalhadas sobre o processo caso seu único parâmetro, *showInfo*, seja dado como verdadeiro.
- *ProcessModel.setInitialConditions(initConds)*
  - Esta é a instrução que define as condições iniciais do sistema. O conjunto de condições iniciais aceitáveis é exibido por meio do método *ProcessModel.createModel*. Os valores das condições iniciais devem ser informados através do parâmetro *initConds*.

- *ProcessModel.simulate(t0, tf, [dtMax], [tol])*
  - Esta é a instrução que executa a simulação do sistema, partindo de um instante inicial t_0, especificado pelo parâmetro $t0$, e seguindo até o instante t_f, especificado pelo parâmetro tf. Os parâmetros *dtMax* e *tol* são, respectivamente, os valores de passo máximo admitido e de tolerância absoluta admitida para o algoritmo de integração numérica.
- *ProcessModel.simulate(tf)*
  - Esta instrução possui o mesmo propósito do instrução anterior, porém utiliza do polimorfismo para definir um método *ProcessModel.simulate* que recebe apenas o parâmetro *tf*. Seu uso se limita a apenas avançar uma simulação previamente efetuada, sendo assim, não é possível utilizar este método sem antes executar o método anterior.

Passando agora a falar da classe *Equipment*, pode-se dizer que esta serve somente para prover e padronizar um conjunto de atributos e métodos para as classes que realmente importam: suas classes filhas. Ao se implementar modelos de equipamentos reais como tanques e compressores, as classes que representaram estes equipamentos deverão ser filhas da classe genérica *Equipment*. Tal classe organiza as informações acerca dos modelos através de cinco métodos que encontra-se explicados na lista 3.5

3.5 – Lista de métodos dos objetos da classe *Equipment*

- Construtor: *Equipment(parameters)*
  - Esta é a instrução utilizada para criar uma instância de equipamento. É importante ressaltar que o construtor utilizado para instanciar equipamentos não é o da classe *Equipment*, mas sim o de suas classes filhas. Cada classe de equipamento exigirá seus próprios parâmetros, representados aqui pelo parâmetro *parameters*.
- *Equipment.addInlets(*inlets)*
  - Esta é a instrução utilizada para definir o conjunto de entradas de fluido que o equipamento possui. Este método pode possuir quantos parâmetros forem necessários, **inlets*, sendo cada um deles um objeto da classe *Stream*.
- *Equipment.addOutlets(*outlets)*
  - Esta é a instrução utilizada para definir o conjunto de saídas de fluido que o equipamento possui. Este método pode possuir quantos parâmetros forem necessários, **outlets*, sendo cada um deles um objeto da classe *Stream*.

- *Equipment.addInputs(*inputs)*
  - Esta é a instrução utilizada para definir o conjunto de entradas, ou formas de atuação sobre o equipamento. Este método pode possuir quantos parâmetros forem necessários, **inputs*, sendo cada um deles uma variável que compõe o modelo matemático do equipamento.
- *Equipment.addOutputs(*outputs)*
  - Esta é a instrução utilizada para definir o conjunto de saídas, ou sinais de sensores presentes no equipamento. Este método pode possuir quantos parâmetros forem necessários, **outputs*, sendo cada um deles uma variável que compõe o modelo matemático do equipamento.
- *Equipment.addEquations(*equations)*
  - Esta é a instrução utilizada para incorporar equações matemáticas ao modelo do equipamento. Este método pode possuir quantos parâmetros forem necessários, **equations*, sendo cada um deles uma instância de equação matemática.

Seguindo com as explicações, passa-se a falar da classe *Subprocess*. Esta classe, assim como a classe *Equipment*, tem por intuito apenas para servir como base para suas classes filhas. A classe de subprocessos pode ser entendida como uma união das classes *ProcessModel* e *Equipments*, uma vez que busca encapsular um modelo completo de um processo e dispô-lo na forma de uma estrutura com entradas e saídas assim como um objeto da classe *Equipment*. Por esta razão, a lista de métodos da qual dispõe a classe *Subprocess*, lista 3.6, é praticamente a união das listas 3.4 e 3.5.

3.6 – Lista de métodos dos objetos da classe *Subprocess*

- Construtor: *Subprocess(parameters)*
  - Esta é a instrução utilizada para criar uma instância de subprocesso. É importante ressaltar que o construtor utilizado para instanciar um subprocesso não é o da classe *Subprocess*, mas sim o de suas classes filhas. Cada classe de subprocesso exigirá seus próprios parâmetros, representados aqui pelo parâmetro *parameters*.
- *Subprocess.addInlets(*inlets)*
  - Esta é a instrução utilizada para definir o conjunto de entradas de fluido que o subprocesso possui. Este método pode possuir quantos parâmetros forem necessários, **inlets*, sendo cada um deles um objeto da classe *Stream*.

- *Subprocess.addOutlets(*outlets)*
  - Esta é a instrução utilizada para definir o conjunto de saídas de fluido que o subprocesso possui. Este método pode possuir quantos parâmetros forem necessários, **outlets*, sendo cada um deles um objeto da classe *Stream*.
- *Subprocess.addInputs(*inputs)*
  - Esta é a instrução utilizada para definir o conjunto de entradas, ou formas de atuação sobre o subprocesso. Este método pode possuir quantos parâmetros forem necessários, **inputs*, sendo cada um deles uma variável que compõe o modelo matemático de seus equipamentos componentes.
- *Subprocess.addOutputs(*outputs)*
  - Esta é a instrução utilizada para definir o conjunto de saídas, ou sinais de sensores presentes no subprocesso. Este método pode possuir quantos parâmetros forem necessários, **outputs*, sendo cada um deles uma variável que compõe o modelo matemático de seus equipamentos componentes.
- *Subprocess.addEquipments(*equips)*
  - Esta é a instrução que responsável por incorporar instâncias de equipamentos ao subprocesso. Este método pode receber quantos parâmetros forem necessário, sendo estes as instâncias dos equipamentos.
- *Subprocess.connect(terminal1, terminal2)*
  - Esta é a instrução que realiza conexões, tanto de terminais de entrada e saída de fluidos, quanto de sinais de atuação e sensores de equipamentos. Este método possui apenas dois parâmetros que são os terminais os quais se deseja conectar.

As próximas classes a serem estudadas são as classes *FluidSource*, *FluidOutlet* e *ExternalComponent*. Estas classes representam os terminais de um cenário de simulação, ou seja, elas não só determinam as características das fontes e sumidouros de fluidos do sistema, como também fornecem saídas de dados que podem ser utilizadas como sinais de atuação para os equipamentos. Esta última característica é marcante em objetos da classe *ExternalComponent*, que possui como finalidade encapsular uma fonte de dados externa à ferramenta e dispô-la na forma de um objeto com entradas e saídas, semelhante aos objetos de equipamentos. As fontes de fluido do sistema podem ser criadas tanto com objetos da classe *FluidSource* como com objetos da classe *ExternalComponent*. Para o caso dos objetos *FluidSource*, faz-se necessária a especificação da composição do fluido por meio da criação de um objeto *Fluid*. Já para os objetos *ExternalComponent*, a composição do fluido é setada por meios externos. Com relação aos métodos, somente os da classe

ExternalComponent são úteis aos usuários da ferramenta, lista 3.7. Na lista 3.8 aparecem somente os construtores das classes *FluidSource* e *FluidOutlet*.

3.7 – Lista de métodos dos objetos da classe *ExternalComponent*

- Construtor: *ExternalComponent(name, *componentAddress)*
  - Esta é a instrução utilizada para criar uma instância de componente externo. O parâmetro *name* é o nome que será dado ao componente, já o conjunto de parâmetros **componentAddress* são todas as informações necessárias para definir o endereço do componente externo na rede.
- *ExternalComponent.setConnectFunction(connectFunction)*
  - Esta é a instrução utilizada para definir a função a qual a classe *ExternalComponent* utilizará para estabelecer uma conexão com o componente externo. O parâmetro *connectFunction* é a função que estabelece a conexão, sendo que esta deve possuir como parâmetros os mesmos campos de endereço definidos no conjunto de parâmetros **componentAddress* do construtor da classe *ExternalComponent*.
- *ExternalComponent.setCloseConnectionFunction(closeConnectionFunction)*
  - Esta é a instrução utilizada para definir a função a qual a classe *ExternalComponent* utilizará para encerrar uma conexão com o componente externo. O parâmetro *closeConnectionFunction* é a função que encerra a conexão, sendo que esta não pode necessitar de nenhum parâmetro para ser chamada.
- *ExternalComponent.setSendFunction(sendFunction)*
  - Esta é a instrução utilizada para definir a função a qual a classe *ExternalComponent* utilizará para enviar uma mensagem ao componente externo. O parâmetro *sendFunction* é a função que executa o envio de mensagens, sendo que esta deve, necessariamente, possuir apenas um único parâmetro que será o conteúdo da mensagem na forma de uma *string*.
- *ExternalComponent.setReceiveFunction(receiveFunction)*
  - Esta é a instrução utilizada para definir a função a qual a classe *ExternalComponent* utilizará para receber uma mensagem enviada pelo componente externo, ou esperar por ela. O parâmetro *receiveFunction* é a função para recepção de mensagens, sendo que esta, necessariamente, precisa retornar o conteúdo da mensagem crua na forma de uma *string*. A função de recepção *receiveFunction* deve ser de espera ocupada.

Uma vez criado um objeto do tipo *ExternalComponent*, ele irá dispor de entradas de fluidos, saídas de fluidos e sinais de atuação. Para conectar estes terminais às entradas ou saídas dos equipamentos do modelo, basta utilizar o método *connect* da classe *ProcessModel*. É importante dizer que nenhum dos terminais de um *ExternalComponent* precisa ser instanciado como acontece durante a modelagem de equipamentos. Tais terminais, na realidade, são criados assim que se tenta acessá-los e conectá-los aos terminais dos equipamentos. Para elucidar melhor como funciona a classe *ExternalComponent* construiu-se o Algoritmo 9 reproduzido na subseção 3.2.3.

3.8 – Construtores das classes *FluidSource* e *FluidOutlet*

- Construtor: *FluidSource(name, fluid, Res, presExpr, flowExpr, tempExpr)*
  - Esta é a instrução utilizada para criar uma instância de uma fonte de fluidos. Seus parâmetros são, respectivamente, o nome que será dado à fonte, o fluido que sairá da fonte (objeto da classe *Fluid*), a resistividade de escoamento da fonte, uma expressão para a pressão interna da fonte, uma expressão para a vazão de fluido na fonte e uma expressão para a temperatura na fonte. É possível especificar uma expressão para a pressão em *presExpr* e não especificar uma para a vazão em *flowExpr*. Caso isso aconteça, o sistema calculará automaticamente uma expressão para a grandeza faltante. O mesmo vale para a situação contrária.
- Construtor: *FluidOutlet(name, fluid, Res, EnvP, EnvT)*
  - Esta é a instrução utilizada para criar uma instância de um sumidouro de fluidos. Os sumidouros representam ambientes como, por exemplo, a atmosfera ou o mar. Seus parâmetros são, respectivamente, o nome que será dado ao sumidouro, o fluido existente no ambiente sumidouro (objeto da classe *Fluid*), a resistividade de escoamento em direção ao sumidouro, uma expressão para a pressão ambiente e uma expressão para a temperatura ambiente. Faz-se necessária a caracterização do fluido presente no ambiente sumidouro devido a possibilidade de, em certos casos de simulação, existirem fluxos reversos em equipamentos, ou seja, fluxos de fluido entrando em terminais por onde, supostamente, deveria apenas sair fluido. Sendo assim, é bem possível que sumidouros eventualmente atuem como fontes de fluido.

Dando prosseguimento, as próximas classes abordadas serão as classes *Stream*, *Fluids* e *Substance*. Começando pela classe *Stream*, esta é uma classe muito importante pois define o conjunto de variáveis que será usado para caracterizar o estado do fluido que supostamente circularia pelos equipamentos simulados. Por exemplo, as variáveis que geralmente são utilizadas para caracterizar o estado de um fluido são pressão, vazão, temperatura e composição, porém, a escolha de um conjunto de variáveis de estado depende fortemente dos modelos dos equipamentos e processos envolvidos na simulação. Devido a isso, os objetos da classe *Stream* também representam os terminais de entrada e saída de fluidos dos equipamentos do sistema, definindo assim que tal equipamento tem seu

modelo matemático voltado a utilização de um determinado conjunto de variáveis de estado de fluido. Tal característica impede que um equipamento projetado para considerar um determinado conjunto de variáveis de estado de fluido seja conectado a um segundo equipamento que não considera o mesmo estado e não saberia lidar com os dados entregues pelo primeiro. Todo objeto da classe *Stream* possui também um objeto da classe *Fluids*, que provê uma coleção de propriedades físico-químicas de fluidos na forma de variáveis para a construção de modelos matemáticos de equipamentos. A classe *Stream* não possui métodos úteis aos usuários, sendo assim, apresenta-se na lista 3.9 apenas o seu método construtor.

3.9 – Construtor da classe *Stream*

- Construtor: *Stream(name)*
  - Esta é a instrução utilizada para criar uma instância de um terminal de equipamento por onde entra ou sai fluidos. Estas instâncias são utilizadas, tanto como referências para que se realize conexões entre equipamentos por meio do método *connect* da classe *ProcessModel*, quanto para guardar um conjunto de variáveis que representará o estado do fluido que passa pelo terminal. Seu único parâmetro é o nome que lhe será dado.

A classe *Fluids* é responsável por caracterizar uma mistura de substâncias, objetos da classe *Substance*. A principal informação guardada por esta classe são as funções de concentração de cada substancia na mistura, podendo estas funções serem contantes ou funções temporais. A linha 13 do Algoritmo 6 traz um exemplo onde é criado um fluido chamado *idealGas* composto por gás hélio e gás argônio com concentrações variando ao longo do tempo, sendo estas, respectivamente, $sen(0,5t)^2$ e $cos(0,5t)^2$. Além das concentração, a classe *Fluids* também seria responsável por calcular propriedades fisico-químicas de misturas de substâncias, como por exemplo o estado físico de cada substância, o número de fases, entre outros. Para isso, tais propriedades devem ser caracterizadas através de equações, podendo estas envolver quaisquer propriedades das suas substâncias componentes, do próprio fluido e as propriedades de estado definidas pela classe *Stream*. Quando, em algum modelo de simulação, for utilizado algum equipamento que faz uso de alguma propriedade físico-química de mistura, a ferramenta de simulação automaticamente incluirá a equação responsável por calculá-la no conjunto de equações do sistema a ser simulado. A lista 3.10 exibe todos os métodos da classe *Fluids*.

3.10 – Lista de métodos dos objetos da classe *Fluids*

- Construtor: *Fluids(*substances)*
  - Esta é a instrução utilizada para criar uma instância de fluido. Um fluido consiste numa mistura de substâncias que cuja composição é determinada pelo parâmetro **substances*. Este parâmetro, na realidade, representa inúmeros parâmetros onde cada um deles deve ser um objeto da classe *Substance*. Internamente, um objeto da classe *Fluids* possui uma referência interna para o objeto da classe *Stream* que o possui.
- *Fluids.addProperties(*properties)*
  - Esta é a instrução utilizada para incorporar equações de propriedades físico-químicas aos objetos da classe *Fluids*. Este método pode possuir quantos parâmetros forem necessários, **properties*, sendo cada um deles um objeto de equação matemática.

A penúltima classe a ser abordada é a classe *Substance*. Os objetos desta classe, assim como os da classe *Fluids*, servem para guardar propriedades físico-químicas, porém agora de substâncias químicas. Exemplo de propriedades de substâncias são entalpia de formação, ponto de fusão, ponto de ebulição, massa específica, pressão de vapor, e assim por diante. Para quantificar tais propriedades, estas também devem ser definidas na forma de equações, podendo envolver quaisquer outras propriedades da própria substância, do próprio fluido a qual pertencem e as propriedades de estado definidas pela classe *Stream*. A lista 3.11 apresenta todos os métodos da classe *Substance*.

3.11 – Lista de métodos dos objetos da classe *Substance*

- Construtor: *Substance(name, fluid)*
  - Esta é a instrução utilizada para criar uma instância de substância. Seus parâmetros são *name* e *fluid* que representam, respectivamente, o nome da substância e uma referência ao fluido que a contém.
- *Substance.addProperties(*properties)*
  - Esta é a instrução utilizada para incorporar equações de propriedades físico-químicas aos objetos da classe *Substance*. Este método pode possuir quantos parâmetros forem necessários, **properties*, sendo cada um deles um objeto de equação matemática.

Parte-se agora para a última classe da camada de modelagem, a classe *Graph*. Esta possui como objetivo prover um ambiente de visualização gráfica para dados de simulação gerados pela ferramenta. Sua utilização é simples, contendo apenas dois métodos: o método *Graph.show* e o método *Graph.clear*. A lista 3.12 apresenta informações mais detalhadas sobre estes métodos.

3.12 – Lista de métodos dos objetos da classe *Graph*

- Construtor: *Graph(*curves)*
  - Esta é a instrução utilizada para criar uma instância de gráfico. Este método pode possuir múltiplos parâmetros, representados pelo parâmetro **curves*, sendo cada um deles uma curva a ser exibida no gráfico.
- *Graph.show()*
  - Esta é a instrução utilizada para exibir graficamente as curvas especificadas no método construtor desta classe. Este método não requer parâmetros.
- *Graph.clear()*
  - Esta é a instrução utilizada para apagar todos os dados de simulação das curvas especificadas no método construtor desta classe. Este método não requer parâmetros.

3.2.2 Codificação de modelos

Esta subseção tem por objetivo apresentar como se utiliza a ferramenta desenvolvida para modelar processos e equipamentos. Para isso, alguns exemplos serão apresentados e explicados para que se entenda como devem ser estruturados os modelos na ferramenta, o que é possível de se modelar e como utilizar os objetos apresentados na subseção 3.2.1.

3.2.2.1 Modelos de processos

Uma das principais diretrizes que nortearam as especificações de como seriam modelados os processo na ferramenta é a simplicidade. A forma como foram definidos os objetos e métodos dentro da camada de modelagem visava a construção de modelos intuitivos e enxutos, sendo necessária poucas linhas de código para descrevê-lo completamente. Para ilustrar a forma como ficou definida a construção de modelos de processos, apresenta-se o Algoritmo 6.

O Algoritmo 6 modela um processo simples onde existe uma fonte de gás ideal, dois dutos, um tanque e duas saídas de gás para a atmosfera. A linha 2 deste algoritmo é a responsável por criar o objeto *Mod* da classe *ProcessModel* no qual serão armazenadas as informações sobre o processo. Na linha 5 corre a criação de um fluido composto das

Algoritmo 6: Modelagem de um processo simples

```
# Creating Model
Mod = ProcessModel("Model name")

# Describe the Fluid
IdealGas = Fluid(Helium = sin(0.5*t)**2, Argon = cos(0.5*t)**2)

# Creating Equipments
Source1 = FluidSource("Fluid Source", IdealGas, 0.5, presFunc = 2 +
↪  0.3*cos(0.75*t))
Duct1 = Duct("Inlet Duct", 0.33)
Tank1 = Tank("Tank", 0.1, 0.25, 0.25, 0.20)
Duct2 = Duct("Outlet Duct", 0.2)
Outlet1 = FluidOutlet("Outlet", IdealGas, 0.16, 1, 300)
Flare1 = FluidOutlet("Flare", IdealGas, 0.14, 1, 300)
Mod.addEquipments(Source1, Duct1, Tank1, Duct2, Outlet1, Flare1)

# Describing Connections - Streams
Mod.connect(Source1.Out, Duct1.In)
Mod.connect(Duct1.Out, Tank1.In)
Mod.connect(Tank1.Out, Duct2.In)
Mod.connect(Tank1.Relief, Flare1.In)
Mod.connect(Duct2.Out, Outlet1.In)

# Describing Connections - Inputs and Outputs
Mod.connect(Tank1.Io, 1)
Mod.connect(Tank1.Oo, 0.7 + 0.2*cos(t))

# Creating the Global Model - Optional
Mod.createModel(True)

# Setting the Initial Conditions - Depends on the case
InitCond = {Tank1.Mt0: 2}
Mod.setInitialConditions(InitCond, True)

# Setting Graphics - Optional
Graph1 = Graph("Molar Mass in the Tank", Tank1.Mt)
Graph2 = Graph("Comparison between Inlet and Outlet Flows", Tank1.In.F,
↪  Tank1.Out.F)
Graph3 = Graph("Relief Stream Variables", Tank1.Relief)
Graph4 = Graph("Relation between Molar Mass and Internal Energy", (Tank1.Mt,
↪  Tank1.E))

# Simulating the Global Model
Mod.simulate(0.0, 15.0)

# Showing Results
Graph1.show()
Graph2.show()
Graph3.show()
Graph4.show()
```

Linguagem de programação: *python 3.7*

substâncias hélio e argônio. A concentração destas substâncias no fluido variam ao longo do tempo e são determinadas pelas expressões $sen(0,5.t)^2$ e $cos(0,5.t)^2$.

Os equipamentos todos são instanciados nas linhas de 8 a 13 sendo estes adicionados ao modelo na linha 14. As linhas de 17 a 21 realizam as conexões entre as entradas e saídas de fluidos dos equipamentos. Já nas linhas 24 e 25 são atribuídas expressões matemáticas para as variáveis de atuação sobre o tanque, *Tank1.Io* e *Tank1.Oo*, que são, respectivamente, os sinais de abertura da válvula de entrada e da válvula de saída do tanque.

O método *createModel* que aparece na linha 28 é um comando opcional que serve para avaliar o sistema em busca de inconsistências e apresentar ao usuário os conjuntos de variáveis que necessitam de condições iniciais (ver subseção 3.3.5). Conhecendo estas variáveis, pode-se definir um valor para suas condições iniciais, o que acontece nas linhas 31 e 32. Dependendo do modelo, podem não existir variáveis que necessitam de condições iniciais, desta forma, as linhas 31 e 32 não devem ser utilizadas. Seguindo adiante, as linhas de 35 a 38 configuram gráficos cujos dados são exibidos graficamente através do comando *show*, presente nas linhas de 44 a 47. Os dados somente estarão disponíveis aos gráficos após a execução do comando *simulate*, localizado na linha 41.

3.2.2.2 Modelos de equipamentos

Durante a descrição de processo efetuada pelo Algoritmo 6, presente na subseção 3.2.2.1, foi utilizado o modelo de dois equipamentos distintos: o equipamento *Duct* e o equipamento *Tank*. Tais equipamentos foram previamente modelados e disponibilizados na biblioteca de equipamentos da ferramenta para que pudessem ser utilizados. O Algoritmo 7 apresenta a forma como o equipamento *Tank* foi modelado.

Como pode ser visto no Algoritmo 7, a modelagem de equipamentos é bastante simples. Tudo começa com a construção da classe de equipamento, no caso a classe *Tank*, especificando a classe *Equipment* como sua classe mãe, linha 1. Após isso, parte-se para a definição dos parâmetros que serão requeridos do usuário quando este instanciar o equipamento. No caso do tanque, os parâmetros requeridos aparecem na linha 2 e são, respectivamente, o seu nome (*name*), o volume interno do tanque (*Volume*), a resistividade da válvula de entrada do tanque (*ResIn*), a resistividade da válvula de saída (*ResOut*), a resistividade da válvula de escape (*ResRelief*) e, por fim, os valores de pressão mínima e máxima que determinarão o fechamento e abertura da válvula de escape (*reliefPressures*). Todos estes parâmetros, com exceção de *name*, apresentam um valor predefinido para caso, por algum motivo, o usuário do modelo não queira especificar os valores dos parâmetros.

Continuando com as explicações, segue-se ao bloco que vai da linha 5 até a linha 10. É neste momento em que são definidas as entradas e saídas de fluidos que o equipamento possui. Neste caso, é definida uma entrada, instanciada como *In* na linha 6, e duas saídas, instanciadas como *Out* e *Relief* nas linhas 7 e 8. As instâncias de entrada e saída de

Algoritmo 7: Modelagem de um tanque simples para gases ideais

```
class Tank(Equipment):
    def __init__(self, name: str, Volume: float = 1, ResIn: float = 0.25, ResOut:
     ↪  float = 0.25, ResRelief: float = 0.2, reliefPressures: tuple = None):
        self.name = name

        # Inlets and Outlets
        In = Stream("In")
        Out = Stream("Out")
        Relief = Stream("Relief")
        self.addInlets(In)
        self.addOutlets(Out, Relief)

        # Constants
        V = Volume
        Ci = 1/ResIn
        Co = 1/ResOut
        Cr = 1/ResRelief
        Pmin = reliefPressures[0] if reliefPressures is not None else None
        Pmax = reliefPressures[1] if reliefPressures is not None else None

        # Variables
        Mt = continuousVariable("Mt", limits = (0, inf))
        P = continuousVariable("P")
        T = continuousVariable("T")
        E = continuousVariable("E")
        Io = continuousVariable("Io", limits = (0,1))
        Oo = continuousVariable("Oo", limits = (0,1))
        Fo = booleanVariable("Fo")

        # Inputs and Outputs
        self.addInputs(Io, Oo)
        self.addOutputs(P, T)

        # Equations - Internal
        eq0 = diff(Mt) <<equal>> In.F - Out.F - Relief.F
        eq1 = P <<equal>> Mt*0.000082*In.T/V
        eq2 = E <<equal>> 3*P*V/2
        eq3 = T <<equal>> In.T
        if reliefPressures is not None:
            eq4 = new(Fo) <<equal>> ((P >= Pmax) | (Fo & ~(P <= Pmin)))
        else:
            eq4 = Fo <<equal>> True

        # Equations - Input/Output
        eq5 = In.F <<equal>> Io*Ci*(In.P - P)
        eq6 = Out.F <<equal>> Oo*Co*(P - Out.P)
        eq7 = Out.T <<equal>> In.T
        eq8 = Out.z <<equal>> In.z
        eq9 = Relief.F <<equal>> Fo*Cr*(P - Relief.P)
        eq10 = Relief.T <<equal>> In.T
        eq11 = Relief.z <<equal>> In.z

        self.addEquations(eq0, eq1, eq2, eq3, eq4, eq5, eq6, eq7, eq8, eq9, eq10,
         ↪  eq11)
```

Linguagem de programação: *python 3.7*

fluidos são objetos da classe *Stream*, explicados na subseção 3.2.1, porém, o que realmente determina quais instâncias serão consideradas entradas ou saídas são os métodos executados nas linhas 9 e 10.

O bloco que vai da linha 12 até a linha 18 trata somente da definição de constantes que serão utilizadas nas equações do equipamento. Para se definir equações, além das constantes, serão também necessárias variáveis. Estas são definidas no bloco das linhas 20 a 27, onde ocorre a criação de seis variáveis contínuas, Mt, P, T, E, Io e Oo, e uma booleana, Fo. A variável Mt representa a massa molar dentro do tanque, a variável P é sua pressão interna, T é a temperatura interna, E é a energia interna do gás dentro do tanque, Io é a abertura da válvula de entrada gás no tanque, Oo é a abertura da válvula de saída e Fo é a abertura da válvula de escape.

As linhas de 29 a 31 marcam o bloco onde se define quais variáveis serão consideradas entradas do equipamento e quais serão consideradas saídas. As entradas são os sinais que atuam sobre o equipamento e as saídas podem ser entendidas como os sinais de seus sensores. Neste exemplo, as aberturas das válvulas de entrada e saída de gás, Io e Oo, são consideradas meios de atuação sobre o equipamento. Já as variáveis de pressão e temperatura interna, P e T, são disponibilizadas como dados de sensores.

Por fim, a etapa mais importante da construção do modelo do *Tank* encontra-se nas linhas de 33 a 52. Nestas são formados três blocos de código onde o primeiro, linhas 33 a 41, define as equações ligadas as variáveis internas do tanque, o segundo, linhas 43 a 50, define as equações ligadas as variáveis de estado dos fluidos de entrada e saída, e o terceiro, linha 52, efetua a adição das equações ao modelo do equipamento. É interessante notar que existem duas possibilidades para a equação $eq4$, linhas 39 e 41. Caso o usuário, no momento da instanciação do equipamento, definir valores para as pressões de abertura e fechamento da válvula de escape, *reliefPressures*, a equação escolhida para $eq4$ considerará um controle automático *on-off* com histerese para a válvula de escape, linha 39, caso contrário, considerará a válvula sempre aberta, linha 41.

3.2.2.3 Modelos de subprocessos

Uma vez explicada a modelagem de processos e de equipamentos, fica simples entender a modelagem de subprocessos. Os métodos utilizados para a modelagem são os mesmos explicados nas subseções 3.2.2.1 e 3.2.2.2, sendo assim, não é necessário explicá-los novamente. O Algoritmo 8 apresenta um exemplo de subprocesso, onde existem nele um tanque e um trocador de calor, ambos controlados por controladores PID. As variáveis controladas do tanque e do aquecedor são, respectivamente, a pressão e temperatura. Esta informação é facilmente notada nas linhas 35 e 38. A sintonia destes controladores é colocada como parâmetro do subprocesso, linha 2, enquanto os sinais de referência destes são definidos como entradas do subprocesso na linha 22.

Algoritmo 8: Modelagem de um subprocesso simples envolvendo um tanque e um trocador térmico controlado

```
class TankHeaterProcess:
    def __init__(self, name, Kp1, Ki1, Kd1, Kp2, Ki2, Kd2):
        self.name = name

        # Creating Equipments
        Duct1 = Duct("Inlet Duct", 0.33)
        Tank1 = Tank("Tank", 0.1, 0.25, 0.25, 0.20)
        Heater1 = HeatExchanger("Inlet Heater", 0.20, 400)
        Duct2 = Duct("Outlet Duct", 0.2)
        PID1 = PIDController(Kp1, Ki1, Kd1)
        PID2 = PIDController(Kp2, Ki2, Kd2)
        self.addEquipments(Duct1, Tank1, Heater1, Duct2, PID1, PID2)

        # Inlets and Outlets
        In = Stream("In")
        Out = Stream("Out")
        Relief = Stream("Relief")
        self.addInlets(In)
        self.addOutlets(Out, Relief)

        # Inputs and Outputs
        self.addInputs(PID1.r, PID2.r)
        self.addOutputs(Tank1.P, Heater1.T)

        # Describing Connections - Streams
        self.connect(In, Duct1.In)
        self.connect(Duct1.Out, Tank1.In)
        self.connect(Tank1.Out, Heater1.In)
        self.connect(Tank1.Relief, Relief)
        self.connect(Heater1.Out, Duct2.In)
        self.connect(Duct2.Out, Out)

        # Describing Connections - Inputs and Outputs
        self.connect(Tank1.Oo, PID1.uc)
        self.connect(Tank1.P, PID1.y)
        self.connect(Tank1.Io, 1)
        self.connect(Heater1.Ho, PID2.uc)
        self.connect(Heater1.T, PID2.y)
```

Linguagem de programação: *python 3.7*

3.2.3 Integração com ferramentas externas

Na subseção 3.2.1 foi apresentada a classe *ExternalComponent* cuja finalidade é encapsular uma fonte de dados externa à ferramenta e disponibilizá-la como fosse um equipamento. Para ilustrar seu uso, apresenta-se o Algoritmo 9, que consiste no mesmo processo modelado no Algoritmo 6, porém com a pressão interna do tanque *Tank1* controlada por um *ExternalComponent*.

Algoritmo 9: Modelagem de um processo controlado externamente

```
# Creating Model
Mod = ProcessModel("Model name")

# Describe the Fluid
IdealGas = Fluid(Helium = sin(0.5*t)**2, Argon = cos(0.5*t)**2)

# Creating Equipments
Source1 = FluidSource("Fluid Source", IdealGas, 0.5, presFunc = 2 +
↪  0.3*cos(0.75*t))
Tank1 = Tank("Tank", 0.25)
Outlet1 = FluidOutlet("Outlet", IdealGas, 0.16, 1, 300)
Flare1 = FluidOutlet("Flare", IdealGas, 0.14, 1, 300)
ExtComp1 = ExternalComponent("Controller", "localhost", 8080)
Mod.addEquipments(Source1, Tank1, Outlet1, Flare1, ExtComp1)

# Describing Connections - Streams
Mod.connect(Source1.Out, Tank1.In)
Mod.connect(Tank1.Out, Outlet1.In)
Mod.connect(Tank1.Relief, Flare1.In)

# Describing Connections - Inputs and Outputs
Mod.connect(Tank1.Io, ExtComp1.u1)
Mod.connect(Tank1.P, ExtComp1.y1)
Mod.connect(Tank1.Oo, 0.7 + 0.2*cos(t))

# Setting the Initial Conditions - Depends on the case
InitCond = {Tank1.Mt0: 2}
Mod.setInitialConditions(InitCond)

# Setting Graphics - Optional
Graph1 = Graph("Inner Pressure of the Tank and its reference", Tank1.P,
↪  ExtComp1.Ref)
Graph2 = Graph("Control signal", ExtComp1.u1)

# Simulating the Global Model
Mod.simulate(0, 10)

# Showing Results
Graph1.show()
Graph2.show()
```

Linguagem de programação: *python 3.7*

Observa-se na linha 12 do Algoritmo 9 a instanciação de um *ExternalComponent* que faz referência a alguma aplicação externa à ferramenta. Nota-se que o protocolo de comunicação escolhido para o *ExternalComponent* é o protocolo TCP/IP (KUROSE; ROSS, 2000), já que não foram definidas nenhuma função de conexão, desconexão, envio e recepção diferentes para a classe. Nota-se também nas linhas 21 e 22 que o processo modelado utiliza duas variáveis do objeto *ExtComp*: *u1* e *y1*. Estas variáveis serão a ligação entre a aplicação externa e a ferramenta durante o processo de simulação do modelo.

3.2.3.1 Protocolo de comunicação

Por padrão, a classe *ExternalComponent* foi projetada para funcionar como um cliente, seguindo o modelo cliente-servidor. Considera-se que este aspecto da ferramenta ainda se encontra em fase inicial, por isso, o protocolo de comunicação estabelecido até então é bastante simplificado. Uma vez estabelecida a conexão entre as aplicações, estas podem se comunicar utilizando um conjunto limitado de mensagens possíveis. Todas as mensagens devem ser do tipo texto (*string*) e obedecer à estrutura apresentada na Figura 18.

Figura 18 – Estrutura de mensagens do protocolo de comunicação da ferramenta

Identificador	Número de parâmetros	Parâmetros...

Na estrutura da Figura 18, o identificador, o número de parâmetros e os parâmetros em si devem todos serem separados por vírgulas. Em uma visão de redes de computador (KUROSE; ROSS, 2000), o protocolo de comunicação aqui estabelecido está posicionado na camada logo acima daquela escolhida para se efetuar a troca de mensagens no *ExternalComponent*. As listas de mensagens deste protocolo são as listas 3.13 e 3.14, sendo estas, respectivamente, a lista de mensagens que serão enviadas pela ferramenta e a lista de mensagens recebíveis por ela.

3.13 – Lista de mensagens que a ferramenta envia à aplicação externa

- Requerimento de condição inicial
  - Identificador: 0
  - Número de parâmetros: n
  - Parâmetros: n variáveis do componente externo
  - Propósito: Requisitar à aplicação externa uma expressão $f(x)$ para cada uma das n variáveis, onde a expressão f deve ser escrita conforme se escreveria na ferramenta de modelagem e x são as variáveis cujos valores a aplicação externa espera receber da ferramenta. Estas funções podem ser constantes, não necessariamente dependendo das variáveis x. Esta mensagem espera como resposta a mensagem de identificador 0 da lista 3.14.

- Requerimento de especificações de atuação
  - Identificador: 1
  - Número de parâmetros: n
  - Parâmetros: n variáveis do componente externo
  - Propósito: Requisitar à aplicação externa as especificações de natureza de cada uma das n variáveis. Esta mensagem espera receber como resposta a mensagem de identificador 1 da lista 3.14.

- Requerimento de valor de variáveis
  - Identificador: 2
  - Número de parâmetros: $n + 1$
  - Parâmetro 1: o instante t de simulação
  - Parâmetros: n variáveis do componente externo
  - Propósito: Requisitar à aplicação externa um valor constante para cada uma das n variáveis. Estas mensagens serão enviadas durante o processo de simulação. Esta mensagem espera como resposta a mensagem de identificador 2 da lista 3.14.

- Resposta de valor de variáveis
  - Identificador: 3
  - Número de parâmetros: n
  - Parâmetros: o valor das n variáveis requeridas
  - Propósito: Enviar à aplicação externa os n valores requeridos para variáveis especificadas por uma mensagem de identificador 3 vinda da aplicação.

3.14 – Lista de mensagens que a ferramenta recebe da aplicação externa

- Resposta de condição inicial
  - Identificador: 0
  - Número de parâmetros: n
  - Parâmetros: as n expressões requeridas
  - Propósito: Enviar à ferramenta de modelagem as n funções requeridas para as variáveis especificadas por uma mensagem de identificador 0 vinda da ferramenta.
- Resposta de especificações de atuação
  - Identificador: 1
  - Número de parâmetros: n
  - Parâmetros: as n naturezas requeridas
  - Propósito: Enviar à ferramenta de modelagem as n naturezas requeridas para variáveis especificadas por uma mensagem de identificador 1 vinda da ferramenta. Naturezas contínuas são representadas pelo caractere "c", as naturezas booleanas pela caractere "b"e as naturezas discretas por um valor numérico que será seu período de avanço discreto dt.
- Resposta de valor de variáveis
  - Identificador: 2
  - Número de parâmetros: n
  - Parâmetros: o valor das n variáveis requeridas
  - Propósito: Enviar à ferramenta de modelagem os n valores requeridos para variáveis especificadas por uma mensagem de identificador 2 vinda da ferramenta.
- Requerimento de valor de variáveis
  - Identificador: 3
  - Número de parâmetros: $n + 1$
  - Parâmetro 1: o instante t de simulação
  - Parâmetros: n variáveis do componente externo presentes no modelo simulado na ferramenta
  - Propósito: Requisitar à ferramenta um valor constante para cada uma das n variáveis. Estas mensagens serão enviadas durante um processo de simulação sendo efetuado na ferramenta. Esta mensagem espera como resposta a mensagem de identificador 3 da lista 3.13.

3.3 Camada de síntese de sistemas

Seguindo adiante com relação as camadas da ferramenta de modelagem, encontra-se a camada de síntese de sistemas. Ela é responsável por agrupar, organizar e unir os dados de modelagem provenientes da camada de modelagem e utilizá-los para sintetizar sistemas de equações. É nela em que se constrói o modelo global do sistema, unindo todas as equações de todos os equipamentos.

3.3.1 Redução de modelos

A primeira etapa no processo de síntese do modelo global é a obtenção de um modelo reduzido em número de variáveis. Esta redução acontece por meio dos dados gerados ao se criar as atribuições (ver subseção 3.1.5). As atribuições são responsáveis por criar as chamadas variáveis unificadas, que representam todo um conjunto de variáveis. Estas variáveis unificadas podem ser variáveis de fato, ou constantes caso alguma atribuição assim indique. Sabendo disso, pode-se dizer que o modelo reduzido do sistema nada mais é do que o modelo obtido através da substituição de todas as variáveis do modelo original por suas variáveis unificadas. O exemplo 3.4 mostra como funciona a redução de modelos. Neste exemplo, u_1 é a variável unificada resultante da atribuição $a = b$.

$$\begin{array}{ll} eq_1: & x + a.b = 5 \\ eq_2: & 2 + b.x = a^2 \\ eq_3: & a = b \end{array} \qquad \begin{array}{ll} eq_1: & x + u_1.u_1 = 5 \\ eq_2: & 2 + u_1.x = {u_1}^2 \end{array} \tag{3.4}$$

Modelo original: 3 equações e 3 variáveis — Modelo reduzido: 2 equações e 2 variáveis

Podem existir casos onde o processo de redução de modelos gera novas atribuições. Quando isso acontece, o sistema é simplificado novamente. O exemplo 3.5 ilustra uma situação deste tipo.

$$\begin{array}{ll} eq_1: & z^2(h^2 + x^2) = y \\ eq_2: & x^2 y = hz^2 \\ eq_3: & z = (x - y)h + 1 \\ eq_4: & x = y \end{array} \qquad \begin{array}{ll} eq_1: & z^2(h^2 + {u_1}^2) = u_1 \\ eq_2: & {u_1}^3 = hz^2 \\ eq_3: & z = 1 \end{array} \qquad \begin{array}{ll} eq_1: & h^2 + {u_1}^2 = u_1 \\ eq_2: & {u_1}^3 = h \end{array} \tag{3.5}$$

Modelo original: 4 equações e 4 variáveis — Primeira redução: 3 equações e 3 variáveis — Segunda redução: 2 equações e 2 variáveis

3.3.2 Divisão em subsistemas de equações

Para que o modelo global reduzido de um processo seja numericamente simulado, pode ser que seja necessário subdividi-lo em sistemas menores de natureza bem definida. Os casos em que tal subdivisão não seria necessária seria o de sistemas compostos unicamente de equações algébricas, ou unicamente de equações diferenciais. Para modelos compostos de equações algébricas e diferenciais, sistemas DAE, seria necessária a construção de dois subsistemas de equações, um para os eventos de inicialização e reinicialização das variáveis do modelo, e outro para a integração numérica.

A ferramenta de modelagem e simulação busca resolver casos de modelos híbridos, onde o modelo global reduzido pode envolver equações de naturezas contínua, discreta e booleana. Os casos híbridos envolvendo equações de natureza booleana não foram suficientemente estudados e não serão tratados neste relatório, apesar de a ferramenta desenvolvida permitir a modelagem de equações desta natureza. Sendo assim, este relatório se limitará a apresentar o caso em que o modelo global é composto de equações tanto de natureza contínuas quanto de natureza discreta.

Partindo de um modelo global de natureza mista, os subsistemas necessários para efetuar sua simulação são os que aparecem na lista 3.15.

3.15 – Lista de subsistemas de um modelo global de equações de natureza contínuo-discreta

- **Subsistema de equações para inicialização das variáveis**

  Este subsistema precisa ser um sistema quadrado de equações, bem definido, para que se possa obter sua solução. O conjunto de equações que o compõe é apresentado em 3.6.

$$\begin{aligned} &F(t_0, y_0, y'_0, d_0) \\ &I(t_0, y_0, y'_0, d_0) \\ &F^n(t_0, y_0, y'_0, d_0) \end{aligned} \tag{3.6}$$

  No sistema 3.6, F representa todas as equações do modelo global reduzido, I representa todas as condições iniciais do sistema e F^n representa as derivadas de ordem até n de algumas equações de F. As variáveis t_0, y_0, y'_0 e d_0 representam, respectivamente, o instante inicial de tempo, os valores iniciais das variáveis contínuas, os valores iniciais das derivadas das variáveis contínuas e os valores iniciais das variáveis discretas.

- **Subsistema de equações DAE para integração numérica**

  Este subsistema é o subsistema que será numericamente simulado pelos algoritmos de solução de sistemas DAE. O conjunto de equações que o compõe este subsistema é apresentado em 3.7, onde F_c representa todas as equações contínuas do modelo global reduzido e F^n representa as derivadas de ordem até n de algumas equações de F. As variáveis t, y, y' e d representam, respectivamente, o tempo, as variáveis contínuas, as derivadas das variáveis contínuas e as variáveis discretas do modelo.

$$\begin{aligned} F_c(t, y, y', d) \\ F^n(t, y, y', d) \end{aligned} \tag{3.7}$$

- **Subsistema de equações a diferenças**

  Este subsistema é utilizado para calcular os valores das variáveis discretas do sistema a cada evento discreto, com exceção do momento da inicialização. Os eventos discretos são todos os momentos em que cada variável discreta avança no tempo. Tais momentos são definidos através de um intervalo dt informado durante a criação da variável. O conjunto de equações que compõe este subsistema é apresentado em 3.8, onde D^k representa todas as equações a diferenças do modelo global reduzido devidamente avançadas. As variáveis k, s_t, s_y, $s_{y'}$ e d representam, respectivamente, amostragem atual dos amostradores da variável tempo (caso existam), amostragem atual dos amostradores das variáveis contínuas (caso existam), amostragem atual das derivadas das variáveis contínuas (caso existam) e as variáveis discretas do modelo.

$$D^k(s_t, s_y, s_{y'}, d) \tag{3.8}$$

- **Subsistema de atribuições**

  Este subsistema é utilizado para calcular os valores de variáveis as quais se aplicou o operador *new*, apresentado na subseção 3.1.3. O propósito por trás deste operador é possibilitar a modelagem de fenômenos com memória, explicados na subseção 3.1.5. O conjunto de equações que compõe este subsistema é apresentado em 3.9, onde N representa todas as atribuições do modelo global reduzido. As variáveis t, y, y', d e b representam, respectivamente, o tempo, as variáveis contínuas, as derivadas das variáveis contínuas, as variáveis discretas e as variáveis booleanas do modelo. No estágio atual da ferramenta, as únicas variáveis booleanas possíveis para b são aquelas na qual se aplicou o operador *new*.

$$N(t, y, y', d, b) \tag{3.9}$$

A obtenção do conjunto de condições iniciais I do subsistema 3.6 será explicada na subseção 3.3.5. Já a obtenção dos conjuntos de equações F^n e D^k dos subsistemas 3.6, 3.7 e 3.8 será explicada a seguir na subseção 3.3.3.

3.3.3 Redução de índice estrutural e redução de atrasos

Na subseção 3.3.2, viu-se que um modelo global híbrido necessita ser dividido em três subsistemas de equações para que possa ser simulado corretamente. Para isso, faz-se necessário encontrar o conjunto de equações F^n, que é composto por derivadas de algumas das equações do modelo original, e o conjunto de equações D^k, que consistem em todas as equações a diferenças do modelo original ajustadas para possuir, ao menos, uma variável no instante presente k. As equações derivadas do conjunto F^n surgem do processo de redução de índice, explicado na subseção 2.2.1, que acontece por meio da execução do Algoritmo 11 com o auxílio do Algoritmo 10. Estes algoritmos nada mais são do que adaptações dos algoritmos de Pelegrini (2007), Algoritmo 1 e Algoritmo 2, apresentados na subseção 2.2.3.3.

Existem alguns motivos que justificam as mudanças nos algoritmos de Pelegrini, o primeiro deles é possibilitar o cálculo do conjunto D^k, o segundo é permitir que o conjunto F^n seja calculado mesmo em situações onde existem variáveis discretas em equações contínuas do modelo global reduzido, como por exemplo $y' + y = noh(d[k], 2) + zoh(d[k-1])$. Vale mencionar que as equações envolvendo o operador *sample*, como por exemplo $e[k] = sample(y, st) - sample(r, st_2)$, são consideradas equações a diferenças, mesmo envolvendo variáveis contínuas. Como visto no exemplo apresentado, é possível o uso de múltiplas operações de amostragem, podendo cada uma delas possuir períodos de amostragem distintos. Tal aspecto faz com que a amostragem de uma variável contínua seja entendida como uma grandeza discreta em seu instante k e com intervalo $dt = st$ entre cada instante discreto.

Algoritmo 10: Pseudocódigo do Algoritmo 1 adaptado para também lidar com variáveis discretas

Entradas: ***G(U,V,E)***: grafo bipartido do sistema de equações, sendo U o conjunto de equações, V o conjunto de variáveis e E o conjunto de relações entre vértices de U e V. ***M***: um acoplamento do grafo G. ***v_{eq}***: vértice de equação analisado, sendo que $v_{eq} \in U$. ***coloridas***: conjunto de vértices de equação já analisados durante a execução do algoritmo. ***alg***: parâmetro que determina se o caminho de aumento pode terminar em um vértice de variável algébrica. ***dis***: parâmetro que determina se o algoritmo considerará apenas as variáveis discretas ou apenas as não discretas.

Saída: Retorna verdadeiro caso se consiga aumentar a cardinalidade do acoplamento M, caso contrário, retorna falso.

```
1  Função augmentMatching(G(U,V,E), M, v_eq, coloridas, alg, dis):
2      coloridas ← coloridas ∪ v_eq
3      para cada variável v_var da equação v_eq faça
4          elegível ← (v_var é variável discreta com atraso nulo e dis é verdadeiro) ou
             (v_var é uma variável diferenciada e dis é falso) ou (v_var não é variável
             discreta e alg é verdadeiro e dis é falso)
5          se elegível for verdadeiro e v_var ∉ M então
6              M ← M ∪ {v_eq, v_var}
7              retorna verdadeiro
8          fim
9      fim
10     para cada variável v_var da equação v_eq faça
11         elegível ← (v_var é variável discreta com atraso nulo e dis é verdadeiro) ou
             (v_var é uma variável diferenciada e dis é falso) ou (v_var não é variável
             discreta e alg é verdadeiro e dis é falso)
12         v_eq2 ← equação associada a v_var em M
13         se elegível for verdadeiro e v_eq2 ∉ coloridas então
14             se augmentMatching(G(U,V,E), M, v_eq2, coloridas, alg, dis) retornar
                 verdadeiro então
15                 M ← M ∪ {v_eq, v_var}
16                 retorna verdadeiro
17             fim
18         fim
19     fim
20     retorna falso
21 fim
```

Algoritmo 11: Pseudocódigo do Algoritmo 2 adaptado para também lidar com variáveis discretas

Entrada: ***G(U,V,E)***: grafo bipartido do sistema de equações, sendo U o conjunto de equações, V o conjunto de variáveis e E o conjunto de relações entre vértices de U e V. ***M***: um acoplamento do grafo G. ***dis***: parâmetro que determina se o algoritmo considerará apenas as variáveis discretas ou apenas as não discretas.

Saída: Retorna verdadeiro caso se consiga obter uma redução de índice até $\nu_s = 0$, em caso de falha, retorna falso

```
1  Função adjustModel(G(U,V,E), M, dis):
2      para cada equação v_eq ∈ U faça
3          coloridas ← ∅
4          se augmentMatching(G(U,V,E), M, v_eq, coloridas, falso, dis) retornar falso
            então
5              se dis é verdadeiro então
6                  v_eq ← v_eq avançado até possuir atraso nulo em alguma variável
7                  se augmentMatching(G(U,V,E), M, v_eq, coloridas, falso, dis)
                    retornar falso então
8                      retorna falso
9                  fim
10             senão
11                 marcadas ← coloridas
12                 coloridas ← ∅
13                 se augmentMatching(G(U,V,E), M, v_eq, coloridas, verdadeiro, dis)
                    retornar falso então
14                     retorna falso
15                 fim
16                 U' ← derivada de todas as equações do conjunto marcadas
17                 V' ← variáveis de todas as equações do conjunto U'
18                 U ← U ∪ U'
19                 V ← V ∪ V'
20                 E ← E ∪ relações entre os elementos de U' e V'
21             fim
22         fim
23     fim
24     retorna verdadeiro
25 fim
```

Analisando o Algoritmo 10, vê-se que as únicas mudanças deste com relação ao Algoritmo 1 acontecem nas linhas 1, 4, 11 e 14. Tais mudanças se resumem a presença de um novo parâmetro *dis* nas linhas 1 e 14, e a alteração do critério de elegibilidade das variáveis nas linhas 4 e 11. A inclusão do parâmetro *dis* busca fazer com que o critério de elegibilidade funcione da seguinte forma: Caso *dis* seja falso, o algoritmo deve considerar não elegíveis todas as variáveis discretas das equações analisadas. Caso *dis* seja verdadeiro, o algoritmo deve considerar apenas as variáveis discretas com atraso nulo como elegíveis, desconsiderando todas as outras. A tabela verdade utilizada para a determinação do critério de elegibilidade é a Tabela 11.

Tabela 11 – Tabela verdade para o critério de elegibilidade do Algoritmo 10

dis	***alg***	v_{var} **é variável diferenciada**	v_{var} **é variável discreta**	v_{var} **possui atraso nulo**	***elegível***
0	0	0	0	0	0
0	0	0	0	1	Não se aplica
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	Não se aplica
0	0	1	1	0	Não se aplica
0	0	1	1	1	Não se aplica
0	1	0	0	0	1
0	1	0	0	1	Não se aplica
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	Não se aplica
0	1	1	1	0	Não se aplica
0	1	1	1	1	Não se aplica
1	0	0	0	0	0
1	0	0	0	1	Não se aplica
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	Não se aplica
1	0	1	1	0	Não se aplica
1	0	1	1	1	Não se aplica
1	1	0	0	0	0
1	1	0	0	1	Não se aplica
1	1	0	1	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	Não se aplica
1	1	1	1	0	Não se aplica
1	1	1	1	1	Não se aplica

Passando agora ao Algoritmo 11, vê-se que este difere do Algoritmo 1, primeiramente, pelo nome da função na linha 1, pela inclusão do parâmetro M na linha 1, pela inclusão do parâmetro dis nas linhas 1, 4 e 13, pela retirada da linha 2 do algoritmo original e pela adição das linhas de 5 a 10. A ideia por trás da mudança é fazer com que o Algoritmo 11 funcione exatamente como o Algoritmo 1 quando o parâmetro dis for falso. Porém, quando o parâmetro dis for verdadeiro, o Algoritmo 11 deve realizar uma tarefa distinta.

Em uma situação prática, o parâmetro dis somente seria considerado verdadeiro quando o Algoritmo 11 estiver analisando um sistema de equações a diferenças. Neste caso, ao se tentar um aumento de cardinalidade de M na linha 4 do Algoritmo 11, a função *augmentMatching* deve considerar apenas as relações entre as equações a diferenças e as suas variáveis discretas com atraso nulo. Caso não se consiga aumentar a cardinalidade do sistema sob estas condições, o algoritmo seguirá até a linha 6, onde a equação analisada v_{eq} passará por um processo de avanço até que alguma de suas variáveis possua atraso nulo. Este procedimento é simples, basta diminuir o atraso de todas as variáveis da equação analisada pelo valor do menor atraso encontrado em suas variáveis. Depois deste procedimento, efetua-se novamente a tentativa de aumento de cardinalidade do acoplamento M na linha 7. Caso falhe novamente, é sinal de que o sistema de equações a diferença está sobredeterminado, caso consiga aumentar a cardinalidade, o algoritmo segue para a próxima equação, linha 2.

3.3.4 Análise estrutural do sistema

A análise estrutural de um sistema de equações tem por objetivo expor as singularidades estruturais que este possui, ver subseção 2.2.3.1. O método utilizado para evidenciar tais singularidades consiste na aplicação da Decomposição Dulmage-Mendelsohn ao grafo do sistema de equações a ser analisado, ver subseção 2.2.3.2. Para isso, é preciso que se conheça ao menos um dos acoplamentos máximos possíveis deste grafo. Tal informação pode ser obtida por meio da execução do Algoritmo 11.

O Algoritmo 12 apresenta o pseudocódigo da Decomposição Dulmage-Mendelsohn que utiliza do Algoritmo 11 para ajustar os sistemas de equações analisados e obter um de seus acoplamentos máximos. Este algoritmo de decomposição também utiliza do Algoritmo 13 para encontrar cada subsistema sobredeterminado G^+ e subdeterminado G^- presente no grafo do sistema de equações.

Algoritmo 12: Pseudocódigo da Decomposição Dulmage-Mendelsohn adaptada para uso do Algoritmo 11

Entrada: ***G(U, V, E)***: grafo bipartido do sistema de equações, sendo U o conjunto de equações, V o conjunto de variáveis e E o conjunto de relações entre vértices de U e V. ***dis***: parâmetro que determina se o algoritmo considerará apenas as variáveis discretas ou apenas as não discretas.

Saída: Retorna as listas contendo cada subsistema G^+, G^w e G^- do grafo $G(U, V, E)$

```
1  Função DMDecomposition(G(U,V,E), dis):
2      M ← ∅
3      G+_list ← ∅
4      G^w ← G
5      G-_list ← ∅
6      se adjustModel(G(U,V,E), M, dis) retornar verdadeiro então
7          para cada equação v_eq ∈ U faça
8              se v_eq ∉ M então
9                  G+ ← goAlternatingPaths(G(U,V,E), M, v_eq)
10                 G^w ← G^w \ G+
11                 G+_list ← G+_list ∪ G+
12             fim
13         fim
14         para cada variável v_var ∈ V faça
15             elegível ← (v_var é variável discreta e dis é verdadeiro) ou (v_var não é
                variável discreta e dis é falso)
16             se v_var ∉ M e elegível é verdadeiro então
17                 G- ← goAlternatingPaths(G(U,V,E), M, v_var)
18                 G^w ← G^w \ G-
19                 G-_list ← G-_list ∪ G-
20             fim
21         fim
22         retorna G+_list, G^w, G-_list
23     fim
24     retorna ∅, ∅, ∅
25 fim
```

Algoritmo 13: Pseudocódigo do algoritmo que obtém os subgrafos G^+ e G^- de um grafo $G(U, V, E)$

Entrada: ***G(U, V, E)***: grafo bipartido do sistema de equações, sendo U o conjunto de equações, V o conjunto de variáveis e E o conjunto de relações entre vértices de U e V. ***M***: um acoplamento do grafo G. ***v_i***: algum vértice do grafo G não pertencente ao acoplamento M.

Saída: Retorna o subgrafo G^+ ou G^-, composto por todos os vértices tocados por caminhos alternantes em $G(U, V, E)$ com relação ao acoplamento M e partindo de v_i

```
1  Função goAlternatingPaths(G(U,V,E), M, v_i):
2      se v_i é um vértice de equação então
3          G+(U+,V+,E+) ← {∅,∅,∅}
4          U+ ← U+ ∪ v_i
5          para cada variável v_var da equação v_i faça
6              se v_var ∉ V+ e v_var ∈ M então
7                  V+ ← V+ ∪ v_var
8                  v_eq ← equação relacionada a v_var no acoplamento M
9                  G* ← goAlternatingPaths(G(U,V,E), M, v_eq, dis)
10                 G+ ← G+ ∪ G*
11             fim
12         fim
13         retorna G+(U+,V+,E+)
14     senão
15         G−(U−,V−,E−) ← {∅,∅,∅}
16         V− ← V− ∪ v_i
17         para cada equação v_eq que utiliza a variável v_i faça
18             se v_eq ∉ U− então
19                 U− ← U− ∪ v_eq
20                 v_var ← variável relacionada a v_eq no acoplamento M
21                 G* ← goAlternatingPaths(G(U,V,E), M, v_var, dis)
22                 G− ← G− ∪ G*
23             fim
24         fim
25         retorna G−(U−,V−,E−)
26     fim
27 fim
```

O Algoritmo 12 é feito para funcionar conforme o procedimento descrito na subseção 2.2.3.2, onde se formará um conjunto de subgrafos G^+, representando as parcelas sobredeterminadas do sistema de equações analisado, o subgrafo G^w, representando a parcela bem determinada, e um conjunto de subgrafos G^-, representando as parcelas subdeterminadas. Toda a iteração realizada nas linha de 7 a 13 é responsável por encontrar os subsistemas G^+ e subtraí-los do subsistema G^w, funcionando de forma análoga à iteração feita nas linhas de 14 a 21 para o caso dos subsistemas G^-. A única diferença entre as duas iterações é a existência de um critério de elegibilidade definido na linha 15 e avaliado na linha 16. Este critério serve que o algoritmo considere apenas as variáveis que respeitam as restrições impostas pelo parâmetro *dis*. A tabela verdade para o critério de elegibilidade encontra-se representada na Tabela 12. Uma expressão alternativa para o critério de elegibilidade na linha 15 poderia ser: *elegível* ← **não** (v_{var} é variável discreta **ou exclusivo** *dis* é **verdadeiro**). Esta expressão também respeita a tabela verdade.

Tabela 12 – Tabela verdade para o critério de elegibilidade do Algoritmo 12

dis	**v_{var} é variável discreta**	***elegível***
0	0	1
0	1	0
1	0	0
1	1	1

Analisando o Algoritmo 13, vê-se que este é um algoritmo recursivo, devido as linhas 9 e 21, responsável por obter um subgrafo G^+ ou G^- do grafo G dependendo da classe do vértice v_i da qual partirão todos os caminhos alternantes recursivamente percorridos. Todos os vértices tocados por estes caminhos alternantes serão os vértices que irão compor os subsistemas G^+ ou G^-. Vê-se no Algoritmo 12 que este executará repetidas vezes o Algoritmo 13, uma para cada vértice não pertencente ao acoplamento máximo M do grafo G. Sendo assim, existirão tantos subgrafos G^+ e G^- quanto houverem vértices fora do acoplamento.

3.3.5 Estados e validação de condições iniciais

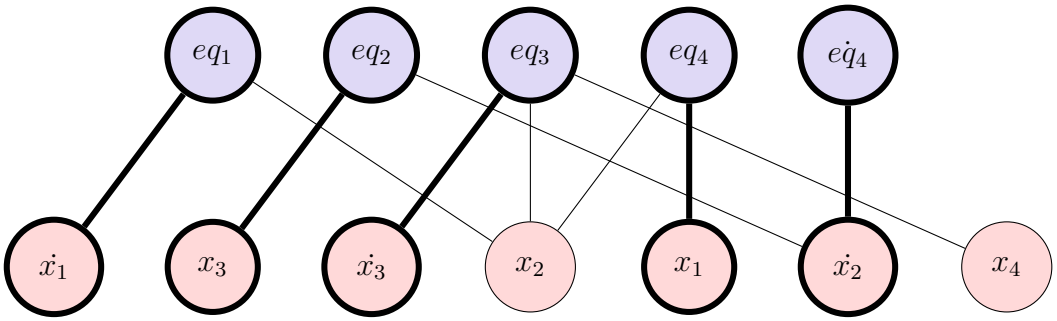
Para que se efetue a simulação numérica de um processo cujo modelo apresenta equações diferenciais, geralmente é necessário a definição de condições iniciais a certas variáveis do modelo, salvo o caso de sistemas DAE bem determinados por suas restrições escondidas, ver subseção 2.2.1. O trabalho de Pantelides (1988) surgiu justamente com o propósito de determinar um conjunto consistente de condições iniciais para problemas DAE. O algoritmo proposto por Pantelides possui algumas falhas de execução que foram corrigidas no trabalho de Pelegrini (2007).

As condições iniciais exigidas para a simulação numérica de um sistema de equações DAE podem ser vistas como subdeterminações deste sistema. Uma vez que se efetue um processo de redução de índice até zero em um sistema DAE, todas as restrições escondidas por este sistema serão reveladas, ver subseção 2.2.1. Cada restrição escondida diminui um grau de liberdade dinâmico do sistema DAE, o que altera o número de condições iniciais requeridas por este. Uma forma de se encontrar as prováveis condições iniciais requeridas por um sistema DAE consiste simplesmente em encontrar um sistema equivalente de índice estrutural zero para o sistema DAE, representar o sistema reduzido na forma de um grafo bipartido $G(U, V, E)$ e encontrar todos os subgrafos $G^-(U^-, V^-, E^-)$ deste grafo G por meio do Algoritmo 12. Cada subgrafo G^- representa uma subdeterminação do sistema, sendo que esta subdeterminação pode ser determinada através da adição de alguma equação que utilize ao menos uma das variáveis do conjunto V^-.

Para ilustrar o este processo de obtenção do conjunto de condições iniciais requeridas por um sistema DAE, apresenta-se um exemplo de sistema DAE, exemplo 3.10, e o grafo de seu modelo reduzido até índice estrutural zero.

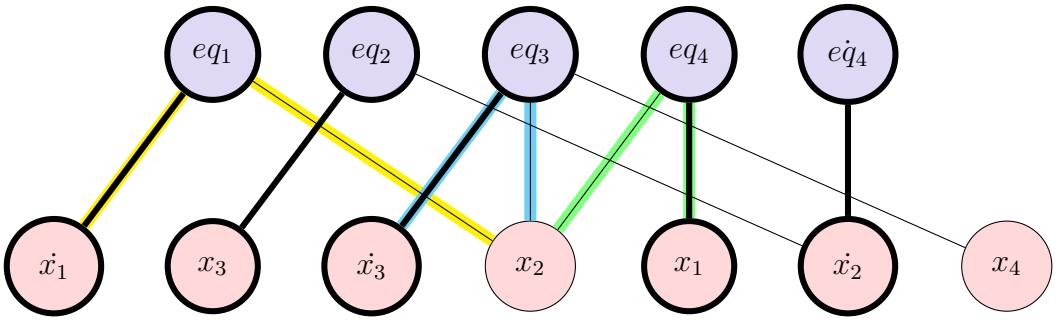
$$
\begin{array}{cc}
\begin{aligned}
eq_1 &: \quad \dot{x_1} = x_2 \\
eq_2 &: \quad \dot{x_2} = x_3 \\
eq_3 &: \quad \dot{x_3} = x_4 \ + \ x_2 \\
eq_4 &: \quad x_1 = {x_2}^2 + 3
\end{aligned}
&
\begin{aligned}
eq_1 &: \quad \dot{x_1} = x_2 \\
eq_2 &: \quad \dot{x_2} = x_3 \\
eq_3 &: \quad \dot{x_3} = x_4 \ + \ x_2 \\
eq_4 &: \quad x_1 = {x_2}^2 + 3 \\
\dot{eq_4} &: \quad \dot{x_1} = 2.x_2.\dot{x_2} \ \rightarrow \ \dot{x_2} = 0,5
\end{aligned}
\\
\text{Sistema original} & \text{Sistema com restrições reveladas}
\end{array}
\tag{3.10}
$$

Figura 19 – Grafo do sistema de equações 3.10 com um de seus acoplamentos máximos possíveis destacados

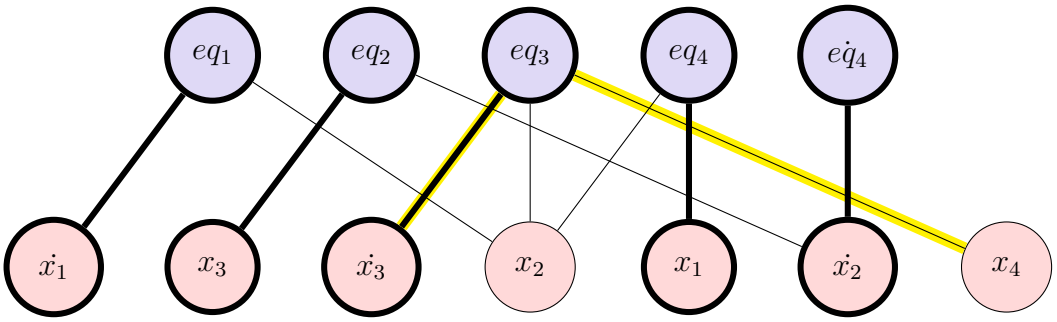


Através da aplicação do Algoritmo 12, pode-se encontrar os subgrafos G^- do grafo G da Figura 19. Para isso, faz-se necessário encontrar os caminhos alternantes de G com relação ao acoplamento máximo M_{max} partindo de vértices de variáveis não pertencentes a M_{max}. Este processo é desempenhado pelo Algoritmo 13 e encontra-se representado na Figura 20.

Figura 20 – Grafo do sistema de equações 3.10 com todos os caminhos alternantes destacados partindo de vértices de variáveis não pertencentes a M_{max}



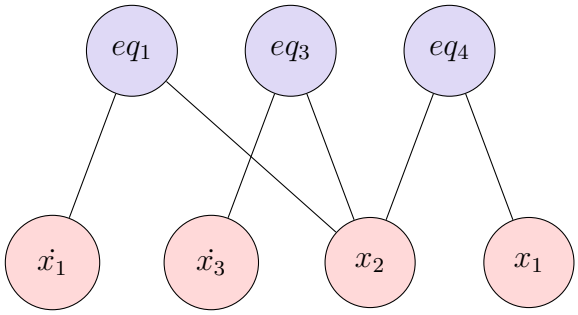
(a) Caminhos alternantes com relação a M_{max} partindo de x_2, destacados em diferentes cores



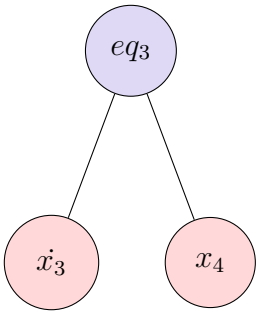
(b) O único caminho alternante com relação a M_{max} partindo de x_4, destacado em amarelo

Os subgrafos G^- revelados pelos caminhos alternantes da Figura 20 encontram-se representados na Figura 21.

Figura 21 – Subgrafos G^- do sistema de equações 3.10



Subgrafo G_1^-, originado do vértice de variável x_2



Subgrafo G_2^-, originado do vértice de variável x_4

A partir dos grafos representados na Figura 21, pode-se concluir que o sistema de equações 3.10 apresenta duas subdeterminações, o que significa que este deve receber duas condições iniciais para ser completamente determinado. As condições iniciais válidas para o sistema são: qualquer uma das variáveis do conjunto $\{x_1,\, x_2,\, \dot{x_1},\, \dot{x_3}\}$ e qualquer uma das variáveis do conjunto $\{x_4,\, \dot{x_3}\}$. Vê-se que a adição de equações como, por exemplo, $x_1 = 2$ e $x_4 = 0$ aos grafos da Figura 21 seria o suficiente para torná-los sistemas bem determinados, removendo a subdeterminação do sistema DAE. Na prática, quaisquer equações $f_2(x_1, x_2, \dot{x_1}, \dot{x_3}) = g_1(x_1, x_2, x_3, x_4, \dot{x_1}, \dot{x_2}, \dot{x_3})$ e $f_2(x_4, \dot{x_3}) = g_2(x_1, x_2, x_3, x_4, \dot{x_1}, \dot{x_2}, \dot{x_3})$ seriam suficientes para determinar as condições iniciais do sistema DAE estudado.

É importante dizer que nem toda a subdeterminação em variáveis de um sistema DAE deve ser entendida como condição inicial deste. Durante o processo de modelagem de equipamentos ou processos complexos, é bem possível que se esqueça de considerar algumas de suas equações descritivas. Estas falhas aparecerão como subdeterminações do modelo sendo entendidas como condições iniciais necessárias. Apesar de ser possível efetuar uma simulação considerando tais falhas desta maneira, é interessante que uma ferramenta de modelagem, de alguma forma, indique ao usuário quando existir a possibilidade de este estar esquecendo alguma equação. Para isso, define-se que são condições iniciais apenas as subdeterminações cujo conjunto de variáveis possui, ao menos, uma variável de estado. Considera-se variável de estado apenas as variáveis que permanecem contínuas mesmo na existência de eventos de descontinuidade em quaisquer equações do modelo. No caso do sistema 3.10, são consideradas estados as variáveis x_1, x_2 e x_3 já que estas não admitem descontinuidade por serem integrais de outras grandezas. Estas definições de estado e condição inicial serve somente para guiar o usuário da ferramenta durante o processo de modelagem. Tal aspecto não é relevante para o processo de simulação.

Passando agora ao processo de validação de condições iniciais, é preciso que se observe um determinado aspecto da Figura 21 para que se possa entender sua complexidade.

Na Figura 21, a variável $\dot{x_3}$ está presente, tanto no subgrafo G_1^-, quanto no subgrafo G_2^-. Tal fato significa que a variável $\dot{x_3}$ é capaz de determinar, tanto G_1^-, quanto G_1^-, porém, não ao mesmo tempo. Devido a simplicidade do exemplo 3.10, caso $\dot{x_3}$ e alguma outra variável fossem entregues pelo usuário como condições iniciais, facilmente se poderia determinar a qual subdeterminação a variável x_3 se refere, bastando apenas avaliar a outra variável primeiro. Este procedimento de avaliação não é tão simples quando se consideram sistemas DAE com muitas condições iniciais, existindo diversas variáveis do sistema repetidas nos subgrafos G^-, como acontece no caso de $\dot{x_3}$. Este problema complexo pode ser resolvido por meio da teoria de grafos.

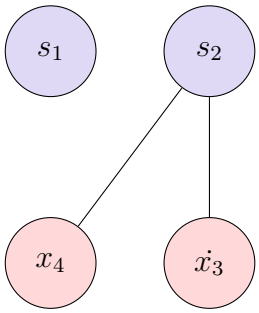
Para avaliar se um conjunto de condições iniciais informado pelo usuário é valido para determinar completamente um sistema de equação DAE, precisa-se, primeiramente, construir um grafo bipartido $S(U, V, E)$ em que os vértices de U representam as subdeterminações, os vértices de V representam as variáveis informadas como condições iniciais e as arestas de E representam as relações entre cada variável de V e as subdeterminações de U na qual estas variáveis estão presentes. Caso o acoplamento máximo do grafo S não for também um acoplamento perfeito, o conjunto de condições iniciais informado pelo usuário é inválido. Para exemplificar tal análise, supõe-se que o usuário tenha entregado as variáveis $\dot{x_1}$ e $\dot{x_3}$. O grafo S para a validação das condições iniciais aparece na Figura 22.

Figura 22 – Subgrafo S para avaliação de condições iniciais entregues pelo usuário ao sistema de equações 3.10

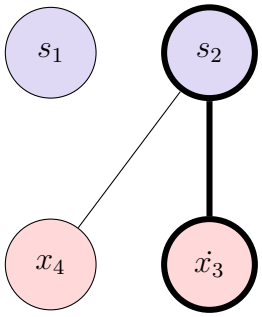
Grafo S Grafo S com acoplamento perfeito M_{per} destacado

Como na Figura 22 o grafo S possui um acoplamento perfeito, pode-se considerar que as condições iniciais entregues pelo usuário são válidas. Um caso de condições iniciais inválidas seria, por exemplo, o de entregar $\dot{x_3}$ e x_4 como condições iniciais inválidas. O grafo S para este caso juntamente com seu acoplamento máximo encontra-se representado na Figura 23.

Figura 23 – Subgrafo S para um conjunto inválido de condições iniciais entregues pelo usuário para o sistema de equações 3.10



Grafo S



Grafo S com um acoplamento máximo M_{max} destacado

O algoritmo responsável por verificar se as condições iniciais entregues para um sistema DAE é, de fato, consistente encontra-se representado pelo Algoritmo 14.

Algoritmo 14: Pseudocódigo do algoritmo que verifica se um conjunto de condições iniciais é consistente para um sistema DAE

Entrada: $\boldsymbol{G^-_{list}}$: lista de subgrafos G^- de um grafo G que representa um sistema de equações DAE. $\boldsymbol{IC}$: O conjunto de condições iniciais entregue pelo usuário.

Saída: Retorna verdadeiro caso o conjunto de condições iniciais IC seja válido, do contrário, retorna falso.

```
1  Função testInitialConditions(G⁻_list, IC):
2      S(U, V, E) ← {∅, IC, ∅}
3      para G⁻ em G⁻_list faça
4          v_s ← vértice de subdeterminação que representa G⁻
5          U ← U ∪ v_s
6          para v_var do conjunto IC faça
7              se v_var ∈ G⁻ então
8                  E ← E ∪ {v_s, v_var}
9              fim
10         fim
11     fim
12     M ← ∅
13     para v_s do conjunto U faça
14         se augmentMatching(S(U, V, E), M, v_s, ∅, verdadeiro, falso) retornar falso
           então
15             retorna falso
16         fim
17     fim
18     retorna verdadeiro
19 fim
```

3.3.6 Depuração do sistemas de equações de um processo

A depuração do sistema de equações de um processo, ou depuração do modelo global, é a responsável por reunir todas as informações estruturais do modelo e, caso não houverem singularidades, construir os subsistemas para inicialização, integração numérica, simulação discreta e atribuição de variáveis, todos definidos na subseção 3.3.2. O algoritmo que cumpre este papel é o Algoritmo 15, que informa ao usuário sobre as singularidades estruturais do sistema e obtém os conjuntos de equações necessários para a formação dos mencionados subsistemas.

Algoritmo 15: Pseudocódigo do algoritmo de depuração do modelo global de um processo

Entrada: ***Model***: modelo global do processo.

Saída: Retorna o conjunto de condições iniciais de plausíveis para o modelo global do processo

```
1  Função goAlternatingPaths(Model):
2      uMod ← versão unidimensional de Model
3      rMod ← redução do modelo uMod
4      N ← atribuições de rMod
5      F ← rMod \ N
6      F_c ← equações contínuas de F
7      D ← equações a diferenças de F
8      G_F ← grafo de F
9      G_{F_c} ← grafo de F_c
10     G_D ← grafo de D
11     G^+_{F_c}, G^w_{F_c}, G^-_{F_c} ← DMDecomposition(G_{F_c}, falso)
12     se G^+_{F_c} é vazio e G^-_{F_c} é vazio então
13         F^n ← equações de G^w_{F_c} - G_{F_c}
14     fim
15     senão se G^+_{F_c} é vazio então
16         Informa a lista de subdeterminações G^-_{F_c} ao usuário para que este possa
           definir as condições iniciais sistema
17     fim
18     senão
19         Informa as sobredeterminações G^+_{F_c} ao usuário
20     fim
21     G^+_D, G^w_D, G^-_D ← DMDecomposition(G_D, verdadeiro)
22     se G^+_D é vazio e G^-_D é vazio então
23         D^k ← equações de G^w_D
24     fim
25     senão
26         Informa as singularidades G^+_D e G^-_D ao usuário
27     fim
28 fim
```

Dentre os conjuntos necessários para compor o subsistemas de inicialização das variáveis, apenas o conjunto de equações I não é obtido por meio do Algoritmo 15. Este conjunto é definido pelo usuário da ferramenta e tem sua validade testada pelo Algoritmo 14.

3.4 Camada de simulação numérica

A camada de simulação numérica consiste na parte da ferramenta responsável por efetuar a simulação numérica dos processos modelados na ferramenta. Para tal propósito, são utilizados algoritmos de integração numérica para sistemas de equações diferenciais algébricas e de solução de sistemas não-lineares de equações algébricas.

Mostrou-se na seção 3.1 que a ferramenta desenvolvida possibilita a modelagem de várias classes de sistemas de equações. Atualmente, nem todos os sistemas de equações modeláveis na ferramenta podem ser numericamente simulados, especialmente aqueles envolvendo equações booleanas. Na lista 3.16 aparecem todos os sistemas modeláveis e suas situações atuais.

3.16 – Lista de sistemas modeláveis na ferramenta e suas situações atuais

- Sistemas NLA
  - Situação: Simulável
- Sistemas ODE
  - Situação: Simulável
- Sistemas DAE
  - Situação: Simulável
- Sistemas de equações a diferenças
  - Situação: Simulável
- Sistemas de equações booleanas
  - Situação: Não simulável
- Sistemas de atribuições
  - Situação: Simulável
- Sistemas híbridos contínuo-discretos
  - Situação: Simulável
- Sistemas híbridos contínuo-booleanos
  - Situação: Não simulável
- Sistemas híbridos discreto-booleanos
  - Situação: Não simulável
- Sistemas híbridos contínuo-discreto-booleanos
  - Situação: Não simulável

Viu-se na seção 3.3, em especial na lista 3.15 da subseção 3.3.2, que todo modelo de processo a ser simulado é estruturado na forma de um sistema híbrido contínuo-discreto composto de quatro subsistemas reproduzidos na lista 3.17 por conveniência.

3.17 – Lista de subsistemas para simulação de um processo

$$\text{Inicialização: } A \begin{cases} F(t_0, y_0, y'_0, d_0) \\ I(t_0, y_0, y'_0, d_0) \\ F^n(t_0, y_0, y'_0, d_0) \end{cases} \qquad \text{Atribuição: } C \begin{cases} N(t, y, y', d, b) \end{cases}$$

$$\text{Integração: } B \begin{cases} F_c(t, y, y', d) \\ F^n(t, y, y', d) \end{cases} \qquad \text{Simulação discreta: } D \begin{cases} D^k(s_t, s_y, s_{y'}, d) \end{cases}$$

Para que se resolvam numericamente os subsistemas A, B e D, os algoritmos de solução necessitarão conhecer a matriz jacobiana de cada um deles. Esta necessidade se

deve à maneira como tais algoritmos encontram a solução numérica, partindo de uma solução inicial provavelmente errada e melhorando-a através de alterações em cada variável do problema direcionadas pelos valores da matriz jacobiana. As matrizes jacobianas de cada subsistema é representada na lista 3.18.

3.18 – Matrizes jacobinas dos subsistemas da lista 3.17

$$J_A = \begin{bmatrix} \frac{\partial F}{\partial t_0} & \frac{\partial F}{\partial y_0} & \frac{\partial F}{\partial y'_0} & \frac{\partial F}{\partial d_0} \\ \frac{\partial I}{\partial t_0} & \frac{\partial I}{\partial y_0} & \frac{\partial I}{\partial y'_0} & \frac{\partial I}{\partial d_0} \\ \frac{\partial F^n}{\partial t_0} & \frac{\partial F^n}{\partial y_0} & \frac{\partial F^n}{\partial y'_0} & \frac{\partial F^n}{\partial d_0} \end{bmatrix} \qquad J_B = \begin{bmatrix} \frac{\partial F_c}{\partial t} & \frac{\partial F_c}{\partial y} & \frac{\partial F_c}{\partial y'} & \frac{\partial F_c}{\partial d} \\ \frac{\partial F^n}{\partial t} & \frac{\partial F^n}{\partial y} & \frac{\partial F^n}{\partial y'} & \frac{\partial F^n}{\partial d} \end{bmatrix}$$

$$J_D = \begin{bmatrix} \frac{\partial D^k}{\partial s_t} & \frac{\partial D^k}{\partial s_y} & \frac{\partial D^k}{\partial s_{y'}} & \frac{\partial D^k}{\partial d} \end{bmatrix}$$

As derivadas parciais com relação a t e t_0 das matrizes jacobianas da lista 3.18 são todas iguais a zero, já que os algoritmos de solução numérica avaliam as jacobianas para encontrar a solução do sistema de equações em determinado instante t, sendo assim, t não é uma variável do problema, mas sim uma constante que vai avançando ao longo do processo de simulação. As variáveis d durante o processo de solução numérica do subsistema B são consideradas todas constantes, portanto, d não é uma variável de B, fazendo com que a derivada parcial com relação a d não deva ser considerada para o jacobiano de B. O mesmo acontece para as derivadas parciais com relação a s_t, s_y e $s_{y'}$ na jacobiana J_D. O sistema real de jacobianas é apresentado na lista 3.19.

3.19 – Lista de matrizes jacobinas simplificadas dos subsistemas da 3.17

$$J_A = \begin{bmatrix} \frac{\partial F}{\partial y_0} & \frac{\partial F}{\partial y'_0} \\ \frac{\partial I}{\partial y_0} & \frac{\partial I}{\partial y'_0} \\ \frac{\partial F^n}{\partial y_0} & \frac{\partial F^n}{\partial y'_0} \end{bmatrix} \qquad J_B = \begin{bmatrix} \frac{\partial F_c}{\partial y} & \frac{\partial F_c}{\partial y'} \\ \frac{\partial F^n}{\partial y} & \frac{\partial F^n}{\partial y'} \end{bmatrix} \qquad J_D = \begin{bmatrix} \frac{\partial D^k}{\partial d} \end{bmatrix}$$

As derivadas parciais de cada equação é efetuada por meio de um operador que funciona de forma semelhante ao operador *diff* apresentado na subseção 3.1.3 e cujo funcionamento foi explicado na subseção 3.1.7 e na subseção 3.1.7.2. A derivada parcial com relação a uma variável x também se propaga ao longo das árvores n-ária que representam o lado direito e esquerdo das equações, porém, ignora todo o ramo que não possui a variável x. A derivada parcial de qualquer variável que não seja x é considerada nula e a derivada parcial de x com relação a x é considerada 1.

3.4.1 O solucionador numérico

Para que se resolvam numericamente os sistemas simuláveis da lista 3.16, faz-se necessário a criação de um algoritmo solucionador, do inglês *solver*, que realizará a estruturação do problema matemático e efetuará sua solução. O algoritmo solucionador, ou algoritmo de simulação numérica, estrutura e sequencia a solução dos subsistemas da lista 3.17 da forma como apresenta a lista 3.20. Para sistemas de equações algébricas não-lineares será utilizará a sigla NLA, do inglês *Nonlinear Algebraic*, já para sistemas de equações diferenciais algébricas, segue-se utilizando a sigla DAE.

3.20 – Procedimento de solução numérica

1. Inicializa-se todas as variáveis do sistema através da solução do subsistema A por um algoritmo de solução de sistemas NLA e efetua-se o procedimento exibido na lista 3.21;

2. Cria-se o conjunto de variáveis de amostragem com os seguintes valores: $s_t = t_0$, $s_y = y_0$ e $s_{y'} = y'_0$;

3. Cria-se as variáveis de atribuição associadas ao subsistema C com seus respectivos valores iniciais;

4. Reúne-se todos os períodos dt e st associados as variáveis discretas do sistema e às operações de amostragem;

5. Obtém-se uma lista de instantes discretos por meio do cálculo e ordenamento crescente de todos os múltiplos inteiros positivos dos períodos dt e st maiores que t_0 e menores que t_f, sendo t_0 e t_f os instantes de tempo inicial e final da simulação;

6. Inclui-se na lista de instantes discretos o instante t_f;

7. Define-se o instante atual t como sendo t_0 e o instante t_i como sendo o primeiro instante de tempo da lista ordenada de instantes discretos;

8. Efetua-se a integração numérica de B por meio de um algoritmo de solução de sistemas DAE. A integração deve partir do instante atual t e seguir até o instante t_i. A cada passo de integração:

   - Efetua-se o procedimento de monitoramento das variáveis exibido na lista 3.21.
   - Em seguida, soluciona-se o sistema C por meio de uma simples chamada de função. Se houverem mudanças em alguma das variáveis de atribuição, efetua-se a reinicialização do sistema no próximo passo de integração através da solução do subsistema B por um algoritmo de solução de sistemas NLA;

9. Ao final de integração numérica, obtém-se a lista de todos os eventos discretos associados ao instante de tempo t_i;

- Se houverem eventos de amostragem, atualiza-se os valores das amostras das variáveis de cada evento de amostragem;
- Se houverem eventos de avanço de instante discreto, atualiza-se o passado de todas as variáveis discretas dos eventos de avanço, calcula-se os valores atuais destas variáveis através da solução do subsistema D e, por fim, reinicializa-se a parte contínua do sistema através da solução do subsistema B. Ambas as soluções são feitas por meio de um algoritmo de solução de sistemas NLA.

10. Define-se o instante t_i como sendo o próximo instante de tempo da lista ordenada de instantes discretos;

11. Se o instante atual t for menor que t_f, retorna-se ao passo 8, caso contrário, finaliza-se a simulação.

O procedimento de monitoramento dos limites máximos e mínimos impostos às variáveis do modelo a ser simulado exige que se conheça as equações associadas a tais variáveis em cada subsistema do modelo. Caso alguma variável ao qual se atribuiu limites seja contínua, é preciso conhecer também as equações associadas as derivadas da variável em questão, já que estas também são impactadas pela limitação. Os valores das variáveis contínuas são obtidos por meio da solução dos subsistemas A e B, portanto, é preciso determinar as equações de A e de B que estão relacionadas às variáveis limitadas e suas derivadas. Tais relações são determinadas pelo acoplamento máximo, e também perfeito, do grafo bipartido que representa tais subsistemas. No caso de a variável limitada ser discreta, o problema é um pouco mais simples, bastando apenas conhecer a equação relacionada à variável limitada dentro do subsistema D. Vale lembrar que este processo de monitoramento é um pouco mais complexo do que parece, uma vez que é possível definir variáveis como limites inferior e superior de outras variáveis. O procedimento da lista 3.21 explica como é estruturado tal procedimento de monitoramento.

3.21 – Procedimento de solução numérica

1. Verifica-se se alguma variável x excedeu os limites superior ou inferior impostos a ela, caso houverem tais limites;

2. Se alguma variável x excedeu algum limite x_{lim} inferior ou superior, procede-se da seguinte maneira:

   - Cria-se uma cópia S_{copy} do subsistema S a qual se está solucionando;
   - Caso a variável x seja contínua e existam derivadas desta no subsistema S_{copy}, substitui-se em S_{copy} a equação relacionada a variável excedida x pela equação de limitação $x = x_{lim}$ e substitui-se as equações relacionadas as derivadas de x pelas equações de limitação $x^n = x^n_{lim}$;

- Caso a variável x seja discreta, substitui-se em S_{copy} a equação relacionada a variável excedida x pela equação de limitação $x[k] = x_{lim}[n]$, sendo n o instante discreto da variável x_{lim} utilizado quando se definiu o limite;
- Soluciona-se o sistema S_{copy} por meio de um algoritmo de solução de sistemas NLA;
- Substitui-se a solução do subsistema S pela solução do sistema S_{copy};
- Caso o subsistema S em questão for o subsistema B, reinicializa-se o integrador definindo como solução inicial os valores obtidos da solução do sistema S_{copy}.

3.4.2 Derivadas simbólico-numéricas e a tolerância de simulação

Existe um aspecto importante a se considerar quanto ao uso de derivadas simbólico-numéricas em modelos de processos. Retomando aqui a definição matemática da derivada simbólico-numérica, presente na subseção 3.1.7.2, tem-se as equações 3.11 e 3.12.

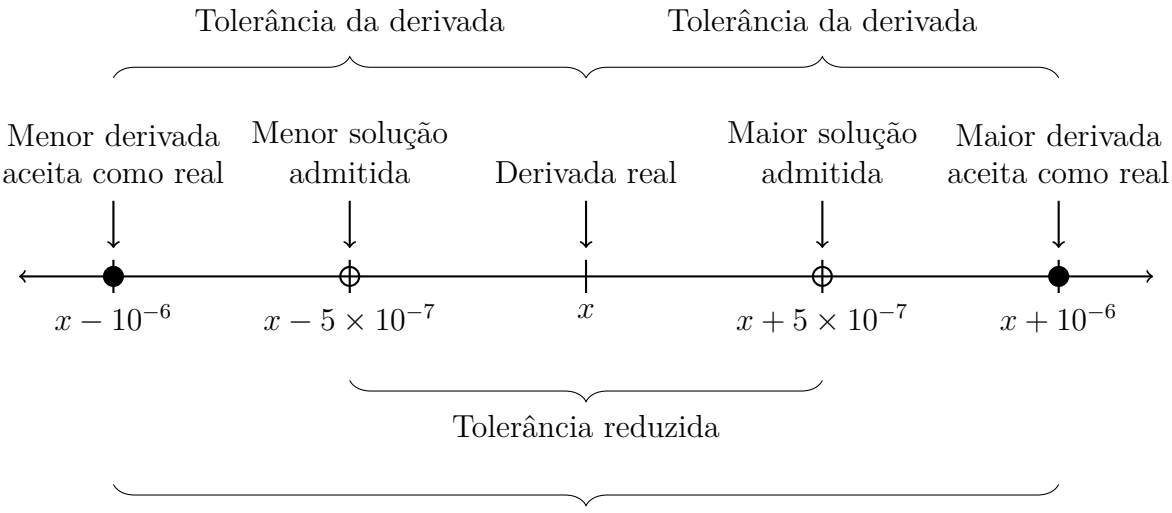
$$\frac{df(x)}{dt} = \sum_{i=1}^{n} \frac{\partial f(x)}{\partial x_i} \frac{dx_i}{dt} \tag{3.11}$$

$$\frac{\partial f(x)}{\partial x_i} = \lim_{h \to 0} \frac{f(x_0, ..., x_i + h, ..., x_n) - f(x)}{h} \tag{3.12}$$

Um vez que os algoritmos de simulação numérica de sistemas de equações utilizam um valor de tolerância para avaliar se uma possível solução numérica está suficientemente próxima da solução real, o cálculo do valor da derivada simbólico-numérica deve ser preciso o suficiente para não comprometer este processo de avaliação. Sabendo que qualquer valor de $h \neq 0$ introduziria erro na estimação da derivada, não seria possível respeitar uma tolerância de solução sem que se façam alguns ajustes. O primeiro ajuste necessário é transformar a equação que utiliza a derivada simbólico-numérica em duas equações, uma que é a equação original com expressão da derivada substituída por uma variável, e outra que é uma igualdade entre a variável de substituição e a expressão substituída. Por exemplo, $eq_1 : x^2 = y + g'(x, y)$ seria transformada no seguinte conjunto de equações: $eq_{1,1} : x^2 = y + z$ e $eq_{1,2} : z = g'(x, y)$. Isso é importante porque, para que se possa atender a uma tolerância de simulação, será necessária uma avaliação do resíduo da equação $eq_{1,2}$, ou seja, $resíduo = z - g'(x, y)$.

Supondo-se uma tolerância absoluta de simulação $tol = 10^{-6}$, um ajuste possível seria diminuí-la para a metade, $tol_2 = 5 \times 10^{-7}$ e utilizar esta tolerância reduzida tanto para o algoritmo de simulação quanto para o cálculo do valor da derivada. A Figura 24 mostra que mesmo nas piores situações possíveis, onde a solução do sistema foi aceita no limite da tolerância reduzida de simulação, mesmo que a derivada também tenha sido aceita no limite de sua tolerância, o erro total somado ainda permanece dentro dos limites aceitáveis da tolerância original.

Figura 24 – Análise de tolerâncias para a solução numérica de um sistema de equações contendo derivada simbólico-numérica



Havendo definido um valor de tolerância para a derivação simbólico-numérica, uma maneira possível de se determinar um bom valor para h na Equação 3.3 é ir diminuindo-o até que a variação no valor da derivada seja menor que o valor da tolerância. Uma forma interessante de se fazer isso é supor um valor inicial h_0 qualquer, por exemplo a própria tolerância, e ir diminuindo-o segundo a Equação 3.13. Quando for encontrado um valor de h_i e h_{i+1} que satisfaça a Equação 3.14, a ferramenta pode guardar os valores de h_i e h_{i+1} e utilizá-los como chute inicial numa próxima vez em que for necessário calcular a derivada simbólico-numérica da função f.

$$h_{i+1} = \frac{h_i}{2} \tag{3.13}$$

$$\left| \frac{f(x_0, ..., x_i + h_i, ..., x_n) - f(x)}{h_i} - \frac{f(x_0, ..., x_i + h_{i+1}, ..., x_n) - f(x)}{h_{i+1}} \right| <= tol \tag{3.14}$$

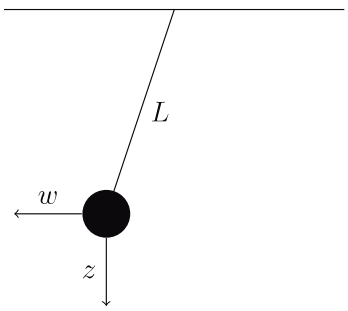
4 Validação da ferramenta

Nesta seção, será avaliado se a ferramenta desenvolvida de fato funciona conforme o esperado. Para isso, serão estudados alguns casos que servirão para comprovar, tanto a confiabilidade dos resultados numéricos, quanto os conceitos utilizados.

4.1 Validação de análise estrutural

O primeiro caso a ser verificado é se a ferramenta desenvolvida encontra de maneira correta o índice estrutural dos sistemas. Para isso, será avaliado o exemplo da Figura 3, apresentada na subseção 2.2.1 e replicado abaixo por conveniência.

Modelo DAE:	Modelo ODE:
$eq_1: \quad x' = w$	$eq_1'': \quad x''' = w''$
$eq_2: \quad y' = z$	$eq_2'': \quad y''' = z''$
$eq_3: \quad w' = T.x$	$eq_3': \quad w'' = T'.x + T.x'$
$eq_4: \quad z' = T.y - g$	$eq_4': \quad z'' = T'.y + T.y'$
$eq_5: \quad x^2 + y^2 = L^2$	$eq_5''': \quad x'''.x + x''.x' + x''.x' + x'.x'' + y'''.y + y''.y' + y''.y' + y'.y'' = 0$



Fonte: (PELEGRINI, 2007)

Os resultados de depuração produzidos pela ferramenta para o caso do pêndulo são apresentados nas Figuras 25 e 26. Na Figura 25, as variáveis g e L do modelo do pêndulo foram substituídas pelos valores $g = 9,8$ e $L = 1$. Além disso, para a construção do modelo reduzido, as variáveis x, w, y, z e T foram substituídas por $u1$, $u2$, $u3$, $u4$ e $u5$. Vê-se na Figura 26 que a ferramenta encontrou o valor correto para o índice estrutural ν_s já que, neste exemplo, o índice estrutural do pêndulo é igual ao seu índice diferencial. Outro resultado importante é que a ferramenta verificou a existência de duas condições iniciais para o pêndulo e explicitou os conjuntos de variáveis elegíveis a condições iniciais. Este resultado é consistente e está de acordo com o grafo da Figura 27, lembrando que as equações derivadas no grafo foram derivadas estruturalmente, Definição 2.3 da subseção 2.2.3.3.

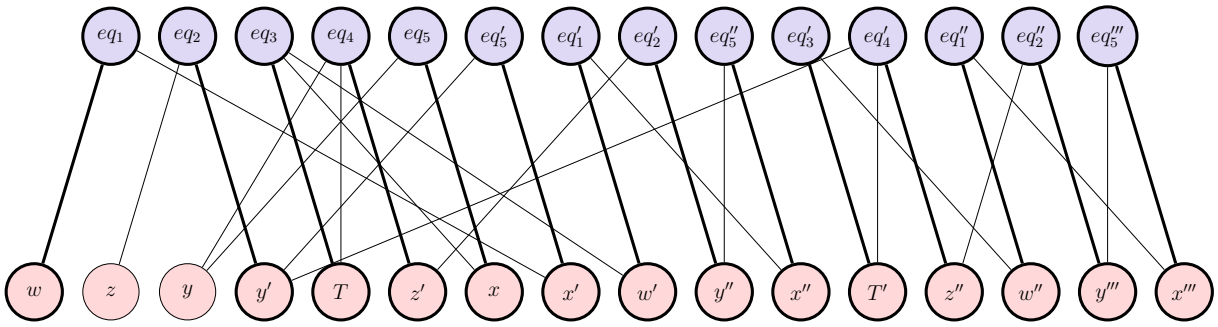
Figura 25 – Modelos produzidos pela ferramenta para o caso do pêndulo oscilante de índice diferencial $\nu_d = 3$

```
Original equation system: 5 equations, 9 variables
x' = w
y' = z
w' = T*x
z' = T*y - 9.8
x**2 + y**2 = 1

Reduced equation system: 5 equations, 9 variables
u1' = u2
u3' = u4
u2' = u5*u1
u4' = u5*u3 - 9.8
u1**2 + u3**2 = 1

Reduced index equation system: 14 equations, 16 variables
u1' = u2
u3' = u4
u2' = u5*u1
u4' = u5*u3 - 9.8
u1**2 + u3**2 = 1
2*u1'*u1 + 2*u3'*u3 = 0
u1'' = u2'
u3'' = u4'
2*u1'''*u1 + 2*u1''*u1' + 2*u3'''*u3 + 2*u3''*u3' = 0
u5''*u1 + u5*u1' = u2''
u5''*u3 + u5*u3' = u4''
u1''' = u2''
u3''' = u4''
2*u1''''*u1 + 2*u1'''*u1' + 2*u1'''*u1' + 2*u1''*u1'' + 2*u3''''*u3 + 2*u3'''*u3' + 2*u3'''*u3' + 2*u3''*u3'' = 0
```

Figura 26 – Resultados de depuração produzidos pela ferramenta para o caso do pêndulo oscilante de índice diferencial $\nu_d = 3$

```
Structural index: 3
Extra equation: 9
Extra variables: 7
Overconstrained subsystems: 0
Underconstrained subsystems: 0
Number of initial conditions: 2
Possible initial conditions:
-> {y, z', y'', x'', w', T}
-> {z, y', x', w, z'', y''', x''', w''', T'}
```

Figura 27 – Grafo do sistema de equações do pêndulo oscilante de índice diferencial $\nu_d = 3$ com um de seus acoplamentos máximos destacados



4.2 Validação de resultados numéricos

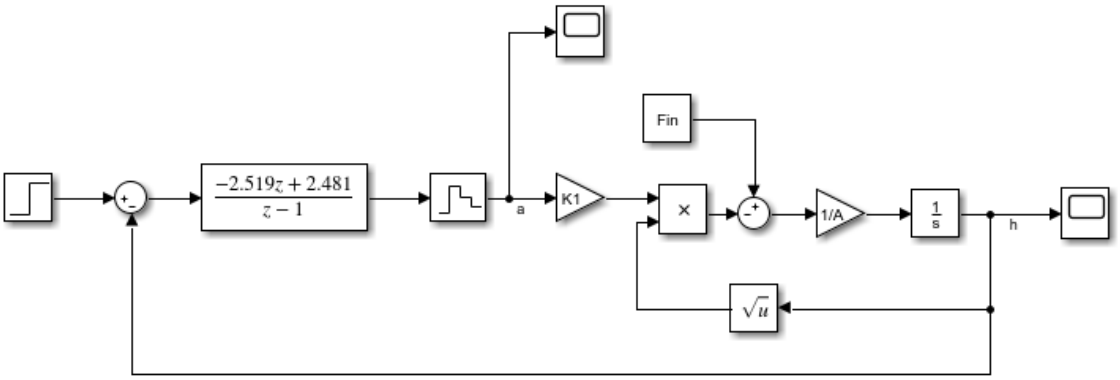
Para a validação de resultados numéricos produzidos pela ferramenta, será estudado o caso de um tanque atmosférico cilíndrico para líquidos com controle discreto de nível. O modelo matemático deste caso encontra-se representado no sistema 4.1. Neste exemplo, h representa o nível do tanque, A representa a área da base do tanque, F_{in} representa o fluxo de entrada de fluido no tanque, F_{out} representa o fluxo de saída, a representa o sinal de abertura de uma válvula linear e instantânea posicionada na saída do tanque e K_1 é uma constante que depende da área da seção transversal do canal de saído do tanque, da gravidade e da massa específica do fluido.

$$\begin{aligned} A.h' &= F_{in} - F_{out} & \\ F_{out} &= K_1.a.\sqrt{h} & u[k] = u[k-1] - 2,519.e[k] + 2,481.e[k-1] \\ a &= u[k] & e[k] = r - s_h \end{aligned} \tag{4.1}$$

Modelo do tanque atmosférico — Modelo do controlador discreto

Neste exemplo, são considerados constante $F_{in} = 0,2$, $K_1 = 0,4$, $A = 1$ e $r = 1$. A variável s_h é a amostragem do sinal h, enquanto $a = u[k]$ representa a atuação do controlador na abertura da válvula por meio de um segurador de ordem zero. O período de amostragem do controlador discreto é $Ts = 0.15s$ e a condição inicial definida para o nível inicial do tanque é $h_0 = 0$. O modelo implementado na plataforma *Simulink* do software MATLAB encontra-se representado na Figura 28.

Figura 28 – Modelo do sistema 4.1 implementado no ambiênte *Simulink* do software de simulação MATLAB



A Figura 29 apresenta a resposta temporal da simulação do sistema 4.1 no ambiente *Simulink*. Já a Figura 30 apresenta o mesmo ensaio, porém realizado na ferramenta de simulação desenvolvida.

Figura 29 – Ensaio do sistema 4.1 realizado no ambiênte *Simulink* do software de simulação MATLAB

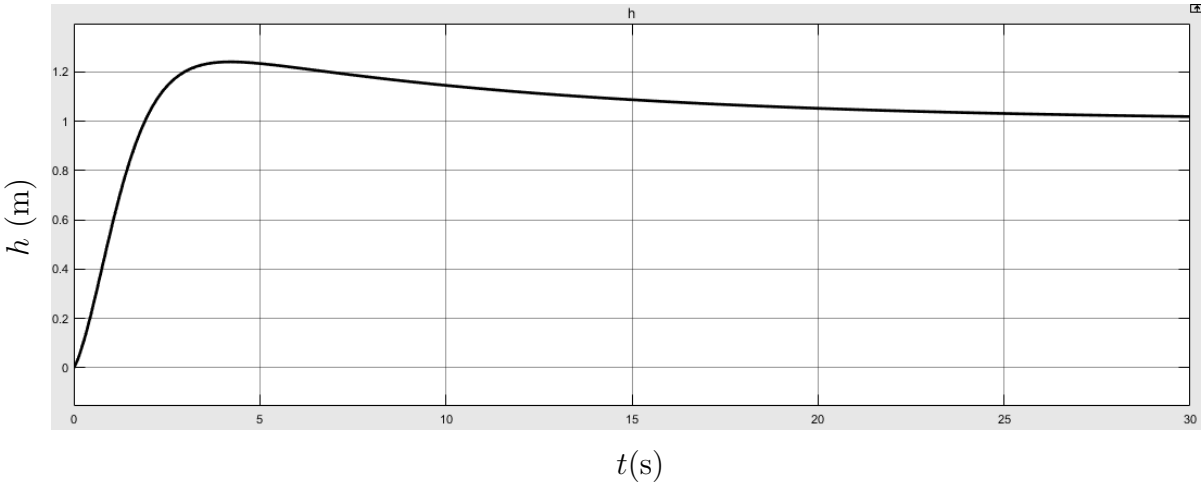
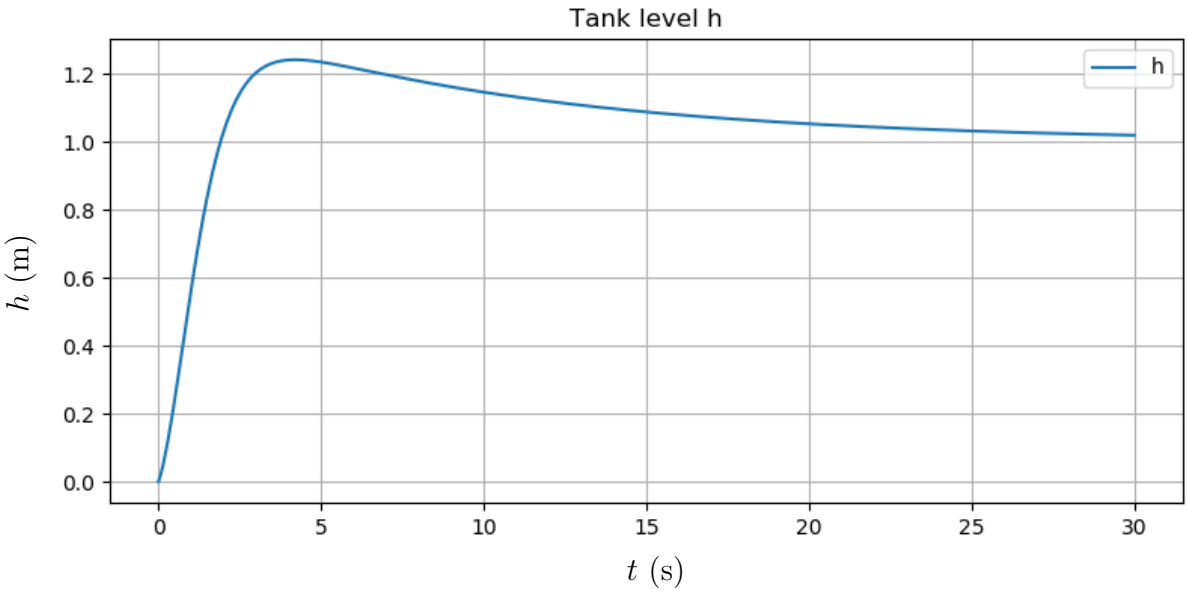


Figura 30 – Ensaio do sistema 4.1 realizado na ferramenta de modelagem e simulação



Comparando-se as figuras 29 e 30, vê-se que ambas alcançaram os mesmos resultados de simulação. Uma vez que o ambiente de simulação *Simulink* é tido como uma ferramenta confiável e consolidada na área de simulação de sistemas dinâmicos, a produção de resultados idênticos por parte da ferramenta desenvolvida serve como validação de seus algoritmos de simulação numérica.

5 Conclusões e perspectivas

Como visto em todo o relatório até aqui, existem muitos detalhes a serem avaliados durante a concepção de uma ferramenta de modelagem e simulação de sistemas dinâmicos. O processo se torna ainda mais complexo quando passa a considerar a possibilidade de modelos matemáticos de natureza não-contínua.

Toda a reformulação feita para a ferramenta desenvolvida como projeto de estágio em Garcez (2019) foi motivada pelos objetivos apresentados no capítulo de introdução, reproduzidos na lista 5.1 por conveniência.

5.1 – Lista de objetivos específicos do projeto de reformulação da ferramenta de modelagem e simulação de sistemas dinâmicos

1. Aumentar o poder de representação e simulação de sistemas por meio de suporte à modelagem de fenômenos contínuos, discretos e booleanos;
2. Aplicar metodologias de análise e depuração de sistemas de equações matemáticas afim de auxiliar na modelagem e definições de processos e equipamentos;
3. Desenvolver todo um ferramental de matemática simbólica para não só melhorar a inserção de modelos na ferramenta, mas também prover métodos de derivação simbólica para solucionar problemas ligados a simulação;
4. Ajustar detalhes de uso da ferramenta conforme a necessidade dos usuários.

Nem todos os objetivos estipulados para o projeto foram alcançados com exatidão. Viu-se na seção 3.4 que a ferramenta desenvolvida ainda não é capaz de resolver numericamente sistemas que envolvam equações booleanas. Tal limitação fere parcialmente o primeiro objetivo específico da lista 5.1. Quanto aos objetivos 2 e 3, estes foram plenamente alcançados, o que configura um grande êxito deste trabalho, uma vez que tais metas aparecem como solução dos problemas de confiabilidade que a versão anterior da ferramenta apresentava. Quanto ao último objetivo específico, não houve tempo hábil até então para que se submetesse a ferramenta a testes de uso.

Pode-se concluir que, embora alguns objetivos não tenham sidos completamente alcançados, o que se conseguiu com a reformulação da ferramenta é bastante satisfatória e certamente cumpre a proposta geral deste trabalho. Para trabalhos futuros, sugere-se o refinamento do protocolo de comunicação com ferramentas externas abordado na subseção 3.2.3.1 e a implementação de técnicas de solução de sistemas híbridos com equações booleanas.

Referências

ABDULLA, T. J.; CASH, J. R.; DIAMANTAKIS, M. T. An mebdf package for the numerical solution of large sparse systems of stiff initial value problems. *Computers Mathematics with Applications*, v. 42, n. 1-2, p. 121–129, 7 2001. Citado na página 22.

BERGE, C. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, v. 43, p. 842–844, 1957. Citado na página 30.

BRENAN, K. E.; CAMPBEL, S. L.; PETZOLD, L. R. *Numerical solution of initial-value problem in differential-algebraic equations*. New York: North-Holland, 1989. ISBN 978-0-89871-353-4. Citado 2 vezes nas páginas 21 e 22.

Costa Jr., E. F. da. *Resolução automática de equações algébrico-diferenciais de índice superior*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, 3 2003. Citado na página 64.

DEUFLHARD, P.; HAIRER, E.; ZUGCK, J. One-step and extrapolation methods for differential-algebraic systems. *Numerische Mathematik*, v. 51, n. 5, p. 501–516, 9 1987. Citado na página 22.

ENGENHARIA de software. In: WIKIPÉDIA: a enciclopédia livre. Wikimedia, 2019. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Engenharia_de_software&oldid=55875911>. Acesso em: 01 de agosto de 2019. Citado na página 15.

EQUAÇÃO diferencial. In: WIKIPÉDIA: a enciclopédia livre. Wikimedia, 2019. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Equa%C3%A7%C3%A3o_diferencial&oldid=55881324>. Acesso em: 01 de agosto de 2019. Citado na página 18.

GARCEZ, F. A. *Desenvolvimento de um ambiente de modelagem e simulação dinâmica de processos e equipamentos*. Florianópolis, SC, 2019. Relatório de estágio. Citado 3 vezes nas páginas 12, 15 e 113.

GARCIA, C. *Modelagem e simulação de processos industriais e de sistemas eletromecânicos*. 2. ed. São Paulo, SP: Edusp, 2005. ISBN 9788531409042. Citado na página 18.

HAIRER, E.; WANNER, G. Stiff differential equations solved by radau methods. *Journal of Computational and Applied Mathematics*, v. 111, n. 1-2, p. 93–111, 11 1999. Citado na página 22.

HINDMARSH, A. C. LSODE and LSODI, two new initial value ordinary differnetial equation solvers. *Association for Computing Machinery*, v. 15, n. 4, p. 10–11, 12 1980. Citado na página 22.

KUROSE, J. F.; ROSS, K. W. *Computer Network: A Top-Down Approach Featuring the Internet*. 6. ed. University of Massachusetts, Amherst: Pearson, 2000. ISBN 978-0-13-285620-1. Citado na página 81.

LIOEN, W.; SWART de J.; VEEN, W. van der. Pside user's guide. 1998. Citado na página 22.

MOCELIN, K. *Melhores práticas de Software*. 2011. Acessado pela última vez em 31 de Julho de 2019. Disponível em: <https://kauanmocelin.wordpress.com/2011/11/29/melhores-praticas-de-software/>. Citado na página 16.

PANTELIDES, C. C. The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific and Statistical Computing*, v. 9, n. 2, p. 213–231, 1988. Citado 2 vezes nas páginas 36 e 94.

PELEGRINI, R. *Depuração para simuladores de processos baseados em equações*. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, 2007. Citado 13 vezes nas páginas 9, 19, 20, 22, 23, 32, 36, 37, 38, 40, 87, 94 e 108.

PETZOLD, L. R. Description of DASSL: a differential/algebraic system solver. In: SANDIA NATIONAL LABS. Livermore, California, 1982. Citado na página 22.

REISSIG, G.; MARTINSON, W. S.; BARTON, P. I. Differential–algebraic equations of index 1 may have an arbitrarily high structural index. *SIAM Journal on Scientific Computing*, v. 21, n. 6, p. 1987–1990, 2000. Citado na página 37.

UNGER, J.; KRöNER, A.; MARQUARDT, W. Structural analysis of differential-algebraic equation systems - theory and applications. *Computers Chemical Engineering*, v. 19, n. 8, p. 867–882, 8 1995. Citado na página 37.

WESTERBERG, A. W. *Process Engineering: Part II - A Systems View*. 1998. Citado na página 12.

ÁRVORE (estrutura de dados). In: WIKIPÉDIA: a enciclopédia livre. Wikimedia, 2019. Disponível em: <https://pt.wikipedia.org/w/index.php?title=%C3%81rvore_(estrutura_de_dados)&oldid=49662694>. Acesso em: 01 de agosto de 2019. Citado na página 16.