UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE
LABORATÓRIO DE ENGENHARIA E EXPLORAÇÃO DE PETRÓLEO

PROJETO ENGENHARIA
DESENVOLVIMENTO DO SOFTWARE
SIMULADOR DE PROPRIEDADES TERMODINÂMICAS DE SUBSTÂNCIAS PURAS A PARTIR DE DENSIDADE E TEMPERATURA
TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA

THÔMAS AUGUSTO STEMPKOWSKI MENEGAZZO
PROFESSOR ANDRÉ DUARTE BUENO (SOFTWARE)
PROFESSOR ADOLFO PUIME PIRES (TERMODINÂMICA)

MACAÉ - RJ
MARÇO - 2015

Sumário

Capítulo 1

Introdução

No presente projeto de engenharia desenvolve-se o software SPTSP(Simulador de Propriedades Termodinâmica de Substâncias Puras), um software aplicado a engenharia de petróleo e que utiliza o paradigma da orientação a objetos.

Neste trabalho é implementada uma equação experimental, baseada na energia livre de Helmholtz, para substâncias puras obtidas a partir da literatura. Por meio desta equação, poderão ser calculadas várias propriedades termodinâmicas do fluido a partir de dados de temperatura e pressão.

1.1 Escopo do problema

A correta previsão das propriedades termodinâmicas e cálculo do equilíbrio de fases constitui-se num ponto fundamental do projeto de qualquer processo da industria química, como sistemas de refrigeração. A exploração e produção de reservas naturais de hidrocarbonetos não foge a essa regra, com a dificuldade adicional de tratar complexas misturas de hidrocarbonetos, muitas vezes não bem caracterizadas, além da presença de compostos não orgânicos, como água e dióxido de carbono, empregados em técnicas para aumentar o fator de recuperação das jazidas.

Na simulação numérica computacional, que usa o modelo composicional de reservatórios de hidrocarbonetos, um dos conjuntos de equações utilizado para a solução é composto pelos coeficientes de distribuição dos componentes entre as fases presentes no meio poroso. Em varios momentos da simulação uma ou mais fases podem ser constituídas de compostos puros. Nesse caso, ao invés de utilizar equações de estado, são utilizadas correlações para prever as propriedades termodinâmicas, bem como as suas derivadas (necessárias para o cálculo do jacobiano do método de Newton).

1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:
  - Deselvolver um software que calcula as propriedades termodinâmicas de uma substância pura a partir de equações experimentais.
- Objetivos específicos do software incluem:

- Unir as diversas equações experimentais em um programa único.
- Deselvolver software com interface gráfica amigável.

Capítulo 2

Especificação

Apresenta-se neste capítulo do projeto de engenharia a concepção, a especificação do sistema a ser modelado e desenvolvido.

2.1 Especificação do programa - descrição dos requisitos

- O projeto a ser desenvolvido consiste em um software que pede qual substância será calculada, sua densidade e temperatura. O programa então pedirá quais propriedades o usuário deseja calcular, se ele deseja as derivadas das propriedades, e quais derivadas. Com isso ele calculará a energia livre de Helmholtz e as derivadas desta necessárias para o cálculo das propriedades escolhidas. Através de relações termodinâmicas as propriedades e suas derivadas serão calculadas, e retornadas ao usuário.
- O software será desenvolvido utilizando o conceito de programação orientada a objeto.
- Será feita uma interface em modo texto e uma interface gráfica amigável.

2.2 Especificação do software - requisitos

2.2.1 Nome do sistema/produto e componentes

Nome	SPTSP
Componentes principais	Sistema para cálculo das propriedades termodinâmicas, interface gráfica.
Missão	Cálculo de propriedades termodinâmicas de substâncias puras.

2.2.2 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

RF-01	O usuário deverá ter liberdade para escolher o que deseja calcular.

RF-02	Deve permitir o carregamento de arquivos criados pelo software.

2.2.3 Requisitos não funcionais

RNF-01	O programa deverá ser multi-plataforma, podendo ser executado em *Windows*, *GNU/Linux* ou *Mac OS X*.

2.3 Casos de uso do software

Apresenta-se na tabela 2.1 um exemplo de caso de uso geral.

Tabela 2.1: Exemplo de caso de uso

Nome do caso de uso:	Cálculo de uma propriedade.
Resumo/descrição:	Cálculo de uma propriedade a partir da temperatura e densidade de uma substância.
Etapas:	1. Criar objeto SPTSP. 2. Entrar com temperatura, densidade e substância. 3. Escolher a propriedade a ser calculada. 4. Calcular propriedade. 5. Analisar resultados.
Cenários alternativos:	Um cenário alternativo envolve o cálculo da derivada de uma propriedade.

2.4 O que são os diagramas de casos de uso?

O diagrama de casos de uso é uma representação visual dos casos de uso. É o diagrama mais simples da UML, sendo utilizado para demonstrar os cenários de uso do sistema pelos usuários, os quais ao verem esses diagramas terão uma visão geral do sistema.

O diagrama de caso de uso pode ser utilizado durante e após a etapa de especificação, adicionando às especificações textuais alguns cenários de uso do software a ser desenvolvido.

2.5 Diagrama de caso de uso geral do software

O diagrama de caso de uso geral da Figura 2.1 mostra o usuário entrando com os dados, calculando propriedades e/ou derivadas, e analisando resultados.

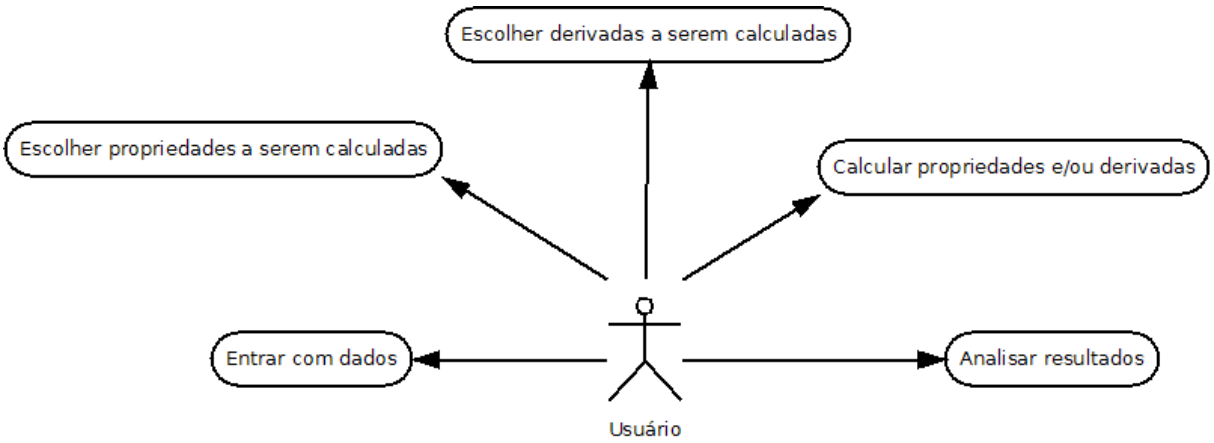

Figura 2.1: Diagrama de caso de uso – Caso de uso geral

Chapter 3

Elaboração

Depois da definição dos objetivos, da especificação do software e da montagem dos primeiros diagramas de caso de uso, a equipe de desenvolvimento do projeto de engenharia passa por um processo de elaboração que envolve o estudo de conceitos relacionados ao sistema a ser desenvolvido, a análise de domínio e a identificação de pacotes.

Na elaboração fazemos uma análise dos requisitos, ajustando os requisitos iniciais de forma a desenvolver um sistema útil, que atenda às necessidades do usuário e, na medida do possível, permita seu reuso e futura extensão.

Eliminam-se os requisitos "impossíveis" e ajusta-se a idéia do sistema de forma que este seja flexível, considerando-se aspectos como custos e prazos.

3.1 Análise de domínio

- Áreas relacionadas:

  Uma área relacionada ao desenvolvimento deste software é a Engenharia de Reservatório. É a área da engenharia de petróleo que trata da descrição do comportamento do óleo, gás e água no interior dos reservatórios de hidrocarbonetos, essencial para o desenvolvimento de campos de petróleo. Engenheiros de reservatório usam dados sobre as propriedades dos fluidos e rochas no reservátorio, assim como o histórico deste para prever seu comportamento futuro.

  Outra área relacionada é a da modelagem numérica computacional, que deselvolve modelos matemáticos para a solução de um determinado problema físico e então deselvolve o modelo computacional por meio de algoritmos para encontrar a solução deste problema por meio de um computador.

- Sub-áreas relacionadas:

  A simulação de reservatórios constitui uma sub-área na engenharia de reservatório.
  Ela se trata da utilização do desenvolvimento de simuladores de reservatório, que por meio de modelos matemáticos buscam prever o comportamento de um reservátorio de petróleo e de seus poços associados. Os simuladores podem ser do tipo *black oil* ou composicionais, no primeiro o óleo é considerado uma substância só, e no segundo uma mistura de diversas substâncias.

  A termodinâmica é uma área da física que estuda os efeitos de mudanças na temperatura, pressão, volume e outras propriedades termodinâmicas de um sistema. Ela é extremamente importante no desenvolvimento de um simulador de reservatório pois os fluidos de um reservatório sofrem diversas alterações físicas durante sua produção, sendo necessária uma boa modelagem termodinâmica para entender como eles reagirão a estas alterações.

3.2 Formulação teórica

O software calcula as propriedades termodinâmicas de uma substância. Para isso será usada uma equação experimental para calcular a energia livre de Helmholtz e suas derivadas. Com estas é possivel calcular todas as propriedades através de relações termodinâmicas apresentadas a seguir.

A energia livre de Helmholtz é um potencial termodinâmico que mede o trabalho útil obtido a partir de um sistema termodinâmico fechado à temperatura constante. Ela é definida como:

$$f(\rho, T) = h(T) - RT - Ts(\rho, T) \tag{3.1}$$

Esta energia será modelada através da sua forma adimensional, a dividindo em uma parte ideal ϕ^o e uma parte residual ϕ^r:

$$\frac{f(\delta, \tau)}{RT} = \phi(\delta, \tau) = \phi^o(\delta, \tau) + \phi^r(\delta, \tau) \tag{3.2}$$

Sendo

$$\delta = \frac{\rho}{\rho_c}$$

a densidade reduzida para uma densidade crítica $\rho_c \left[\frac{kg}{m^3}\right]$e

$$\tau = \frac{T_c}{T}$$

a temperatura reduzida para uma temperatura crítica $T_c[K]$.

A parte ideal ϕ^o é representada pela equação:

$$\phi^o = \ln \delta + n_1^o + n_2^o \tau + n_3^o \ln \tau + \sum_{i=4}^{8} n_i^o \ln[1 - e^{-\gamma_i^o \tau}], \tag{3.3}$$

E a parte residual ϕ^r pela soma de quatro partes:

$$\phi^r = \phi_1^r + \phi_2^r + \phi_3^r + \phi_4^r \tag{3.4}$$

Parte polinomial

$$\phi_1^r = \sum_{i=1}^{m} n_i \delta^{d_i} \tau^{t_i} \tag{3.5}$$

Parte exponencial

$$\phi_2^r = \sum_{i=1}^{n} n_i \delta^{d_i} \tau^{t_i} e^{-\delta^{c_i}} \tag{3.6}$$

Parte de curva Gaussiana

$$\phi_3^r = \sum_{i=52}^{r} n_i \delta^{d_i} \tau^{t_i} e^{-a_i(\delta - \epsilon_i)^2 - \beta_i(\tau - \gamma_i)^2} \tag{3.7}$$

Parte não-analítica

$$\phi_4^r = \sum_{i=1}^{s} \triangle^{b_i} \delta \psi \tag{3.8}$$

com

$$\triangle = \theta^2 + B_i[(\delta - 1)^2]^{a_i}$$

$$\theta = (1-\tau) + A_i[(\delta,-1)^2]^{1/(2\beta_i)}$$

$$\psi = e^{-C_i(\delta-1)^2 - D_i(\tau-1)^2}$$

Sendo os coeficientes com subíndice i constantes adquiridas experimentalmente, obtidas na literatura [Bucker and Wagner, 2006b, Bucker and Wagner, 2006a] [Setzmann and Wagner, 1991, Span and Wagner, 1996, Wagner and Prub, 2002, Span et al., 2000] para cada substância.

Com estas equações é possível obter diversas propriedades termodinâmicas, a partir das relações a seguir:

Pressão$[Bar]$:

$$\frac{p(\delta,\tau)}{\rho RT} = 1 + \delta\phi^r_\delta \tag{3.9}$$

Entropia$\left[\frac{kJ}{kg.K}\right]$:

$$\frac{s(\delta,\tau)}{R} = \tau(\phi^o_\tau + \phi^{rM=}_\tau) - \phi^o - \phi^r \tag{3.10}$$

Energia Interna$\left[\frac{kJ}{kg}\right]$:

$$\frac{u(\delta,\tau)}{RT} = \tau(\phi^o_\tau + \phi^r_\tau) \tag{3.11}$$

Entalpia$\left[\frac{kJ}{kg}\right]$:

$$\frac{h(\delta,\tau)}{RT} = 1 + \tau(\phi^o_\tau + \phi^r_\tau) + \delta\phi^r_\delta \tag{3.12}$$

Energia livre de Gibbs$\left[\frac{kJ}{kg}\right]$:

$$\frac{g(\delta,\tau)}{RT} = 1 + \phi^o + \phi^r + \delta\phi^r_\delta \tag{3.13}$$

Capacidade calorífica isocórica $\left[\frac{kJ}{kg.K}\right]$:

$$\frac{c_v(\delta,\tau)}{R} = -\tau^2(\phi^o_{\tau\tau} + \phi^r_{\tau\tau}) \tag{3.14}$$

Capacidade calorífica isobárica $\left[\frac{kJ}{kg.K}\right]$:

$$\frac{c_p(\delta,\tau)}{R} = -\tau^2(\phi^o_{\tau\tau} + \phi^r_{\tau\tau}) + \frac{(1+\delta\phi^r_\delta - \delta\tau\phi^r_{\delta\tau})^2}{1 + 2\delta\phi^r_\delta + \delta^2\phi^r_{\delta\delta}} \tag{3.15}$$

Velocidade do som$\left[\frac{m}{s}\right]$:

$$\frac{w^2(\delta,\tau)}{RT} = 1 + 2\delta\phi^r_\delta + \delta^2\phi^r_{\delta\delta} - \frac{(1+\delta\phi^r_\delta - \delta\tau\phi^r_{\delta\tau})^2}{\tau^2(\phi^o_{\tau\tau} + \phi^r_{\tau\tau})} \tag{3.16}$$

Coeficiente de Joule-Thomson$\left[\frac{K}{MPa}\right]$:

$$\mu R\rho = \frac{-(\delta\phi^r_\delta + \delta^2\phi^r_{\delta\delta} + \delta\tau\phi^r_{\delta\tau})}{(1+\delta\phi^r_\delta - \delta\tau\phi^r_{\delta\tau})^2 - \tau^2(\phi^o_{\tau\tau} + \phi^r_{\tau\tau})(1 + 2\delta\phi^r_\delta + \delta^2\phi^r_{\delta\delta})} \tag{3.17}$$

Coeficiente de enforcamento isotérmico$\left[\frac{kJ}{kg\cdot MPa}\right]$:

$$\delta_T \rho = \frac{1 + \delta\phi_\delta^r - \delta\tau\phi_{\delta\tau}^r}{1 + 2\delta\phi_\delta^r + \delta^2\phi_{\delta\delta}^r} \tag{3.18}$$

Coeficiente de temperatura-pressão isentrópico $\left[\frac{K}{MPa}\right]$:

$$\beta_s \rho R = \frac{1 + \delta\phi_\delta^r - \delta\tau\phi_{\delta\tau}^r}{(1 + \delta\phi_\delta^r - \delta\tau\phi_{\delta\tau}^r)^2 - \tau^2(\phi_{\tau\tau}^o + \phi_{\tau\tau}^r)(1 + 2\delta\phi_\delta^r + \delta^2\phi_{\delta\delta}^r)} \tag{3.19}$$

3.3 Identificação de pacotes – assuntos

- Pacote Substância: Esse pacote armazena os dados de uma substância, e possui métodos para mostrá-los ou gravá-los.
- Pacote Helmholtz: Esse pacote contêm os coeficientes para a equação experimental de cada substância, e os metodos de cálculo da energia livre de Helmholtz e suas derivadas.
- Pacote Simulador: Gerencia a simulação, tendo métodos que usam o pacote Helmholtz para calcular as propriedades do pacote Substância.

3.4 O que é o diagrama de pacotes – assuntos

Um diagrama de pacotes é útil para mostrar as dependências entre as diversas partes do sistema; pode incluir: sistemas, subsistemas, hierarquias de classes, classes, interfaces, componentes, nós, colaborações e casos de uso.

3.5 Diagrama de pacotes – assuntos

O diagrama de pacotes da Figura 3.1 mostra a relação entre os diferentes pacotes do software desenvolvido.

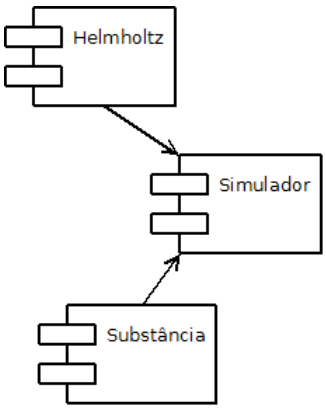

Figure 3.1: Diagrama de Pacotes

Capítulo 4

AOO – Análise Orientada a Objeto

A terceira etapa do desenvolvimento de um projeto de engenharia, no nosso caso um software aplicado a engenharia de petróleo, é a AOO – Análise Orientada a Objeto. A AOO utiliza algumas regras para identificar os objetos de interesse, as relacões entre as classes, os atributos, os métodos, as heranças, as associações, as agregações, as composições e as dependências.

O modelo de análise deve ser conciso, simplificado e deve mostrar o que deve ser feito, não se preocupando como isso será realizado.

O resultado da análise é um conjunto de diagramas que identificam os objetos e seus relacionamentos.

4.1 Diagramas de classes

O diagrama de classes é apresentado na Figura 4.1, e detalhado nas Figuras 4.2 e 4.3. O software foi dividido em três classes principais: Uma para armazenar os dados, CSubstância; Uma para calcular a energia de Helmholtz, CHelmholtz, que possui classes herdeiras devido a alguns valores usados nos calculos mudarem para diferentes substâncias; E uma para cálculo das propriedades, CSimuladorPropriedades;

4.1.1 Dicionário de classes

A descrição detalhada dos metodos e atributos das diferentes classes está no arquivo em .html incluído junto com as listagens.

4.2 Diagrama de seqüência – eventos e mensagens

O diagrama de seqüência enfatiza a troca de eventos e mensagens e sua ordem temporal. Contém informações sobre o fluxo de controle do software. Costuma ser montado a partir de um diagrama de caso de uso e estabelece o relacionamento dos atores (usuários e sistemas externos) com alguns objetos do sistema.

4.2.1 Diagrama de sequência geral

Veja o diagrama de seqüência na Figura 4.4.

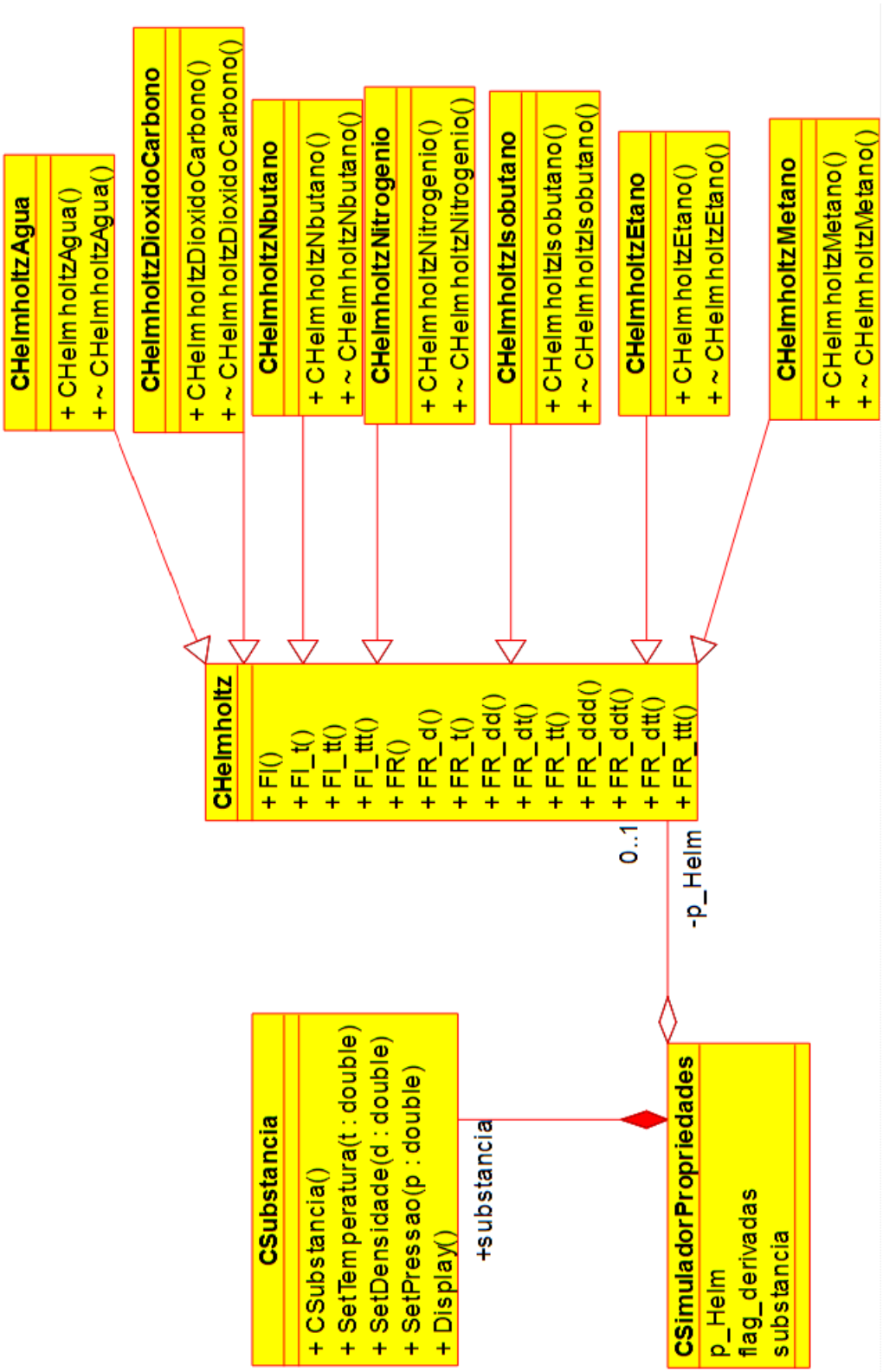

Figura 4.1: Diagrama de classes

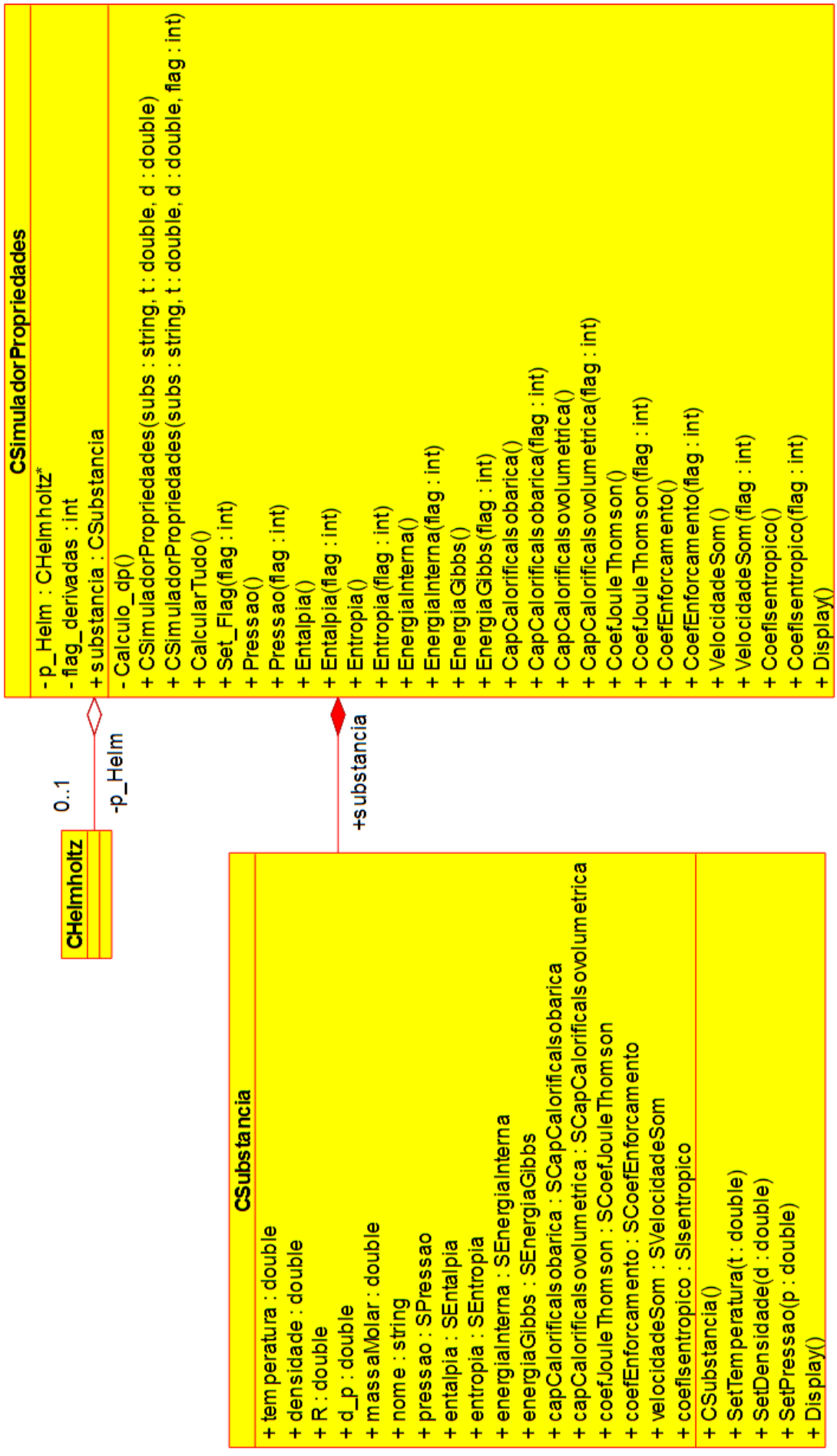

Figura 4.2: Diagrama de classes detalhado 1

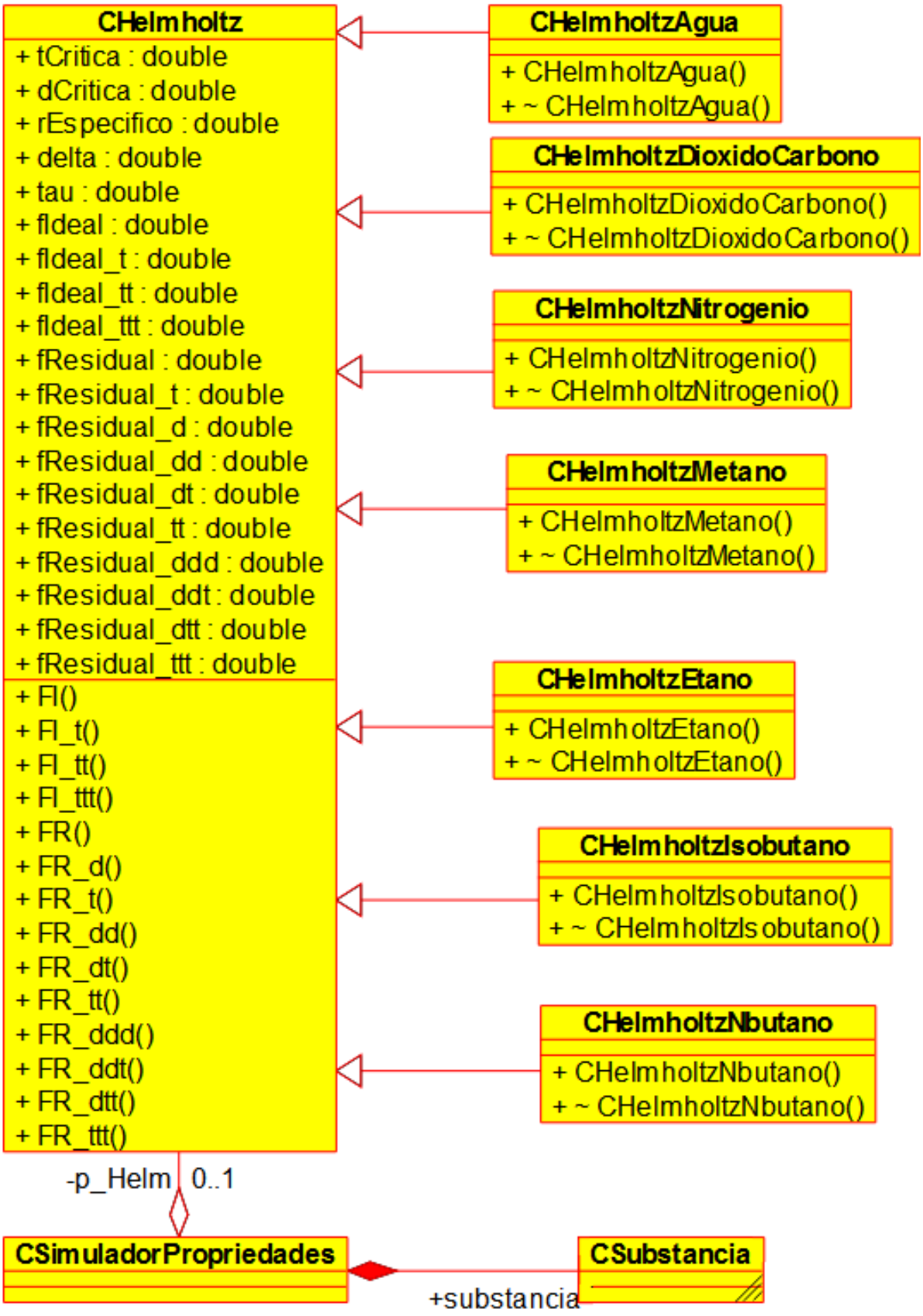

Figura 4.3: Diagrama de classes detalhado 2

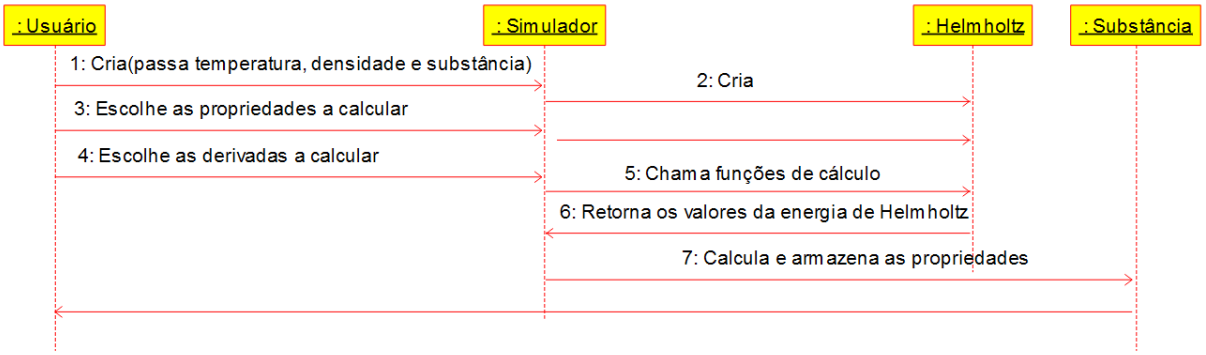

Figura 4.4: Diagrama de seqüência geral

4.3 Diagrama de comunicação – colaboração

No diagrama de comunicação o foco é a interação e a troca de mensagens e dados entre os objetos.

Veja na Figura 4.3 o diagrama de comunicação mostrando a sequência de cálculo. Observe que dependendo das propriedades escolhidas o objeto Simulador irá cria objetos Helmholtz e pedir dados diferentes.

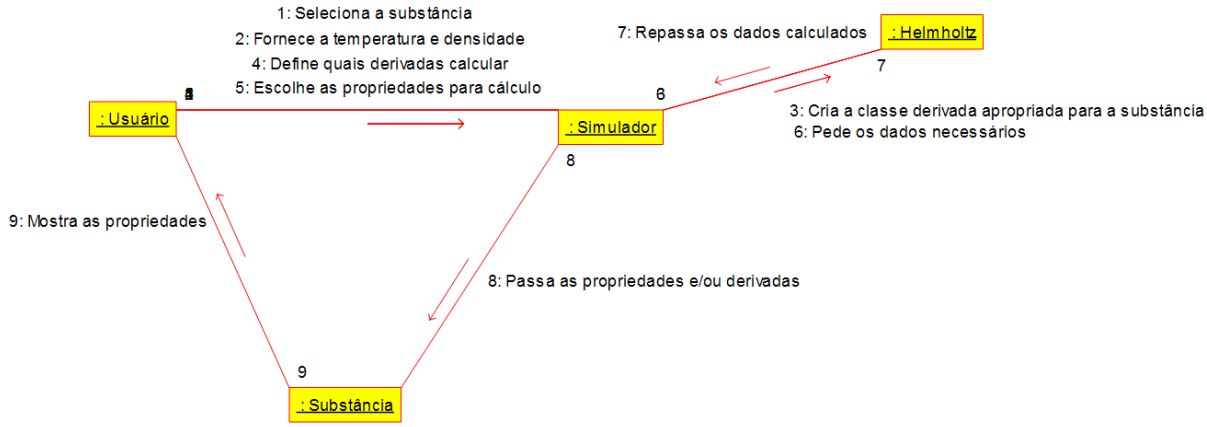

Figura 4.5: Diagrama de comunicação

4.4 Diagrama de máquina de estado

Um diagrama de máquina de estado representa os diversos estados que um objeto assume e os eventos que ocorrem ao longo de sua vida ou mesmo ao longo de um processo (histórico do objeto). É usado para modelar aspectos dinâmicos do objeto.

Veja na Figura 4.6 o diagrama de máquina de estado para o objeto Helmholtz. Observe que ele apenas é acessado indiretamente, por meio de CSimulador.

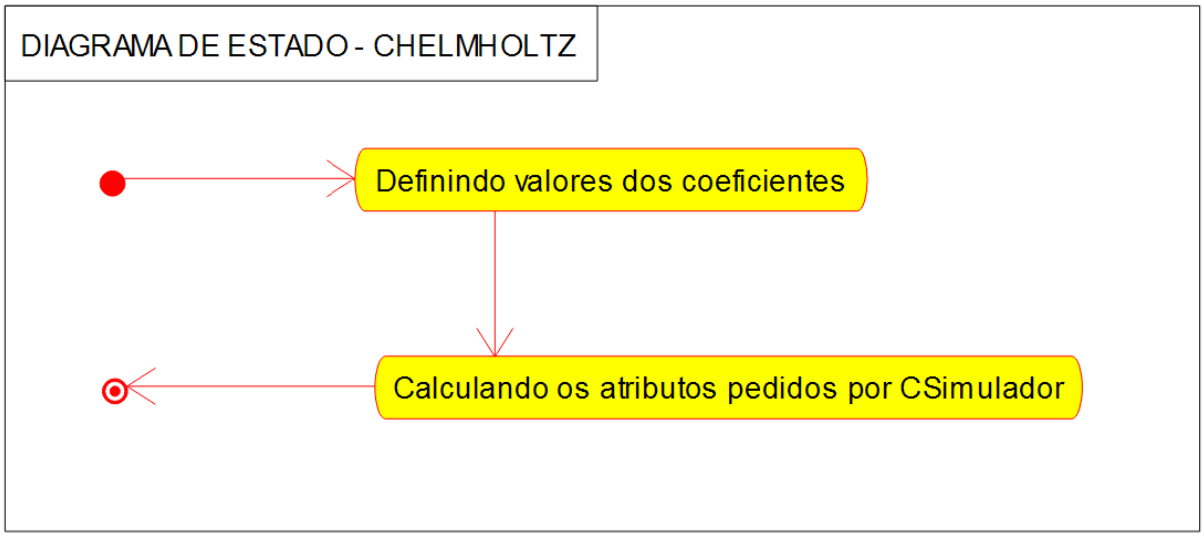

Figura 4.6: Diagrama de máquina de estado: Classe CHelmholtz

4.5 Diagrama de atividades

Veja na Figura 4.7 o diagrama de atividades correspondente a uma atividade específica do diagrama de máguina de estado. Observe que dependendo do valor de flag_derivadas as funções chamadas são diferentes.

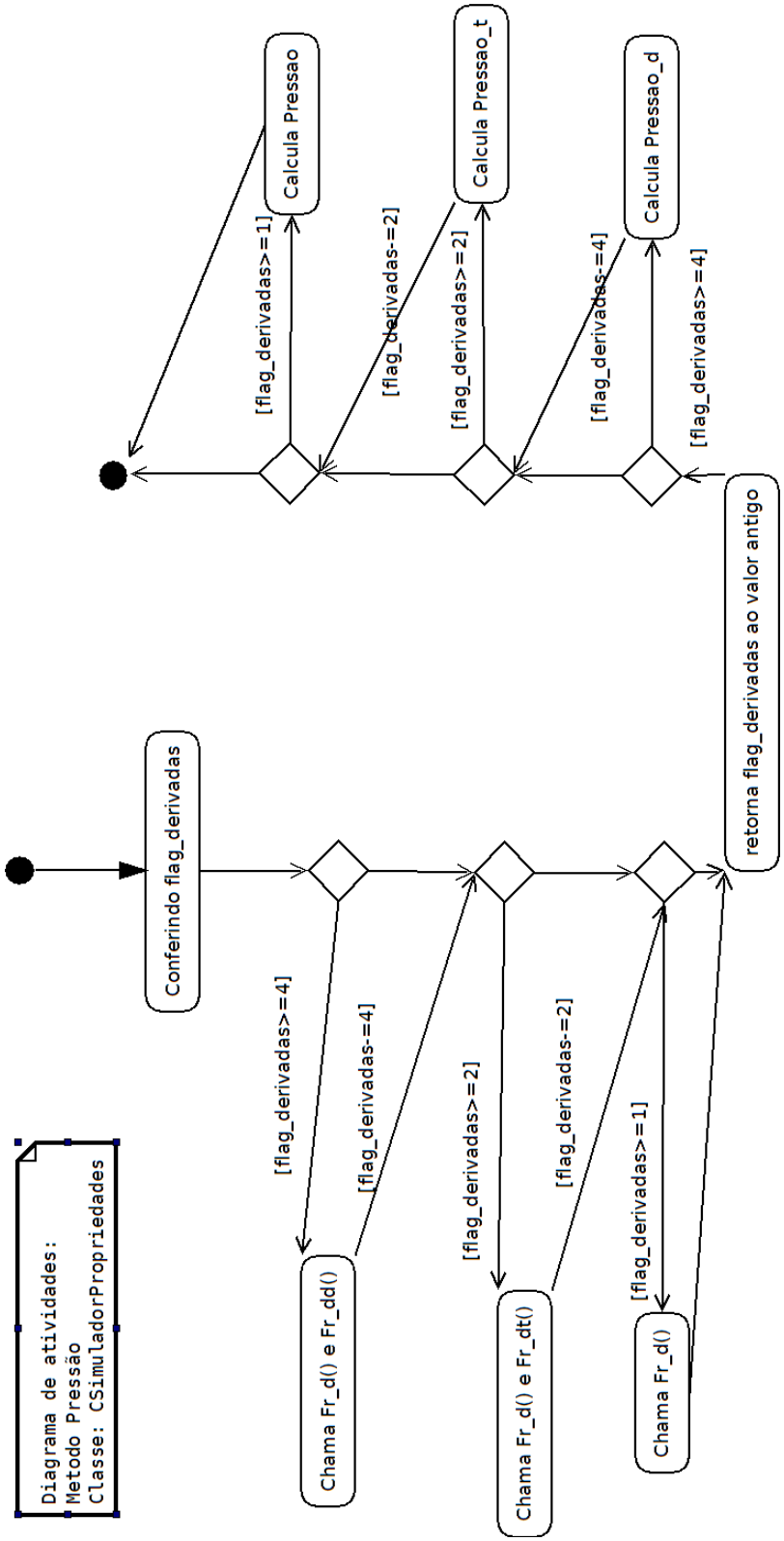

Figura 4.7: Diagrama de atividades, método CSimuladorPropriedades::Pressao

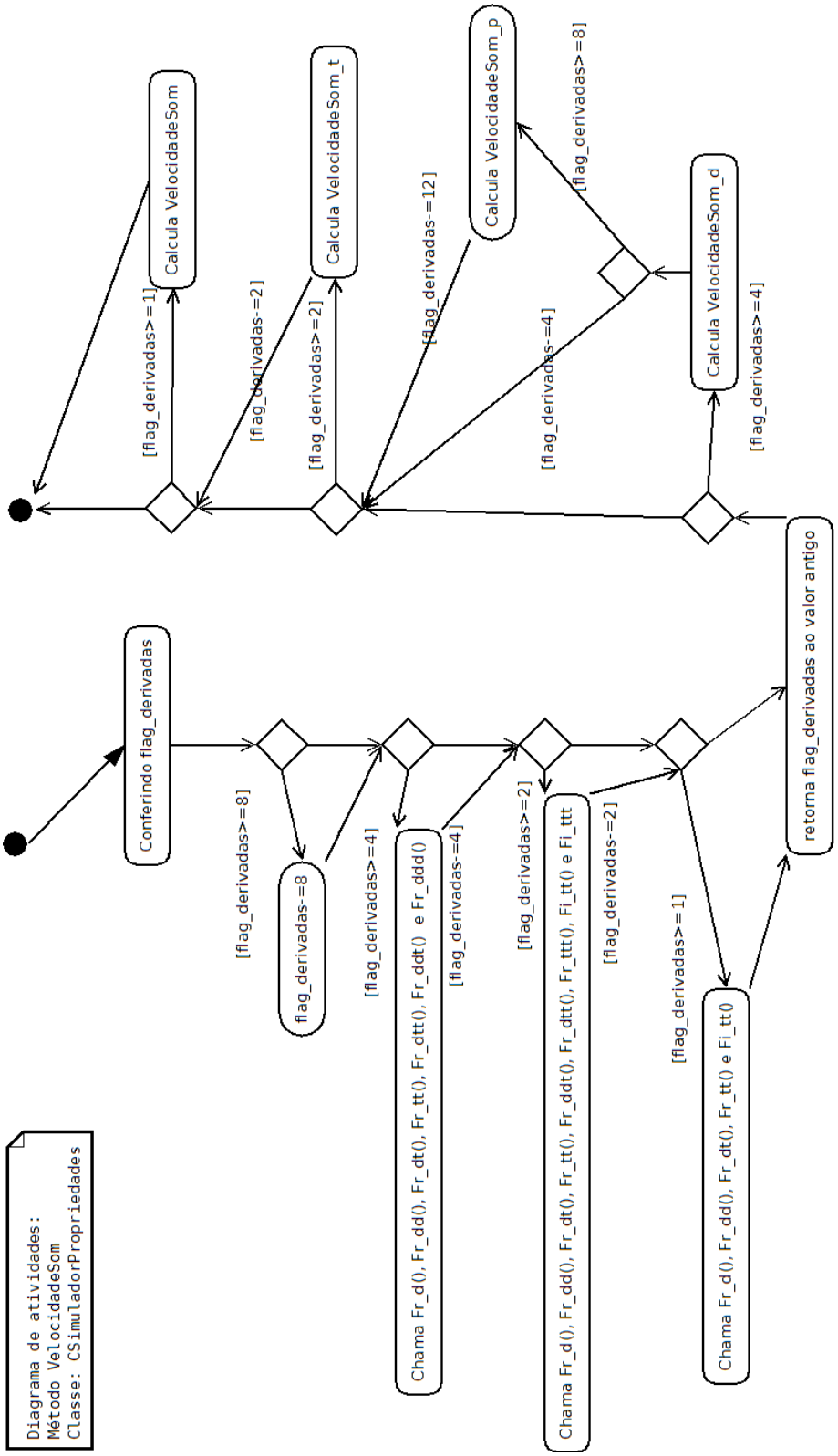

Figura 4.8: Diagrama de atividades, método CSimuladorPropriedades::VelocidadeSom

Capítulo 5

Projeto

Neste capítulo do projeto de engenharia veremos questões associadas ao projeto do sistema.

5.1 Projeto do sistema

Depois da análise orientada a objeto desenvolve-se o projeto do sistema, o qual envolve etapas como a definição dos protocolos, da interface API, o uso de recursos, a subdivisão do sistema em subsistemas, a alocação dos subsistemas ao hardware e a seleção das estruturas de controle, a seleção das plataformas do sistema, das biblitoecas externas, dos padrões de projeto, além da tomada de decisões conceituais e políticas que formam a infraestrutura do projeto.

Deve-se definir padrões de documentação, padrões para o nome das classes, padrões de retorno e de parâmetros em métodos, características da interface do usuário e características de desempenho.

O projeto do sistema é a estratégia de alto nível para resolver o problema e elaborar uma solução. Você deve se preocupar com itens como:

1. Plataformas

   - O software irá funcionar nos sistemas operacionais Windows e GNU/Linux, e possivelmente no Mac OS X, sendo desenvolvido no Windows e testado no Windows e GNU/Linux.
   - Não hávera necessidade de grandes mudanças para tornar o programa multiplataforma pois a linguagem escolhida, C++, tem suporte em todos estes sistemas operacionais, [Bueno, 2003].
   - A interface gráfica sera feita usando Qt:http://qt-project.org/, uma biblioteca multiplataforma.

2. Bibliotecas

   - Será utilizada a biblioteca padrão da linguagem C++.
   - Para a criação da interface gráfica será usada a biblioteca Qt: http://qt-project.org/.
   - O programa será desenvolvido com a interface Dev-C++ 5.6.3 .

3. Ambiente de desenvolvimento

   - O programa será desenvolvido e testado em um notebook HP com Windows 8 64-bit, processador Intel core i3-4000M, 4GB de memória RAM e placa de video intel HD Graphics 4600.

5.2 Projeto orientado a objeto – POO

O projeto orientado a objeto é a etapa posterior ao projeto do sistema. Baseia-se na análise, mas considera as decisões do projeto do sistema. Acrescenta a análise desenvolvida e as características da plataforma escolhida (hardware, sistema operacional e linguagem de programação). Passa pelo maior detalhamento do funcionamento do software, acrescentando atributos e métodos que envolvem a solução de problemas específicos não identificados durante a análise.

Envolve a otimização da estrutura de dados e dos algoritmos, a minimização do tempo de execução, de memória e de custos. Existe um desvio de ênfase para os conceitos da plataforma selecionada.

Exemplo: na análise você define que existe um método para salvar um arquivo em disco, define um atributo nomeDoArquivo, mas não se preocupa com detalhes específicos da linguagem. Já no projeto, você inclui as bibliotecas necessárias para acesso ao disco, cria um objeto específico para acessar o disco, podendo, portanto, acrescentar novas classes àquelas desenvolvidas na análise.

Efeitos do projeto no modelo estrutural

- Adicionar nos diagramas de pacotes as bibliotecas e subsistemas selecionados no projeto do sistema (exemplo: a biblioteca gráfica selecionada).
  - Neste projeto será necessária a adição do pacote da biblioteca Qt.

O diagrama de pacotes da Figura 5.1 mostra a relação entre os diferentes pacotes do software desenvolvido e o pacote Qt.

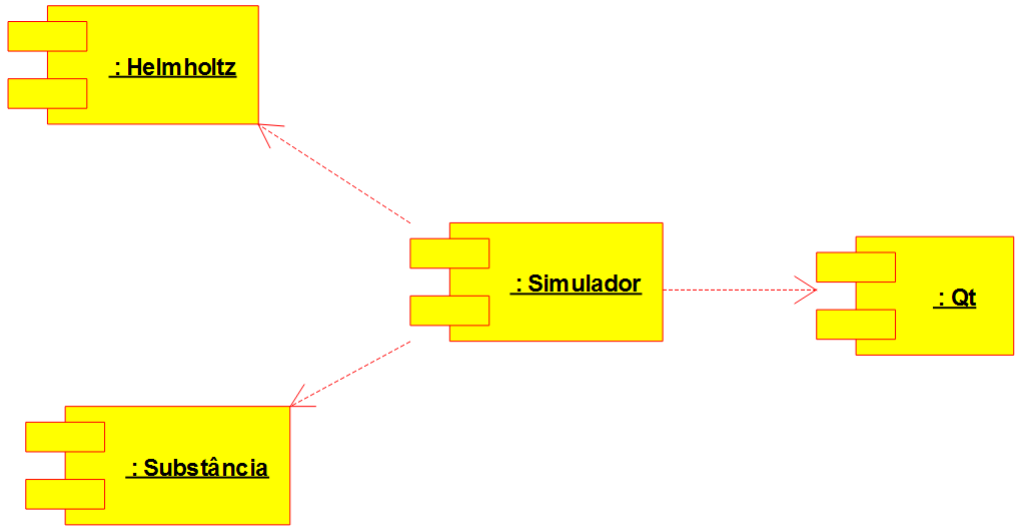

Figura 5.1: Diagrama de Pacotes Revisado

- Novas classes e associações oriundas das bibliotecas selecionadas e da linguagem escolhida devem ser acrescentadas ao modelo.
  - Neste projeto serão adicionadas classes para a interface gráfica.

O diagrama de classes revisado é apresentado na Figura 5.2

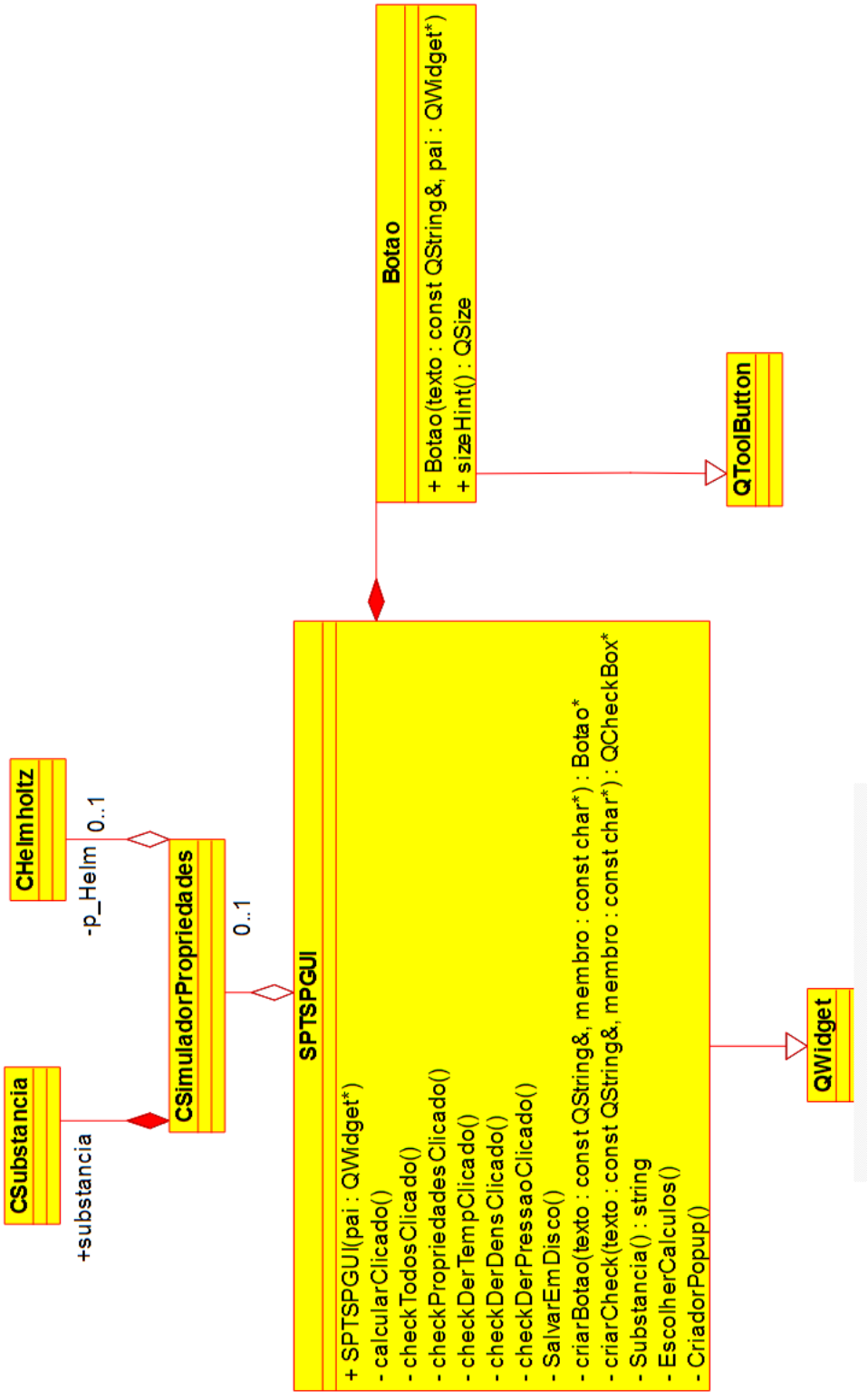

Figura 5.2: Diagrama de Classes Revisado

Efeitos do projeto nas otimizações

- Identifique pontos a serem otimizados em que podem ser utilizados processos concorrentes.
  - Neste projeto pode ser usada a computação em paralelo, pois a parte que requer maior poder de processamento são os cálculos das derivadas da energia de Helmholtz, que são funções

independentes entre si.

– Por exemplo, durante o cálculo da velocidade do som poderiam ser calculados paralelamente $\phi^o_{\tau\tau}$, $\phi^r_{\tau\tau}$, ϕ^r_{δ}, $\phi^r_{\delta\tau}$ e $\phi^r_{\delta\delta}$.

Capítulo 6

Implementação

Neste capítulo do projeto de engenharia apresentamos os códigos fonte que foram desenvolvidos.

6.1 Código fonte

Apresenta-se a seguir um conjunto de classes (arquivos .h e .cpp) além do programa main. Apresenta-se na listagem 6.1 o arquivo com código da classe `CSimuladorPropriedades`.

Listing 6.1: Arquivo de cabeçalho da classe CSimuladorPropriedades.

```
#ifndef CSIMULADORPROPRIEDADES_H
#define CSIMULADORPROPRIEDADES_H
#include "CHelmholtz.h"
#include "CHelmholtzAgua.h"
#include "CSubstancia.h"
#include <string>
////////////////////////////////////////////////////////////////////////
/// @author Thomas Augusto Menegazzo
/// @class CSimuladorPropriedades
/// @file CSimuladorPropriedades.h
/// @brief Classe com metodos de calculo das propriedades temodinamicas a partir de
    CHelmholtz.
////////////////////////////////////////////////////////////////////////

class CSimuladorPropriedades
{
        private:
                ///Ponteiro para a classe que calcula a enegia de Helmholtz.
                CHelmholtz* p_Helm;

                ///Atributo para definir o que calcular.
                ///Funciona estilo somatorio: 1 para calcular a propriedade, 2, 4 e 8
                    para derivadas de temperatura,densidade e pressao respectivamente.
                ///Ex: 7 calcularia tudo menos a derivada com relacao a pressao
                int flag_derivadas;

                ///calcula a derivada da densidade com relacao a pressao,
                ///usada para o calculo das derivadas com relacao a pressao
                void Calculo_dp();
        public:

                ///Construtor da classe.
                ///@param subs Substancia a ser calculada.
                ///@param t Temperatura.
```

```
        ///@param d Densidade.
        CSimuladorPropriedades(std::string subs, double t, double d);

        ///Sobrecarga do construtor.
        ///@param subs Substancia a ser calculada.
        ///@param t Temperatura.
        ///@param d Densidade.
        ///@param flag Valor de flag_derivadas.
        CSimuladorPropriedades(std::string subs, double t, double d, int flag);

        ///Objeto para armazenamento das propriedades obtidas.
CSubstancia substancia;

///Calcula todas as propriedades.
        void CalcularTudo();

        ///Seta o atributo flag_derivadas.
        ///@param flag Valor a ser setado.
        void Set_Flag(int flag){flag_derivadas=flag;}

        ///Calcula a pressao e suas derivadas.
        void Pressao();

        ///Sobrecarga do método Pressao().
///@param flag Valor a ser usado temporariamente como flag_derivadas.
void Pressao(int flag);

///Calcula a entalpia e suas derivadas.
void Entalpia();

///Sobrecarga do método Entalpia().
///@param flag Valor a ser usado temporariamente como flag_derivadas.
void Entalpia(int flag);

///Calcula a entropia e suas derivadas.
void Entropia();

///Sobrecarga do método Entropia().
///@param flag Valor a ser usado temporariamente como flag_derivadas.
void Entropia(int flag);

///Calcula a energia interna e suas derivadas.
void EnergiaInterna();

///Sobrecarga do método EnergiaInterna().
///@param flag Valor a ser usado temporariamente como flag_derivadas.
void EnergiaInterna(int flag);

///Calcula a energia de Gibbs e suas derivadas.
void EnergiaGibbs();

///Sobrecarga do método EnergiaGibbs().
///@param flag Valor a ser usado temporariamente como flag_derivadas.
void EnergiaGibbs(int flag);

///Calcula a capacidade calorífica isobárica e suas derivadas.
void CapCalorificaIsobarica();

///Sobrecarga do método CapCalorificaIsobarica().
///@param flag Valor a ser usado temporariamente como flag_derivadas.
```

```
        void CapCalorificaIsobarica(int flag);

        ///Calcula a capacidade calorífica isocórica e suas derivadas.
        void CapCalorificaIsovolumetrica();

        ///Sobrecarga do método CapCalorificaIsovolumetrica().
        ///@param flag Valor a ser usado temporariamente como flag_derivadas.
        void CapCalorificaIsovolumetrica(int flag);

        ///Calcula coeficiente de Joule-Thomson e suas derivadas.
        void CoefJouleThomson();

        ///Sobrecarga do método CoefJouleThomson().
        ///@param flag Valor a ser usado temporariamente como flag_derivadas.
        void CoefJouleThomson(int flag);

        ///Calcula o coeficiente de enforcamento e suas derivadas.
        void CoefEnforcamento();

        ///Sobrecarga do método CoefEnforcamento().
        ///@param flag Valor a ser usado temporariamente como flag_derivadas.
        void CoefEnforcamento(int flag);

        ///Calcula a velocidade do som no meio e suas derivadas.
        void VelocidadeSom();

        ///Sobrecarga do método VelocidadeSom().
        ///@param flag Valor a ser usado temporariamente como flag_derivadas.
        void VelocidadeSom(int flag);

        ///Calcula o coeficiente de temperatura-pressão isentrópico e suas derivadas.
        void CoefIsentropico();

        ///Sobrecarga do método CoefIsentropico().
        ///@param flag Valor a ser usado temporariamente como flag_derivadas.
        void CoefIsentropico(int flag);

        ///função para mostrar os dados calculados na tela
        void Display();
};
#endif
```

Apresenta-se na listagem 6.2 o arquivo de implementação da classe `CSimuladorPropriedades`.

Listing 6.2: Arquivo de implementação da classe CSimuladorPropriedades.

```
#include "CSimuladorPropriedades.h"
#include <iostream>
#include <cmath>
CSimuladorPropriedades::CSimuladorPropriedades(std::string subs, double t, double d)
{
    //Escolhendo a classe derivada de CHelmholtz.
        if (subs=="H2O"){p_Helm=new CHelmholtzAgua;substancia.massaMolar=18;substancia.
            nome="Agua";}
        if (subs=="CH4"){p_Helm=new CHelmholtzMetano;substancia.massaMolar=16;substancia.
            nome="Metano";}
        if (subs=="C2H6"){p_Helm=new CHelmholtzEtano;substancia.massaMolar=30;substancia.
            nome="Etano";}
        if (subs=="CO2"){p_Helm=new CHelmholtzDioxidoCarbono;substancia.massaMolar=44;
            substancia.nome="Dioxido de Carbono";}
        if (subs=="nC4H10"){p_Helm=new CHelmholtzNbutano;substancia.massaMolar=58.12;
            substancia.nome="n-Butano";}
```

```
        if (subs=="iC4H10"){p_Helm=new CHelmholtzIsobutano;substancia.massaMolar=58.12;
            substancia.nome="isoButano";}
        if (subs=="N2"){p_Helm=new CHelmholtzNitrogenio;substancia.massaMolar=28.013;
            substancia.nome="Nitrogenio";}
    //Passando valores de temperatura e densidade.
    substancia.temperatura=t;
        substancia.densidade=d;
        p_Helm->tau=p_Helm->tCritica/t;
        p_Helm->delta=d/p_Helm->dCritica;
        flag_derivadas=1;
}
CSimuladorPropriedades::CSimuladorPropriedades(std::string subs, double t, double d, int
    flag)
{
    //Escolhendo a classe derivada de CHelmholtz.
        if (subs=="H2O"){p_Helm=new CHelmholtzAgua;substancia.massaMolar=18.01528;
            substancia.nome="Agua";}
        if (subs=="CH4"){p_Helm=new CHelmholtzMetano;substancia.massaMolar=16.04;
            substancia.nome="Metano";}
        if (subs=="C2H6"){p_Helm=new CHelmholtzEtano;substancia.massaMolar=30.07;
            substancia.nome="Etano";}
        if (subs=="CO2"){p_Helm=new CHelmholtzDioxidoCarbono;substancia.massaMolar=44.01;
            substancia.nome="Dioxido de Carbono";}
        if (subs=="nC4H10"){p_Helm=new CHelmholtzNbutano;substancia.massaMolar=58.12;
            substancia.nome="n-Butano";}
        if (subs=="iC4H10"){p_Helm=new CHelmholtzIsobutano;substancia.massaMolar=58.12;
            substancia.nome="isoButano";}
        if (subs=="N2"){p_Helm=new CHelmholtzNitrogenio;substancia.massaMolar=28.013;
            substancia.nome="Nitrogenio";}
    //Passando valores de temperatura e densidade, e da flag.
    substancia.temperatura=t;
        substancia.densidade=d;
        p_Helm->tau=p_Helm->tCritica/t;
        p_Helm->delta=d/p_Helm->dCritica;
        flag_derivadas=flag;
}
void CSimuladorPropriedades::Display()
{
        std::cout<<std::endl<<"Calculando propriedades para "<<substancia.nome<<" a T="<<
            substancia.temperatura<<" e D="<<substancia.densidade<<':'<<std::endl;
        substancia.Display();
}
void CSimuladorPropriedades::Calculo_dp()
{
        if (substancia.pressao.d==0)
        {
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                substancia.pressao.d=(1+2*p_Helm->delta*p_Helm->fResidual_d+pow(p_Helm->
                    delta,2)*p_Helm->fResidual_dd)*substancia.temperatura*p_Helm->
                    rEspecifico/100;
        }
        substancia.d_p=1/substancia.pressao.d;
}
void CSimuladorPropriedades::CalcularTudo()
{
        Pressao();
        Entalpia();
        Entropia();
        EnergiaInterna();
```

```
        EnergiaGibbs();
        CapCalorificaIsobarica();
        CapCalorificaIsovolumetrica();
        CoefJouleThomson();
        CoefEnforcamento();
        VelocidadeSom();
        CoefIsentropico();
}
void CSimuladorPropriedades::Pressao(int flag)
{
    int temp = flag_derivadas;
    Set_Flag(flag);
    Pressao();
    Set_Flag(temp);
}
void CSimuladorPropriedades::Pressao()
{
        int temp=flag_derivadas;
        if (temp>=8)temp-=8;
        if (temp>=4){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                temp-=4;
                }
        if (temp>=2){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                temp-=2;
                }
        if (temp>=1){if (p_Helm->fResidual_d==0){p_Helm->FR_d();}temp-=1;}
        temp=flag_derivadas;
        if (temp>=8)temp-=8;
        if (temp>=4){
                substancia.pressao.d=(1+2*p_Helm->delta*p_Helm->fResidual_d+pow(p_Helm->
                    delta,2)*p_Helm->fResidual_dd)*substancia.temperatura*p_Helm->
                    rEspecifico/100;
                temp-=4;
                }
        if (temp>=2){
                double x=substancia.densidade*p_Helm->rEspecifico*substancia.temperatura
                    *(p_Helm->delta*p_Helm->fResidual_dt-((p_Helm->delta*p_Helm->
                    fResidual_d+1)/p_Helm->tau))/100;
                substancia.pressao.t=-x*p_Helm->tau/substancia.temperatura;
                temp-=2;
                }
        if (temp>=1){
                substancia.pressao.valor=((1+(p_Helm->delta*p_Helm->fResidual_d))*
                    substancia.densidade*substancia.temperatura*p_Helm->rEspecifico)/100;
                temp--;
                }
}
void CSimuladorPropriedades::Entalpia(int flag)
{
    int temp = flag_derivadas;
    Set_Flag(flag);
    Entalpia();
    Set_Flag(temp);
}
void CSimuladorPropriedades::Entalpia()
{
```

```
        int temp=flag_derivadas;
        if (temp>=8)temp-=8;
        if (temp>=4){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                temp-=4;
                }
        if (temp>=2){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
                if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
                temp-=2;}
        if (temp>=1){
                if (p_Helm->fResidual_t==0){p_Helm->FR_t();}
                if (p_Helm->fIdeal_t==0){p_Helm->FI_t();}
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                temp-=1;
                }
        temp=flag_derivadas;
        if (temp>=4){
                substancia.entalpia.d=p_Helm->tCritica*p_Helm->rEspecifico*(p_Helm->
                    fResidual_dt+(p_Helm->fResidual_d+p_Helm->delta*p_Helm->fResidual_dd)
                    /p_Helm->tau)/p_Helm->dCritica;
                if (temp>=8){
                        if (substancia.d_p==0) Calculo_dp();
                        substancia.entalpia.p=substancia.entalpia.d*substancia.d_p;
                        temp-=8;
                        }
                temp-=4;
                }
        if (temp>=2){
                substancia.entalpia.t=-p_Helm->rEspecifico*(pow(p_Helm->tau,2)*(p_Helm->
                    fIdeal_tt+p_Helm->fResidual_tt)+p_Helm->tau*p_Helm->delta*p_Helm->
                    fResidual_dt-(p_Helm->delta*p_Helm->fResidual_d+1));
                temp-=2;
                }
        if (temp>=1){
                substancia.entalpia.valor=(1+p_Helm->tau*(p_Helm->fIdeal_t+p_Helm->
                    fResidual_t)+p_Helm->delta*p_Helm->fResidual_d)*p_Helm->rEspecifico*
                    substancia.temperatura;
                temp-=1;
                }
}void CSimuladorPropriedades::Entropia(int flag)
{
    int temp = flag_derivadas;
    Set_Flag(flag);
    Entropia();
    Set_Flag(temp);
}
void CSimuladorPropriedades::Entropia()
{
        int temp=flag_derivadas;
        if (temp>=8)temp-=8;
        if (temp>=4){
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                temp-=4;
                }
```

```
        if (temp>=2){
                if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
                if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
                temp-=2;
                }
        if (temp>=1){
                if (p_Helm->fResidual_t==0){p_Helm->FR_t();}
                if (p_Helm->fIdeal_t==0){p_Helm->FI_t();}
                if (p_Helm->fResidual==0){p_Helm->FR();}
                if (p_Helm->fIdeal==0){p_Helm->FI();}
                temp-=1;
                }
        temp=flag_derivadas;
        if (temp>=4){
                substancia.entropia.d=p_Helm->rEspecifico*(p_Helm->tau*p_Helm->
                    fResidual_dt-p_Helm->fResidual_d-pow(p_Helm->delta,-1))/p_Helm->
                    dCritica;
                if (temp>=8){
                        if (substancia.d_p==0) Calculo_dp();
                        substancia.entropia.p=substancia.entropia.d*substancia.d_p;
                        temp-=8;
                        }
                temp-=4;}
        if (temp>=2){
                substancia.entropia.t=-p_Helm->rEspecifico*pow(p_Helm->tau,2)*(p_Helm->
                    fIdeal_tt+p_Helm->fResidual_tt)/substancia.temperatura;
                temp-=2;}
        if (temp>=1){
                substancia.entropia.valor=(p_Helm->tau*(p_Helm->fIdeal_t+p_Helm->
                    fResidual_t)-(p_Helm->fIdeal+p_Helm->fResidual))*p_Helm->rEspecifico;
                temp-=1;}
}
void CSimuladorPropriedades::EnergiaGibbs(int flag)
{
    int temp = flag_derivadas;
    Set_Flag(flag);
    EnergiaGibbs();
    Set_Flag(temp);
}
void CSimuladorPropriedades::EnergiaGibbs()
{
        int temp=flag_derivadas;
        if (temp>=8)temp-=8;
        if (temp>=4){
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                temp-=4;
                }
        if (temp>=2){
                if (p_Helm->fResidual_t==0){p_Helm->FR_t();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fIdeal_t==0){p_Helm->FI_t();}
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual==0){p_Helm->FR();}
                if (p_Helm->fIdeal==0){p_Helm->FI();}
                temp-=2;
                }
        if (temp>=1){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual==0){p_Helm->FR();}
```

```
                if (p_Helm->fIdeal==0){p_Helm->FI();}
                temp-=1;
                }
        temp=flag_derivadas;
        if (temp>=4){
                substancia.energiaGibbs.d=p_Helm->rEspecifico*substancia.temperatura*(pow
                    (p_Helm->delta,-1)+2*p_Helm->fResidual_d+p_Helm->delta*p_Helm->
                    fResidual_dd)/p_Helm->dCritica;
                if (temp>=8){
                        if (substancia.d_p==0) Calculo_dp();
                        substancia.energiaGibbs.p=substancia.energiaGibbs.d*substancia.
                            d_p;
                        temp-=8;
                        }
                temp-=4;}
        if (temp>=2){
                double x=p_Helm->tCritica*(p_Helm->fIdeal_t+p_Helm->fResidual_t+p_Helm->
                    delta*p_Helm->fResidual_dt);
                x-=substancia.temperatura*(1+p_Helm->fIdeal+p_Helm->fResidual+p_Helm->
                    delta*p_Helm->fResidual_d);
                substancia.energiaGibbs.t=-x*p_Helm->rEspecifico/substancia.temperatura;
                temp-=2;}
        if (temp>=1){substancia.energiaGibbs.valor=(1+p_Helm->fIdeal+p_Helm->fResidual+(
            p_Helm->delta*p_Helm->fResidual_d))*p_Helm->rEspecifico*substancia.
            temperatura;
                temp-=1;}
}
void CSimuladorPropriedades::EnergiaInterna(int flag)
{
    int temp = flag_derivadas;
    Set_Flag(flag);
    EnergiaInterna();
    Set_Flag(temp);
}
void CSimuladorPropriedades::EnergiaInterna()
{
        int temp=flag_derivadas;
        if (temp>=8)temp-=8;
        if (temp>=4){
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                temp-=4;
                }
        if (temp>=2){
                if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
                if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
                temp-=2;
                }
        if (temp>=1){
                if (p_Helm->fResidual_t==0){p_Helm->FR_t();}
                if (p_Helm->fIdeal_t==0){p_Helm->FI_t();}
                temp-=1;
                }
        temp=flag_derivadas;
        if (temp>=4){
                substancia.energiaInterna.d=p_Helm->tCritica*p_Helm->rEspecifico*p_Helm->
                    fResidual_dt/p_Helm->dCritica;
                if (temp>=8){
                        if (substancia.d_p==0) Calculo_dp();
                        substancia.energiaInterna.p=substancia.energiaInterna.d*
                            substancia.d_p;
```

```
                        temp-=8;
                        }
                temp-=4;}
        if (temp>=2){
                substancia.energiaInterna.t=-p_Helm->tCritica*p_Helm->rEspecifico*(p_Helm
                    ->fResidual_tt+p_Helm->fIdeal_tt)*p_Helm->tau/substancia.temperatura;
                temp-=2;
                }
        if (temp>=1){
                substancia.energiaInterna.valor=p_Helm->tau*(p_Helm->fResidual_t+p_Helm->
                    fIdeal_t)*p_Helm->rEspecifico*substancia.temperatura;
                temp-=1;
                }
}
void CSimuladorPropriedades::CapCalorificaIsovolumetrica(int flag)
{
    int temp = flag_derivadas;
    Set_Flag(flag);
    CapCalorificaIsovolumetrica();
    Set_Flag(temp);
}
void CSimuladorPropriedades::CapCalorificaIsovolumetrica()
{
        int temp=flag_derivadas;
        if (temp>=8)temp-=8;
        if (temp>=4){
                if (p_Helm->fResidual_dtt==0){p_Helm->FR_dtt();}
                temp-=4;
                }
        if (temp>=2){
                if (p_Helm->fResidual_ttt==0){p_Helm->FR_ttt();}
                if (p_Helm->fIdeal_ttt==0){p_Helm->FI_ttt();}
                if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
                if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
                temp-=2;
                }
        if (temp>=1){
                if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
                if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
                temp-=1;
                }
        temp=flag_derivadas;
        if (temp>=4){
                substancia.capCalorificaIsovolumetrica.d=-p_Helm->rEspecifico*pow(p_Helm
                    ->tau,2)*p_Helm->fResidual_dtt/p_Helm->dCritica;
                if (temp>=8){
                        if (substancia.d_p==0) Calculo_dp();
                        substancia.capCalorificaIsovolumetrica.p=substancia.
                            capCalorificaIsovolumetrica.d*substancia.d_p;
                        temp-=8;
                        }
                temp-=4;
                }
        if (temp>=2){
                substancia.capCalorificaIsovolumetrica.t=p_Helm->rEspecifico*pow(p_Helm->
                    tau,2)*(p_Helm->tau*(p_Helm->fResidual_ttt+p_Helm->fIdeal_ttt)+2*(
                    p_Helm->fResidual_tt+p_Helm->fIdeal_tt))/substancia.temperatura;
                temp-=2;}
        if (temp>=1){substancia.capCalorificaIsovolumetrica.valor=-p_Helm->tau*p_Helm->
            tau*(p_Helm->fResidual_tt+p_Helm->fIdeal_tt)*p_Helm->rEspecifico;
```

```
                temp-=1;
                }
}
void CSimuladorPropriedades::CapCalorificaIsobarica(int flag)
{
    int temp = flag_derivadas;
    Set_Flag(flag);
    CapCalorificaIsobarica();
    Set_Flag(temp);
}
void CSimuladorPropriedades::CapCalorificaIsobarica()
{
        int temp=flag_derivadas;
        if (temp>=8)temp-=8;
        if (temp>=4){
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_ddt==0){p_Helm->FR_ddt();}
                if (p_Helm->fResidual_ddd==0){p_Helm->FR_ddd();}
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dtt==0){p_Helm->FR_dtt();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                temp-=4;
                }
        if (temp>=2){
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_ddt==0){p_Helm->FR_ddt();}
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dtt==0){p_Helm->FR_dtt();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (substancia.capCalorificaIsovolumetrica.t==0)
                    CapCalorificaIsovolumetrica();
                temp-=2;
                }
        if (temp>=1){
                if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
                if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                temp-=1;
                }
        temp=flag_derivadas;
        if (temp>=4){
                double p1=-pow(p_Helm->tau,2)*p_Helm->fResidual_dtt;
                double den=1+2*p_Helm->delta*p_Helm->fResidual_d+pow(p_Helm->delta,2)*
                    p_Helm->fResidual_dd;
                double p2=(p_Helm->fResidual_d+p_Helm->delta*p_Helm->fResidual_dd-p_Helm
                    ->tau*p_Helm->fResidual_dt-p_Helm->delta*p_Helm->tau*p_Helm->
                    fResidual_ddt)*2*(1+p_Helm->delta*p_Helm->fResidual_d-p_Helm->delta*
                    p_Helm->tau*p_Helm->fResidual_dt)*den;
                double p3=(2*p_Helm->fResidual_d+4*p_Helm->delta*p_Helm->fResidual_dd+pow
                    (p_Helm->delta,2)*p_Helm->fResidual_ddd)*pow(1+p_Helm->delta*p_Helm->
                    fResidual_d-p_Helm->delta*p_Helm->tau*p_Helm->fResidual_dt,2);
                double x=p1+(p2-p3)/pow(den,2);
                substancia.capCalorificaIsobarica.d=x*p_Helm->rEspecifico/p_Helm->
                    dCritica;
                if (temp>=8){
                        if (substancia.d_p==0) Calculo_dp();
                        substancia.capCalorificaIsobarica.p=substancia.
                            capCalorificaIsobarica.d*substancia.d_p;
```

```
                        temp-=8;
                        }
                temp-=4;
                }
        if (temp>=2){
                double den=1+2*p_Helm->delta*p_Helm->fResidual_d+pow(p_Helm->delta,2)*
                    p_Helm->fResidual_dd;
                double p1=-2*p_Helm->delta*p_Helm->tau*p_Helm->fResidual_dtt*(1+p_Helm->
                    delta*p_Helm->fResidual_d-p_Helm->delta*p_Helm->tau*p_Helm->
                    fResidual_dt)*den;
                double p2=(2*p_Helm->delta*p_Helm->fResidual_dt+pow(p_Helm->delta,2)*
                    p_Helm->fResidual_ddt)*pow(1+p_Helm->delta*p_Helm->fResidual_d-p_Helm
                    ->delta*p_Helm->tau*p_Helm->fResidual_dt,2);
                double f=p_Helm->rEspecifico*(p1-p2)/pow(den,2);
                substancia.capCalorificaIsobarica.t=substancia.
                    capCalorificaIsovolumetrica.t-f*p_Helm->tau/substancia.temperatura;
                temp-=2;
                }
        if (temp>=1){
                double x=-p_Helm->tau*p_Helm->tau*(p_Helm->fIdeal_tt+p_Helm->fResidual_tt
                    );
                x+=pow(1+p_Helm->delta*p_Helm->fResidual_d-p_Helm->delta*p_Helm->tau*
                    p_Helm->fResidual_dt,2)/(1+2*p_Helm->delta*p_Helm->fResidual_d+p_Helm
                    ->fResidual_dd*pow(p_Helm->delta,2));
                substancia.capCalorificaIsobarica.valor=x*p_Helm->rEspecifico;
                temp-=1;
                }
}
void CSimuladorPropriedades::VelocidadeSom(int flag)
{
    int temp = flag_derivadas;
    Set_Flag(flag);
    VelocidadeSom();
    Set_Flag(temp);
}
void CSimuladorPropriedades::VelocidadeSom()
{
        int temp=flag_derivadas;
        if (temp>=8)temp-=8;
        if (temp>=4){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_ddd==0){p_Helm->FR_ddd();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
                if (p_Helm->fResidual_ddt==0){p_Helm->FR_ddt();}
                if (p_Helm->fResidual_dtt==0){p_Helm->FR_dtt();}
                temp-=4;
                }
        if (temp>=2){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
                if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
                if (p_Helm->fResidual_ddt==0){p_Helm->FR_ddt();}
                if (p_Helm->fResidual_dtt==0){p_Helm->FR_dtt();}
                if (p_Helm->fResidual_ttt==0){p_Helm->FR_ttt();}
                if (p_Helm->fIdeal_ttt==0){p_Helm->FI_ttt();}
                temp-=2;
```

```
        }
    if (temp>=1){
        if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
        if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
        if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
        if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
        if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
        temp-=1;
        }
    temp=flag_derivadas;
    if (temp>=4){
        if(substancia.velocidadeSom.valor==0){
            double x=1+2*p_Helm->delta*p_Helm->fResidual_d+pow(p_Helm->delta
                ,2)*p_Helm->fResidual_dd;
            x-=pow(1+p_Helm->delta*p_Helm->fResidual_d-p_Helm->delta*p_Helm->
                tau*p_Helm->fResidual_dt,2)/(pow(p_Helm->tau,2)*(p_Helm->
                fResidual_tt+p_Helm->fIdeal_tt));
            substancia.velocidadeSom.valor=sqrt(x*p_Helm->rEspecifico*
                substancia.temperatura)*31.623164418947891;
            }
        double den=pow(p_Helm->tau,2)*(p_Helm->fIdeal_tt+p_Helm->fResidual_tt);
        double p1=2*p_Helm->fResidual_d+4*p_Helm->delta*p_Helm->fResidual_dd+pow(
            p_Helm->delta,2)*p_Helm->fResidual_ddd;
        double p2=(p_Helm->fResidual_d+p_Helm->delta*p_Helm->fResidual_dd-p_Helm
            ->tau*p_Helm->fResidual_dt-p_Helm->delta*p_Helm->tau*p_Helm->
            fResidual_ddt)*2*(1+p_Helm->delta*p_Helm->fResidual_d-p_Helm->delta*
            p_Helm->tau*p_Helm->fResidual_dt)*den;
        double p3=pow(p_Helm->tau,2)*p_Helm->fResidual_dtt*pow(1+p_Helm->delta*
            p_Helm->fResidual_d-p_Helm->delta*p_Helm->tau*p_Helm->fResidual_dt,2)
            ;
        substancia.velocidadeSom.d=1000*p_Helm->rEspecifico*substancia.
            temperatura*(p1-(p2-p3)/pow(den,2))/(2*p_Helm->dCritica*substancia.
            velocidadeSom.valor);
        if (temp>=8){
            if (substancia.d_p==0) Calculo_dp();
            substancia.velocidadeSom.p=substancia.velocidadeSom.d*substancia.
                d_p;
            temp-=8;
            }
        temp-=4;
        }
    if (temp>=2){
        if(substancia.velocidadeSom.valor==0){
            double x=1+2*p_Helm->delta*p_Helm->fResidual_d+pow(p_Helm->delta
                ,2)*p_Helm->fResidual_dd;
            x-=pow(1+p_Helm->delta*p_Helm->fResidual_d-p_Helm->delta*p_Helm->
                tau*p_Helm->fResidual_dt,2)/(pow(p_Helm->tau,2)*(p_Helm->
                fResidual_tt+p_Helm->fIdeal_tt));
            substancia.velocidadeSom.valor=sqrt(x*p_Helm->rEspecifico*
                substancia.temperatura)*31.623164418947891;
            }
        double den=pow(p_Helm->tau,2)*(p_Helm->fIdeal_tt+p_Helm->fResidual_tt);
        double p1=(pow(1+p_Helm->delta*p_Helm->fResidual_d-p_Helm->delta*p_Helm->
            tau*p_Helm->fResidual_dt,2)/den-(1+2*p_Helm->delta*p_Helm->
            fResidual_d+pow(p_Helm->delta,2)*p_Helm->fResidual_dd))/p_Helm->tau;
        double p2=2*p_Helm->delta*p_Helm->fResidual_dt+pow(p_Helm->delta,2)*
            p_Helm->fResidual_ddt;
        double p3=p_Helm->delta*p_Helm->tau*p_Helm->fResidual_dtt*2*(1+p_Helm->
            delta*p_Helm->fResidual_d-p_Helm->delta*p_Helm->tau*p_Helm->
            fResidual_dt)*den;
```

```
                double p4=(2*p_Helm->tau*(p_Helm->fIdeal_tt+p_Helm->fResidual_tt)+pow(
                    p_Helm->tau,2)*(p_Helm->fIdeal_ttt+p_Helm->fResidual_ttt))*pow(1+
                    p_Helm->delta*p_Helm->fResidual_d-p_Helm->delta*p_Helm->tau*p_Helm->
                    fResidual_dt,2);
                substancia.velocidadeSom.t=-1000*p_Helm->tau*p_Helm->rEspecifico*(p1+p2+(
                    p3+p4)/pow(den,2))/(2*substancia.velocidadeSom.valor);
                temp-=2;
                }
        if (temp>=1&&substancia.velocidadeSom.valor==0){
                double x=1+2*p_Helm->delta*p_Helm->fResidual_d+pow(p_Helm->delta,2)*
                    p_Helm->fResidual_dd;
                x-=pow(1+p_Helm->delta*p_Helm->fResidual_d-p_Helm->delta*p_Helm->tau*
                    p_Helm->fResidual_dt,2)/(pow(p_Helm->tau,2)*(p_Helm->fResidual_tt+
                    p_Helm->fIdeal_tt));
                substancia.velocidadeSom.valor=sqrt(x*p_Helm->rEspecifico*substancia.
                    temperatura)*31.623164418947891;
                temp-=1;
                }
}
void CSimuladorPropriedades::CoefEnforcamento(int flag)
{
    int temp = flag_derivadas;
    Set_Flag(flag);
    CoefEnforcamento();
    Set_Flag(temp);
}
void CSimuladorPropriedades::CoefEnforcamento()
{
        int temp=flag_derivadas;
        if (temp>=8)temp-=8;
        if (temp>=4){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_ddt==0){p_Helm->FR_ddt();}
                if (p_Helm->fResidual_ddd==0){p_Helm->FR_ddd();}
                temp-=4;
                }
        if (temp>=2){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_ddt==0){p_Helm->FR_ddt();}
                if (p_Helm->fResidual_dtt==0){p_Helm->FR_dtt();}
                temp-=2;
                }
        if (temp>=1){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                temp-=1;
                }
        temp=flag_derivadas;
        if (temp>=4){
                double num=1+p_Helm->delta*p_Helm->fResidual_d-p_Helm->delta*p_Helm->tau*
                    p_Helm->fResidual_dt;
                double den=1+2*p_Helm->delta*p_Helm->fResidual_d+pow(p_Helm->delta,2)*
                    p_Helm->fResidual_dd;
                double p1=(num/den-1)/p_Helm->delta;
                double p21=p_Helm->fResidual_d+p_Helm->delta*p_Helm->fResidual_dd-p_Helm
```

```
                    ->tau*p_Helm->fResidual_dt-p_Helm->delta*p_Helm->tau*p_Helm->
                    fResidual_ddt;
                double p22=2*p_Helm->fResidual_d+4*p_Helm->delta*p_Helm->fResidual_dd+pow
                    (p_Helm->delta,2)*p_Helm->fResidual_ddd;
                double p2=(p22*num-p21*den)/pow(den,2);
                substancia.coefEnforcamento.d=(p1+p2)/(substancia.densidade*p_Helm->
                    dCritica);
                if (temp>=8){
                        if (substancia.d_p==0) Calculo_dp();
                        substancia.coefEnforcamento.p=substancia.coefEnforcamento.d*
                            substancia.d_p;
                        temp-=8;
                        }
                temp-=4;
                }
        if (temp>=2){
                double num=1+p_Helm->delta*p_Helm->fResidual_d-p_Helm->delta*p_Helm->tau*
                    p_Helm->fResidual_dt;
                double den=1+2*p_Helm->delta*p_Helm->fResidual_d+pow(p_Helm->delta,2)*
                    p_Helm->fResidual_dd;
                double x=(p_Helm->delta*p_Helm->tau*p_Helm->fResidual_dtt*den+(2*p_Helm->
                    delta*p_Helm->fResidual_dt+pow(p_Helm->delta,2)*p_Helm->fResidual_ddt
                    )*num)/(pow(den,2)*substancia.densidade);
                substancia.coefEnforcamento.t=-x*p_Helm->tau/substancia.temperatura;
                temp-=2;
                }
        if (temp>=1){
                substancia.coefEnforcamento.valor=1-(1+p_Helm->delta*p_Helm->fResidual_d-
                    p_Helm->delta*p_Helm->tau*p_Helm->fResidual_dt)/(1+2*p_Helm->delta*
                    p_Helm->fResidual_d+pow(p_Helm->delta,2)*p_Helm->fResidual_dd);
                substancia.coefEnforcamento.valor/=substancia.densidade;
                temp-=1;
                }
}
void CSimuladorPropriedades::CoefJouleThomson(int flag)
{
    int temp = flag_derivadas;
    Set_Flag(flag);
    CoefJouleThomson();
    Set_Flag(temp);
}
void CSimuladorPropriedades::CoefJouleThomson()
{
        int temp=flag_derivadas;
        if (temp>=8)temp-=8;
        if (temp>=4){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_ddt==0){p_Helm->FR_ddt();}
                if (p_Helm->fResidual_dtt==0){p_Helm->FR_dtt();}
                if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
                if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
                temp-=4;
                }
        if (temp>=2){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_ddt==0){p_Helm->FR_ddt();}
```

```
        if (p_Helm->fResidual_dtt==0){p_Helm->FR_dtt();}
        if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
        if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
        if (p_Helm->fResidual_ttt==0){p_Helm->FR_ttt();}
        if (p_Helm->fIdeal_ttt==0){p_Helm->FI_ttt();}
        temp-=2;
        }
if (temp>=1){
        if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
        if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
        if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
        if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
        if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
        temp-=1;
        }
temp=flag_derivadas;
if (temp>=4){
        double fdt=p_Helm->fResidual_dt;
        double t=p_Helm->tau; double d=p_Helm->delta;
        double den=pow(1+d*p_Helm->fResidual_d-d*t*fdt,2)-pow(t,2)*(p_Helm->
            fIdeal_tt+p_Helm->fResidual_tt)*(1+2*d*p_Helm->fResidual_d+pow(d,2)*
            p_Helm->fResidual_dd);
        double num=d*p_Helm->fResidual_d+pow(d,2)*p_Helm->fResidual_dd+d*t*fdt;
        double p1=num/(den*d);
        double p2=p_Helm->fResidual_d+3*d*p_Helm->fResidual_dd+pow(d,2)*p_Helm->
            fResidual_dd+t*fdt+d*t*p_Helm->fResidual_ddt;
        double p31=(p_Helm->fResidual_d+d*p_Helm->fResidual_dd-t*fdt-d*t*p_Helm->
            fResidual_ddt)*2*(1+d*p_Helm->fResidual_d-d*t*fdt);
        double p32=pow(t,2)*p_Helm->fResidual_dtt*(1+2*d*p_Helm->fResidual_d+pow(
            d,2)*p_Helm->fResidual_dd);
        double p33=pow(t,2)*(p_Helm->fResidual_tt+p_Helm->fIdeal_tt)*(2*p_Helm->
            fResidual_d+4*d*p_Helm->fResidual_dd+pow(d,2)*p_Helm->fResidual_dd);
        double p3=p31-p32-p33;
        double x=(p1-(p2*den-p3*num)/pow(den,2))/(p_Helm->rEspecifico*substancia.
            densidade);
        substancia.coefJouleThomson.d=x*1000/p_Helm->dCritica;
        if (temp>=8){
              if (substancia.d_p==0) Calculo_dp();
              substancia.coefJouleThomson.p=substancia.coefJouleThomson.d*
                  substancia.d_p;
              temp-=8;
              }
        temp-=4;
        }
if (temp>=2){
        double fdt=p_Helm->fResidual_dt;
        double t=p_Helm->tau; double d=p_Helm->delta;
        double den=pow(1+d*p_Helm->fResidual_d-d*t*fdt,2)-pow(t,2)*(p_Helm->
            fIdeal_tt+p_Helm->fResidual_tt)*(1+2*d*p_Helm->fResidual_d+pow(d,2)*
            p_Helm->fResidual_dd);
        double num=d*p_Helm->fResidual_d+pow(d,2)*p_Helm->fResidual_dd+d*t*fdt;
        double p1=2*d*fdt+pow(d,2)*p_Helm->fResidual_ddt+d*t*p_Helm->
            fResidual_dtt;
        double p21=-2*t*(p_Helm->fIdeal_tt+p_Helm->fResidual_tt)*(1+2*d*p_Helm->
            fResidual_d+pow(d,2)*p_Helm->fResidual_dd);
        double p22=-pow(t,2)*(p_Helm->fResidual_ttt+p_Helm->fIdeal_ttt)*(1+2*d*
            p_Helm->fResidual_d+pow(d,2)*p_Helm->fResidual_dd);
        double p23=-pow(t,2)*(p_Helm->fResidual_tt+p_Helm->fIdeal_tt)*(2*d*fdt+
            pow(d,2)*p_Helm->fResidual_ddt);
        double p2=-2*d*t*p_Helm->fResidual_dtt*(1+d*p_Helm->fResidual_d-d*t*fdt)
```

```
                    +(p21+p22+p23);
                double x=-(p1*den-p2*num)/(pow(den,2)*p_Helm->rEspecifico*substancia.
                    densidade);
                substancia.coefJouleThomson.t=-x*t*1000/substancia.temperatura;
                temp-=2;
                }
        if (temp>=1){
                double x=pow(1+p_Helm->delta*p_Helm->fResidual_d-p_Helm->delta*p_Helm->
                    tau*p_Helm->fResidual_dt,2);
                x-=pow(p_Helm->tau,2)*(p_Helm->fResidual_tt+p_Helm->fIdeal_tt)*(1+2*
                    p_Helm->delta*p_Helm->fResidual_d+pow(p_Helm->delta,2)*p_Helm->
                    fResidual_dd);
                substancia.coefJouleThomson.valor=-(p_Helm->delta*p_Helm->fResidual_d+pow
                    (p_Helm->delta,2)*p_Helm->fResidual_dd+p_Helm->delta*p_Helm->tau*
                    p_Helm->fResidual_dt)/x;
                substancia.coefJouleThomson.valor*=1000/(p_Helm->rEspecifico*substancia.
                    densidade);
                temp-=1;
                }
}
void CSimuladorPropriedades::CoefIsentropico(int flag)
{
    int temp = flag_derivadas;
    Set_Flag(flag);
    CoefIsentropico();
    Set_Flag(temp);
}
void CSimuladorPropriedades::CoefIsentropico()
{
        int temp=flag_derivadas;
        if (temp>=8)temp-=8;
        if (temp>=4){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_ddd==0){p_Helm->FR_ddd();}
                if (p_Helm->fResidual_dtt==0){p_Helm->FR_dtt();}
                if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
                if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
                temp-=4;
                }
        if (temp>=2){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_ddt==0){p_Helm->FR_ddt();}
                if (p_Helm->fResidual_dtt==0){p_Helm->FR_dtt();}
                if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
                if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
                if (p_Helm->fResidual_ttt==0){p_Helm->FR_ttt();}
                if (p_Helm->fIdeal_ttt==0){p_Helm->FI_ttt();}
                temp-=2;
                }
        if (temp>=1){
                if (p_Helm->fResidual_d==0){p_Helm->FR_d();}
                if (p_Helm->fResidual_dd==0){p_Helm->FR_dd();}
                if (p_Helm->fResidual_dt==0){p_Helm->FR_dt();}
                if (p_Helm->fResidual_tt==0){p_Helm->FR_tt();}
                if (p_Helm->fIdeal_tt==0){p_Helm->FI_tt();}
                temp-=1;
```

```
            }
        temp=flag_derivadas;
        if (temp>=4){
                double t=p_Helm->tau;double d=p_Helm->delta;
                double num=1+d*p_Helm->fResidual_d-d*t*p_Helm->fResidual_dt;
                double den=pow(num,2)-(pow(t,2)*(p_Helm->fResidual_tt+p_Helm->fIdeal_tt)
                    *(1+2*d*p_Helm->fResidual_d+pow(d,2)*p_Helm->fResidual_dd));
                double p1=p_Helm->fResidual_d+d*p_Helm->fResidual_dd-t*p_Helm->
                    fResidual_dt-d*t*p_Helm->fResidual_dtt;
                double p21=2*p1*num-pow(t,2)*p_Helm->fResidual_dtt*(1+2*d*p_Helm->
                    fResidual_d+pow(d,2)*p_Helm->fResidual_dd);
                double p22=pow(t,2)*(p_Helm->fResidual_tt+p_Helm->fIdeal_tt)*(4*d*p_Helm
                    ->fResidual_dd+2*p_Helm->fResidual_d+pow(d,2)*p_Helm->fResidual_ddd);
                double p2=p21-p22;
                double x=(p1*den-p2*num)/pow(den,2);
                double y=x-num/(den*d);
                substancia.coefIsentropico.d=y/(p_Helm->dCritica*substancia.densidade*
                    p_Helm->rEspecifico);
                if (temp>=8){
                        if (substancia.d_p==0) Calculo_dp();
                        substancia.coefIsentropico.p=substancia.coefIsentropico.d*
                            substancia.d_p;
                        temp-=8;
                        }
                temp-=4;
                }
        if (temp>=2){
                double t=p_Helm->tau;double d=p_Helm->delta;
                double num=1+d*p_Helm->fResidual_d-d*t*p_Helm->fResidual_dt;
                double den=pow(num,2)-(pow(t,2)*(p_Helm->fResidual_tt+p_Helm->fIdeal_tt)
                    *(1+2*d*p_Helm->fResidual_d+pow(d,2)*p_Helm->fResidual_dd));
                double p11=-2*d*t*p_Helm->fResidual_dtt*num;
                double p12=2*t*(p_Helm->fResidual_tt+p_Helm->fIdeal_tt)*(1+2*d*p_Helm->
                    fResidual_d+pow(d,2)*p_Helm->fResidual_dd);
                double p13=pow(t,2)*(p_Helm->fResidual_ttt+p_Helm->fIdeal_ttt)*(1+2*d*
                    p_Helm->fResidual_d+pow(d,2)*p_Helm->fResidual_dd);
                double p14=pow(t,2)*(p_Helm->fResidual_tt+p_Helm->fIdeal_tt)*(2*d*p_Helm
                    ->fResidual_dt+pow(d,2)*p_Helm->fResidual_ddt);
                double p1=p11-(p12+p13+p14);
                double x=(-d*t*p_Helm->fResidual_dtt*den-p1*num)/(p_Helm->rEspecifico*
                    substancia.densidade*pow(den,2));
                substancia.coefIsentropico.t=-x*t/substancia.temperatura;
                temp-=2;
                }
        if (temp>=1){
                double x=pow(1+p_Helm->delta*p_Helm->fResidual_d-p_Helm->delta*p_Helm->
                    tau*p_Helm->fResidual_dt,2);
                x-=pow(p_Helm->tau,2)*(p_Helm->fResidual_tt+p_Helm->fIdeal_tt)*(1+2*
                    p_Helm->delta*p_Helm->fResidual_d+pow(p_Helm->delta,2)*p_Helm->
                    fResidual_dd);
                substancia.coefIsentropico.valor=(1+p_Helm->delta*p_Helm->fResidual_d-
                    p_Helm->delta*p_Helm->tau*p_Helm->fResidual_dt)/x;
                substancia.coefIsentropico.valor/=(p_Helm->rEspecifico*substancia.
                    densidade);
                temp-=1;
                }
}
```

Apresenta-se na listagem 6.3 o arquivo com código da classe `CHelmholtz`.

Listing 6.3: Arquivo de cabeçalho da classe CHelmholtz.

```
#ifndef CHELMHOLTZ_H
#define CHELMHOLTZ_H
////////////////////////////////////////////////////////////////////////
/// @author Thomas Augusto Menegazzo
/// @class CHelmholtz
/// @file CHelmholtz.h
/// @brief Classe com os metodos de calculo da energia de Helmholtz e suas derivadas.
////////////////////////////////////////////////////////////////////////

class CHelmholtz
{
        protected:
                ///Matriz de coeficientes para a parte polinomial.
                double** coeficientesPolinomio;

                ///Matriz de coeficientes para a parte exponencial.
                double** coeficientesExponenciais;

                ///Matriz de coeficientes para a parte de curva Gaussiana.
                double** coeficientesGaussianos;

                ///Matriz de coeficientes para a parte não-analítica.
                double** coeficientesNaoAnaliticos;

                ///Vetor de coeficientes do inicio da parte ideal.
                double* coeficientesIdeais1;

                ///Matriz de coeficientes do fim da parte ideal.
                double** coeficientesIdeais2;

                ///Matriz com o numero de somatorios de cada parte da equacao.
                int numeroCoeficientes[5];

                ///Função para alocacao de matrizes dinâmicas.
                ///@param nx Número de linhas da matriz.
                ///@param ny Número de colunas da matriz.
                static double** Aloca(int nx, int ny);

                ///Função para desalocação.
                ///@param pm Ponteiro para matriz a ser desalocada.
                ///@param nx Número de linhas da matriz.
                static void Desaloca(double**& pm, int nx);
        public:

        ///Temperatura crítica da substancia escolhida.
                double tCritica;

        ///Densidade crítica da substancia escolhida.
                double dCritica;

        ///Constante específica dos gases.
                double rEspecifico;

                ///Densidade reduzida.
        double delta;

                ///Temperatura reduzida.
                double tau;
```

```
        ///Valor da parte ideal da equação.
        double fIdeal;

        ///Valor da primeira derivada com relação à temperatura da parte ideal.
        double fIdeal_t;

        ///Valor da segunda derivada com relação à temperatura da parte ideal.
        double fIdeal_tt;

        ///Valor da terceira derivada com relação à temperatura da parte ideal.
        double fIdeal_ttt;

        ///Valor da parte residual da equacao.
        double fResidual;

        ///Valor da primeira derivada com relação à temperatura da parte residual
            .
        double fResidual_t;

        ///Valor da primeira derivada com relação à densidade da parte residual.
        double fResidual_d;

        ///Valor da segunda derivada com relação à densidade da parte residual.
        double fResidual_dd;

        ///Valor da segunda derivada mista.
        double fResidual_dt;

        ///Valor da segunda derivada com relação à temperatura da parte residual.
        double fResidual_tt;

        ///Valor da terceira derivada com relação à densidade da parte residual.
        double fResidual_ddd;

        ///Valor da derivada com relação à temperatura de fResidual_dd.
        double fResidual_ddt;

        ///Valor da derivada com relação à temperatura de fResidual_dt.
        double fResidual_dtt;

        ///Valor da terceira derivada com relação à temperatura da parte residual
            .
        double fResidual_ttt;
public:

        ///Metodo para cálculo de fIdeal.
        void FI();

        ///Metodo para cálculo de fIdeal_t.
        void FI_t();

        ///Metodo para cálculo de fIdeal_tt.
        void FI_tt();

        ///Metodo para cálculo de fIdeal_ttt.
        void FI_ttt();

        ///Metodo para cálculo de fResidual.
        void FR();
```

```
                ///Metodo para cálculo de fResidual_d.
                void FR_d();

                ///Metodo para cálculo de fResidual_t.
                void FR_t();

                ///Metodo para cálculo de fResidual_dd.
                void FR_dd();

                ///Metodo para cálculo de fResidual_dt.
                void FR_dt();

                ///Metodo para cálculo de fResidual_tt.
                void FR_tt();

                ///Metodo para cálculo de fResidual_ddd.
                void FR_ddd();

                ///Metodo para cálculo de fResidual_ddt.
                void FR_ddt();

                ///Metodo para cálculo de fResidual_dtt.
                void FR_dtt();

                ///Metodo para cálculo de fResidual_ttt.
                void FR_ttt();
};
#endif
```

Apresenta-se na listagem 6.4 o arquivo de implementação da classe `CHelmholtz`.

Listing 6.4: Arquivo de implementação da classe CHelmholtz.

```
#include "CHelmholtz.h"
#include <cmath>
#include <iostream>
using namespace std;
double** CHelmholtz::Aloca(int nx, int ny)
{
        double** pm=new double* [nx];
        for(int i=0;i<nx;i++) pm[i]=new double[ny];
        return pm;
}

void CHelmholtz::Desaloca (double**& pm, int nx)
{
        if(pm)
        for(int i=0;i<nx;i++)delete pm[i];
        delete pm;
}

void CHelmholtz::FR()
{
    fResidual=0;
    //Parte polinomial
        for(int i=0;i<numeroCoeficientes[1];i++){
                fResidual+=coeficientesPolinomio[i][2]*pow(delta,coeficientesPolinomio[i
                    ][0])*pow(tau,coeficientesPolinomio[i][1]);
        }
    //Parte exponencial
        for(int i=0;i<numeroCoeficientes[2];i++){
```

```
                fResidual+=coeficientesExponenciais[i][3]*pow(delta,
                    coeficientesExponenciais[i][1])*pow(tau,coeficientesExponenciais[i
                    ][2])*exp(-pow(delta,coeficientesExponenciais[i][0]));
        }
    //Parte de curva Gaussiana
        for(int i=0;i<numeroCoeficientes[3];i++){
                double a=coeficientesGaussianos[i][2]*pow(delta,coeficientesGaussianos[i
                    ][0])*pow(tau,coeficientesGaussianos[i][1]);
                double b=exp(-coeficientesGaussianos[i][3]*pow((delta-
                    coeficientesGaussianos[i][6]),2)-coeficientesGaussianos[i][4]*pow((
                    tau-coeficientesGaussianos[i][5]),2));
                fResidual+=a*b;
        }
    //Parte nao-analitica
        for(int i=0;i<numeroCoeficientes[4];i++){
                double fi=(1-tau)+coeficientesNaoAnaliticos[i][6]*pow(pow((delta-1),2)
                    ,(1/(2*coeficientesNaoAnaliticos[i][7])));
                double mdelta=pow(fi,2)+coeficientesNaoAnaliticos[i][2]*pow(pow((delta-1)
                    ,2),coeficientesNaoAnaliticos[i][0]);
                double ex=exp(-(coeficientesNaoAnaliticos[i][4]*pow((delta-1),2)+
                    coeficientesNaoAnaliticos[i][5]*pow((tau-1),2)));
                fResidual+=coeficientesNaoAnaliticos[i][3]*ex*delta*pow(mdelta,
                    coeficientesNaoAnaliticos[i][1]);
        }
}

void CHelmholtz::FR_d()
{
        fResidual_d=0;
        for(int i=0;i<numeroCoeficientes[1];i++){
                fResidual_d+=coeficientesPolinomio[i][2]*coeficientesPolinomio[i][0]*pow(
                    delta,((coeficientesPolinomio[i][0]-1)))*pow(tau,
                    coeficientesPolinomio[i][1]);
        }
        for(int i=0;i<numeroCoeficientes[2];i++){
                fResidual_d+=coeficientesExponenciais[i][3]*pow(delta,(
                    coeficientesExponenciais[i][1]-1))*pow(tau,coeficientesExponenciais[i
                    ][2])*(coeficientesExponenciais[i][1]-coeficientesExponenciais[i][0]*
                    pow(delta,coeficientesExponenciais[i][0]))*exp(-pow(delta,
                    coeficientesExponenciais[i][0]));
        }
        for(int i=0;i<numeroCoeficientes[3];i++){
                double a=coeficientesGaussianos[i][2]*pow(delta,coeficientesGaussianos[i
                    ][0])*pow(tau,coeficientesGaussianos[i][1]);
                double b=exp(-coeficientesGaussianos[i][3]*pow((delta-
                    coeficientesGaussianos[i][6]),2)-coeficientesGaussianos[i][4]*pow((
                    tau-coeficientesGaussianos[i][5]),2));
                double c=(coeficientesGaussianos[i][0]/delta)-2*coeficientesGaussianos[i
                    ][3]*(delta-coeficientesGaussianos[i][6]);
                fResidual_d+=a*b*c;
        }
        for(int i=0;i<numeroCoeficientes[4];i++){
                double fi=(1-tau)+coeficientesNaoAnaliticos[i][6]*pow(pow((delta-1),2)
                    ,(1/(2*coeficientesNaoAnaliticos[i][7])));
                double mdelta=pow(fi,2)+coeficientesNaoAnaliticos[i][2]*pow(pow((delta-1)
                    ,2),coeficientesNaoAnaliticos[i][0]);
                double mdelta_d=(delta-1)*(coeficientesNaoAnaliticos[i][6]*fi*(2/
                    coeficientesNaoAnaliticos[i][7])*pow(pow((delta-1),2),(1/(2*
                    coeficientesNaoAnaliticos[i][7]))-1)+2*coeficientesNaoAnaliticos[i
                    ][2]*coeficientesNaoAnaliticos[i][0]*pow(pow((delta-1),2),(
```

```
                    coeficientesNaoAnaliticos[i][0]-1)));
                double mdeltab_d=coeficientesNaoAnaliticos[i][1]*pow(mdelta,(
                    coeficientesNaoAnaliticos[i][1]-1))*mdelta_d;
                double ex=exp(-(coeficientesNaoAnaliticos[i][4]*pow((delta-1),2)+
                    coeficientesNaoAnaliticos[i][5]*pow((tau-1),2)));
                double ex_d=-2*coeficientesNaoAnaliticos[i][4]*(delta-1)*ex;
                fResidual_d+=coeficientesNaoAnaliticos[i][3]*pow(mdelta,
                    coeficientesNaoAnaliticos[i][1])*(ex+delta*ex_d);
                fResidual_d+=coeficientesNaoAnaliticos[i][3]*(delta*mdeltab_d*ex);
        }
}

void CHelmholtz::FR_dd()
{
        fResidual_dd=0;
        for(int i=0;i<numeroCoeficientes[1];i++){
                fResidual_dd+=coeficientesPolinomio[i][2]*coeficientesPolinomio[i][0]*(
                    coeficientesPolinomio[i][0]-1)*pow(delta,((coeficientesPolinomio[i
                    ][0]-2)))*pow(tau,coeficientesPolinomio[i][1]);
        }
        for(int i=0;i<numeroCoeficientes[2];i++){
                fResidual_dd+=coeficientesExponenciais[i][3]*pow(delta,(
                    coeficientesExponenciais[i][1]-2))*pow(tau,coeficientesExponenciais[i
                    ][2])*exp(-pow(delta,coeficientesExponenciais[i][0]))*((
                    coeficientesExponenciais[i][1]-coeficientesExponenciais[i][0]*pow(
                    delta,coeficientesExponenciais[i][0]))*(coeficientesExponenciais[i
                    ][1]-1-coeficientesExponenciais[i][0]*pow(delta,
                    coeficientesExponenciais[i][0]))-pow(coeficientesExponenciais[i
                    ][0],2)*pow(delta,coeficientesExponenciais[i][0]));
        }
        for(int i=0;i<numeroCoeficientes[3];i++){
                double a=coeficientesGaussianos[i][2]*pow(tau,coeficientesGaussianos[i
                    ][1]);
                double b=exp(-coeficientesGaussianos[i][3]*pow((delta-
                    coeficientesGaussianos[i][6]),2)-coeficientesGaussianos[i][4]*pow((
                    tau-coeficientesGaussianos[i][5]),2));
                double c=-2*coeficientesGaussianos[i][3]*pow(delta,coeficientesGaussianos
                    [i][0])+4*pow(coeficientesGaussianos[i][3],2)*pow(delta,
                    coeficientesGaussianos[i][0])*pow(delta-coeficientesGaussianos[i
                    ][6],2);
                double d=-4*coeficientesGaussianos[i][0]*coeficientesGaussianos[i][3]*pow
                    (delta,coeficientesGaussianos[i][0]-1)*(delta-coeficientesGaussianos[
                    i][6])+coeficientesGaussianos[i][0]*(coeficientesGaussianos[i][0]-1)*
                    pow(delta,coeficientesGaussianos[i][0]-2);
                fResidual_dd+=a*b*(c+d);
        }
        for(int i=0;i<numeroCoeficientes[4];i++){
                double fi=(1-tau)+coeficientesNaoAnaliticos[i][6]*pow(pow((delta-1),2)
                    ,(1/(2*coeficientesNaoAnaliticos[i][7])));
                double mdelta=pow(fi,2)+coeficientesNaoAnaliticos[i][2]*pow(pow((delta-1)
                    ,2),coeficientesNaoAnaliticos[i][0]);
                double mdelta_d=(delta-1)*(coeficientesNaoAnaliticos[i][6]*fi*(2/
                    coeficientesNaoAnaliticos[i][7])*pow(pow((delta-1),2),(1/(2*
                    coeficientesNaoAnaliticos[i][7]))-1)+2*coeficientesNaoAnaliticos[i
                    ][2]*coeficientesNaoAnaliticos[i][0]*pow(pow((delta-1),2),(
                    coeficientesNaoAnaliticos[i][0]-1)));
                double mdeltab_d=coeficientesNaoAnaliticos[i][1]*pow(mdelta,(
                    coeficientesNaoAnaliticos[i][1]-1))*mdelta_d;
                double ex=exp(-(coeficientesNaoAnaliticos[i][4]*pow((delta-1),2)+
                    coeficientesNaoAnaliticos[i][5]*pow((tau-1),2)));
```

```
                double ex_d=-2*coeficientesNaoAnaliticos[i][4]*(delta-1)*ex;
                double ex_dd=(2*coeficientesNaoAnaliticos[i][4]*pow(delta-1,2)-1)*2*
                    coeficientesNaoAnaliticos[i][4]*ex;
                double mdelta_dd=(1/(delta-1))*mdelta_d+pow(delta-1,2)*4*
                    coeficientesNaoAnaliticos[i][2]*coeficientesNaoAnaliticos[i][0]*(
                    coeficientesNaoAnaliticos[i][0]-1)*pow(pow(delta-1,2),
                    coeficientesNaoAnaliticos[i][0]-2);
                mdelta_dd+=pow(delta-1,2)*2*pow(coeficientesNaoAnaliticos[i][6],2)*pow(1/
                    coeficientesNaoAnaliticos[i][7],2)*pow(pow(pow(delta-1,2),1/((2*
                    coeficientesNaoAnaliticos[i][7]))-1),2);

                mdelta_dd+=pow(delta-1,2)*coeficientesNaoAnaliticos[i][6]*fi*(4/
                    coeficientesNaoAnaliticos[i][7])*(1/(2*coeficientesNaoAnaliticos[i
                    ][7])-1)*pow(pow(delta-1,2),(1/(2*coeficientesNaoAnaliticos[i][7]))
                    -2);
                double mdeltab_dd=coeficientesNaoAnaliticos[i][1]*(pow(mdelta,
                    coeficientesNaoAnaliticos[i][1]-1)*mdelta_dd+(
                    coeficientesNaoAnaliticos[i][1]-1)*pow(mdelta,
                    coeficientesNaoAnaliticos[i][1]-2)*pow(mdelta_d,2));

                double a1=pow(mdelta,coeficientesNaoAnaliticos[i][1])*(2*ex_d+delta*ex_dd
                    );
                double a2=2*mdeltab_d*(ex+delta*ex_d);
                double a3=mdeltab_dd*delta*ex;
                fResidual_dd+=coeficientesNaoAnaliticos[i][3]*(a1+a2+a3);
        }
}

void CHelmholtz::FR_ddd()
{
        fResidual_ddd=0;
        for(int i=0;i<numeroCoeficientes[1];i++){
                fResidual_ddd+=coeficientesPolinomio[i][2]*coeficientesPolinomio[i][0]*(
                    coeficientesPolinomio[i][0]-1)*(coeficientesPolinomio[i][0]-2)*pow(
                    delta,((coeficientesPolinomio[i][0]-3)))*pow(tau,
                    coeficientesPolinomio[i][1]);
        }
        for(int i=0;i<numeroCoeficientes[2];i++){
                double dc=pow(delta,coeficientesExponenciais[i][0]);
                double c=coeficientesExponenciais[i][0];
                fResidual_ddd+=coeficientesExponenciais[i][3]*exp(-dc)*pow(tau,
                    coeficientesExponenciais[i][2])*(pow(delta,coeficientesExponenciais[i
                    ][1]-3)*((coeficientesExponenciais[i][1]-c*dc)*(
                    coeficientesExponenciais[i][1]-1-c*dc)-pow(c,2)*dc)*(
                    coeficientesExponenciais[i][1]-2)-c*pow(delta,c-1)*pow(delta,
                    coeficientesExponenciais[i][1]-2)*((coeficientesExponenciais[i][1]-c*
                    dc)*(coeficientesExponenciais[i][1]-1-c*dc)-pow(c,2)*dc)-pow(delta,
                    coeficientesExponenciais[i][1]-2)*pow(c,2)*(c*pow(delta,c-1)+pow(
                    delta,c-1)*(2*coeficientesExponenciais[i][1]-1-2*c*dc)));
        }
        for(int i=0;i<numeroCoeficientes[3];i++){
                double a=coeficientesGaussianos[i][2]*pow(tau,coeficientesGaussianos[i
                    ][1]);
                double b=exp(-coeficientesGaussianos[i][3]*pow((delta-
                    coeficientesGaussianos[i][6]),2)-coeficientesGaussianos[i][4]*pow((
                    tau-coeficientesGaussianos[i][5]),2));
                double c=-2*coeficientesGaussianos[i][3]*pow(delta,coeficientesGaussianos
                    [i][0])+4*pow(coeficientesGaussianos[i][3],2)*pow(delta,
                    coeficientesGaussianos[i][0])*pow(delta-coeficientesGaussianos[i
                    ][6],2)-4*coeficientesGaussianos[i][0]*coeficientesGaussianos[i][3]*
```

```
                pow(delta,coeficientesGaussianos[i][0]-1)*(delta-
                coeficientesGaussianos[i][6])+coeficientesGaussianos[i][0]*(
                coeficientesGaussianos[i][0]-1)*pow(delta,coeficientesGaussianos[i
                ][0]-2);
            double d=-2*coeficientesGaussianos[i][3]*coeficientesGaussianos[i][0]*pow
                (delta,coeficientesGaussianos[i][0]-1)+4*pow(coeficientesGaussianos[i
                ][3],2)*coeficientesGaussianos[i][0]*pow(delta,coeficientesGaussianos
                [i][0]-1)*pow(delta-coeficientesGaussianos[i][6],2)+8*pow(
                coeficientesGaussianos[i][3],2)*pow(delta,coeficientesGaussianos[i
                ][0])*(delta-coeficientesGaussianos[i][6])-4*coeficientesGaussianos[i
                ][0]*(coeficientesGaussianos[i][0]-1)*coeficientesGaussianos[i][3]*
                pow(delta,coeficientesGaussianos[i][0]-2)*(delta-
                coeficientesGaussianos[i][6])-4*coeficientesGaussianos[i][0]*
                coeficientesGaussianos[i][3]*pow(delta,coeficientesGaussianos[i
                ][0]-1)+coeficientesGaussianos[i][0]*(coeficientesGaussianos[i][0]-1)
                *(coeficientesGaussianos[i][0]-2)*pow(delta,coeficientesGaussianos[i
                ][0]-3);
            fResidual_ddd-=2*coeficientesGaussianos[i][3]*(delta-
                coeficientesGaussianos[i][6])*a*b*c;
            fResidual_ddd+=a*b*d;
    }
    for(int i=0;i<numeroCoeficientes[4];i++){
            double b=coeficientesNaoAnaliticos[i][1];
            double fi=(1-tau)+coeficientesNaoAnaliticos[i][6]*pow(pow((delta-1),2)
                ,(1/(2*coeficientesNaoAnaliticos[i][7])));
            double mdelta=pow(fi,2)+coeficientesNaoAnaliticos[i][2]*pow(pow((delta-1)
                ,2),coeficientesNaoAnaliticos[i][0]);
            double mdelta_d=(delta-1)*(coeficientesNaoAnaliticos[i][6]*fi*(2/
                coeficientesNaoAnaliticos[i][7])*pow(pow((delta-1),2),(1/(2*
                coeficientesNaoAnaliticos[i][7]))-1)+2*coeficientesNaoAnaliticos[i
                ][2]*coeficientesNaoAnaliticos[i][0]*pow(pow((delta-1),2),(
                coeficientesNaoAnaliticos[i][0]-1)));
            double mdeltab_d=coeficientesNaoAnaliticos[i][1]*pow(mdelta,(
                coeficientesNaoAnaliticos[i][1]-1))*mdelta_d;
            double ex=exp(-(coeficientesNaoAnaliticos[i][4]*pow((delta-1),2)+
                coeficientesNaoAnaliticos[i][5]*pow((tau-1),2)));
            double ex_d=-2*coeficientesNaoAnaliticos[i][4]*(delta-1)*ex;
            double ex_dd=(2*coeficientesNaoAnaliticos[i][4]*pow(delta-1,2)-1)*2*
                coeficientesNaoAnaliticos[i][4]*ex;
            double mdelta_dd=(1/(delta-1))*mdelta_d+pow(delta-1,2)*4*
                coeficientesNaoAnaliticos[i][2]*coeficientesNaoAnaliticos[i][0]*(
                coeficientesNaoAnaliticos[i][0]-1)*pow(pow(delta-1,2),
                coeficientesNaoAnaliticos[i][0]-2);
            mdelta_dd+=pow(delta-1,2)*2*pow(coeficientesNaoAnaliticos[i][6],2)*pow(1/
                coeficientesNaoAnaliticos[i][7],2)*pow(pow(pow(delta-1,2),1/((2*
                coeficientesNaoAnaliticos[i][7]))-1),2);
            mdelta_dd+=pow(delta-1,2)*coeficientesNaoAnaliticos[i][6]*fi*(4/
                coeficientesNaoAnaliticos[i][7])*(1/(2*coeficientesNaoAnaliticos[i
                ][7])-1)*pow(pow(delta-1,2),(1/(2*coeficientesNaoAnaliticos[i][7]))
                -2);

            double mdeltab_dd=coeficientesNaoAnaliticos[i][1]*(pow(mdelta,
                coeficientesNaoAnaliticos[i][1]-1)*mdelta_dd+(
                coeficientesNaoAnaliticos[i][1]-1)*pow(mdelta,
                coeficientesNaoAnaliticos[i][1]-2)*pow(mdelta_d,2));
            double mdelta_ddd=mdelta_dd/(delta-1)-mdelta_d/pow(delta-1,2);//ok
            mdelta_ddd+=2*(delta-1)*(4*coeficientesNaoAnaliticos[i][2]*
                coeficientesNaoAnaliticos[i][0]*(coeficientesNaoAnaliticos[i][0]-1)*
                pow(pow(delta-1,2),coeficientesNaoAnaliticos[i][0]-2)+2*pow(
                coeficientesNaoAnaliticos[i][6]/coeficientesNaoAnaliticos[i][7],2)*
```

```
                    pow(pow(pow(delta-1,2),1/(2*coeficientesNaoAnaliticos[i][7])-1),2)
                    +(4*coeficientesNaoAnaliticos[i][6]/coeficientesNaoAnaliticos[i][7])*
                    fi*(1/(2*coeficientesNaoAnaliticos[i][7])-1)*pow(pow(delta-1,2),1/(2*
                    coeficientesNaoAnaliticos[i][7])-2));
                mdelta_ddd+=pow(delta-1,3)*(8*coeficientesNaoAnaliticos[i][2]*
                    coeficientesNaoAnaliticos[i][0]*(coeficientesNaoAnaliticos[i][0]-1)*(
                    coeficientesNaoAnaliticos[i][0]-2)*pow(pow(delta-1,2),
                    coeficientesNaoAnaliticos[i][0]-3)+12*pow(coeficientesNaoAnaliticos[i
                    ][6]/coeficientesNaoAnaliticos[i][7],2)*(1/(2*
                    coeficientesNaoAnaliticos[i][7])-1)*pow(pow(delta-1,2),1/(2*
                    coeficientesNaoAnaliticos[i][7])-1)*pow(pow(delta-1,2),1/(2*
                    coeficientesNaoAnaliticos[i][7])-2)+8*(coeficientesNaoAnaliticos[i
                    ][6]/coeficientesNaoAnaliticos[i][7])*fi*(1/(2*
                    coeficientesNaoAnaliticos[i][7])-1)*(1/(2*coeficientesNaoAnaliticos[i
                    ][7])-2)*pow(pow(delta-1,2),1/(2*coeficientesNaoAnaliticos[i][7])-3))
                    ;
                double ex_ddd=4*pow(coeficientesNaoAnaliticos[i][4],2)*(delta-1)*(3-2*
                    coeficientesNaoAnaliticos[i][4]*pow(delta-1,2))*ex;
                double mdeltab_ddd=b*(pow(mdelta,b-1)*mdelta_ddd+(b-1)*pow(mdelta,b-2)*
                    mdelta_d*mdelta_dd+(b-1)*(b-2)*pow(mdelta,b-3)*pow(mdelta_d,3)+(b-1)*
                    pow(mdelta,b-2)*mdelta_dd*2*mdelta_d);

                double p1=3*mdeltab_d*(2*ex_d+delta*ex_dd);
                double p2=3*mdeltab_dd*(ex+delta*ex_d);
                double p3=pow(mdelta,coeficientesNaoAnaliticos[i][1])*(3*ex_dd+delta*
                    ex_ddd);
                double p4=mdeltab_ddd*delta*ex;
                fResidual_ddd+=coeficientesNaoAnaliticos[i][3]*(p1+p2+p3+p4);
        }
}

void CHelmholtz::FR_ddt()
{
        fResidual_ddt=0;
        for(int i=0;i<numeroCoeficientes[1];i++){
                fResidual_ddt+=coeficientesPolinomio[i][2]*coeficientesPolinomio[i][0]*(
                    coeficientesPolinomio[i][0]-1)*pow(delta,((coeficientesPolinomio[i
                    ][0]-2)))*coeficientesPolinomio[i][1]*pow(tau,coeficientesPolinomio[i
                    ][1]-1);
        }
        for(int i=0;i<numeroCoeficientes[2];i++){
                fResidual_ddt+=coeficientesExponenciais[i][3]*pow(delta,(
                    coeficientesExponenciais[i][1]-2))*coeficientesExponenciais[i][2]*pow
                    (tau,coeficientesExponenciais[i][2]-1)*exp(-pow(delta,
                    coeficientesExponenciais[i][0]))*((coeficientesExponenciais[i][1]-
                    coeficientesExponenciais[i][0]*pow(delta,coeficientesExponenciais[i
                    ][0]))*(coeficientesExponenciais[i][1]-1-coeficientesExponenciais[i
                    ][0]*pow(delta,coeficientesExponenciais[i][0]))-pow(
                    coeficientesExponenciais[i][0],2)*pow(delta,coeficientesExponenciais[
                    i][0]));
        }
        for(int i=0;i<numeroCoeficientes[3];i++){
                double a=coeficientesGaussianos[i][2]*pow(tau,coeficientesGaussianos[i
                    ][1]);
                double b=exp(-coeficientesGaussianos[i][3]*pow((delta-
                    coeficientesGaussianos[i][6]),2)-coeficientesGaussianos[i][4]*pow((
                    tau-coeficientesGaussianos[i][5]),2));
                double c=-2*coeficientesGaussianos[i][3]*pow(delta,coeficientesGaussianos
                    [i][0])+4*pow(coeficientesGaussianos[i][3],2)*pow(delta,
                    coeficientesGaussianos[i][0])*pow(delta-coeficientesGaussianos[i
```

```
                ][6],2);
            double d=-4*coeficientesGaussianos[i][0]*coeficientesGaussianos[i][3]*pow
                (delta,coeficientesGaussianos[i][0]-1)*(delta-coeficientesGaussianos[
                i][6])+coeficientesGaussianos[i][0]*(coeficientesGaussianos[i][0]-1)*
                pow(delta,coeficientesGaussianos[i][0]-2);
            double e=(coeficientesGaussianos[i][1]/tau)-2*coeficientesGaussianos[i
                ][4]*(tau-coeficientesGaussianos[i][5]);
            fResidual_ddt+=a*b*(c+d)*e;
        }
        for(int i=0;i<numeroCoeficientes[4];i++){
            double fi=(1-tau)+coeficientesNaoAnaliticos[i][6]*pow(pow((delta-1),2)
                ,(1/(2*coeficientesNaoAnaliticos[i][7])));
            double mdelta=pow(fi,2)+coeficientesNaoAnaliticos[i][2]*pow(pow((delta-1)
                ,2),coeficientesNaoAnaliticos[i][0]);
            double mdelta_d=(delta-1)*(coeficientesNaoAnaliticos[i][6]*fi*(2/
                coeficientesNaoAnaliticos[i][7])*pow(pow((delta-1),2),(1/(2*
                coeficientesNaoAnaliticos[i][7]))-1)+2*coeficientesNaoAnaliticos[i
                ][2]*coeficientesNaoAnaliticos[i][0]*pow(pow((delta-1),2),(
                coeficientesNaoAnaliticos[i][0]-1)));
            double mdeltab_d=coeficientesNaoAnaliticos[i][1]*pow(mdelta,(
                coeficientesNaoAnaliticos[i][1]-1))*mdelta_d;
            double ex=exp(-(coeficientesNaoAnaliticos[i][4]*pow((delta-1),2)+
                coeficientesNaoAnaliticos[i][5]*pow((tau-1),2)));
            double ex_d=-2*coeficientesNaoAnaliticos[i][4]*(delta-1)*ex;
            double ex_dd=(2*coeficientesNaoAnaliticos[i][4]*pow(delta-1,2)-1)*2*
                coeficientesNaoAnaliticos[i][4]*ex;
            double mdelta_dd=(1/(delta-1))*mdelta_d+pow(delta-1,2)*4*
                coeficientesNaoAnaliticos[i][2]*coeficientesNaoAnaliticos[i][0]*(
                coeficientesNaoAnaliticos[i][0]-1)*pow(pow(delta-1,2),
                coeficientesNaoAnaliticos[i][0]-2);
            mdelta_dd+=pow(delta-1,2)*2*pow(coeficientesNaoAnaliticos[i][6],2)*pow(1/
                coeficientesNaoAnaliticos[i][7],2)*pow(pow(pow(delta-1,2),1/((2*
                coeficientesNaoAnaliticos[i][7]))-1),2);
            mdelta_dd+=pow(delta-1,2)*coeficientesNaoAnaliticos[i][6]*fi*(4/
                coeficientesNaoAnaliticos[i][7])*(1/(2*coeficientesNaoAnaliticos[i
                ][7])-1)*pow(pow(delta-1,2),(1/(2*coeficientesNaoAnaliticos[i][7]))
                -2);
            double mdeltab_dd=coeficientesNaoAnaliticos[i][1]*(pow(mdelta,
                coeficientesNaoAnaliticos[i][1]-1)*mdelta_dd+(
                coeficientesNaoAnaliticos[i][1]-1)*pow(mdelta,
                coeficientesNaoAnaliticos[i][1]-2)*pow(mdelta_d,2));
            double mdeltab_t=-2*fi*coeficientesNaoAnaliticos[i][1]*pow(mdelta,
                coeficientesNaoAnaliticos[i][1]-1);
            double ex_t=-2*coeficientesNaoAnaliticos[i][5]*(tau-1)*ex;
            double mdeltab_dt=-coeficientesNaoAnaliticos[i][6]*
                coeficientesNaoAnaliticos[i][1]*(2/coeficientesNaoAnaliticos[i][7])*
                pow(mdelta,coeficientesNaoAnaliticos[i][1]-1)*(delta-1)*pow(pow(delta
                -1,2),(1/(2*coeficientesNaoAnaliticos[i][7])-1))-2*fi*
                coeficientesNaoAnaliticos[i][1]*(coeficientesNaoAnaliticos[i][1]-1)*
                pow(mdelta,coeficientesNaoAnaliticos[i][1]-2)*mdelta_d;
            double ex_dt=4*coeficientesNaoAnaliticos[i][4]*coeficientesNaoAnaliticos[
                i][5]*(delta-1)*(tau-1)*ex;
            double mdelta_dt=(1-delta)*(0.64/0.3)*pow(pow(delta-1,2),(1/0.6)-1);

            double mdelta_ddt=mdelta_dt/(delta-1)-pow(delta-1,2)*
                coeficientesNaoAnaliticos[i][6]*(4/coeficientesNaoAnaliticos[i][7])
                *(1/(2*coeficientesNaoAnaliticos[i][7])-1)*pow(pow(delta-1,2),1/(2*
                coeficientesNaoAnaliticos[i][7])-2);

            double mdeltab_ddt=coeficientesNaoAnaliticos[i][1]*(mdelta_ddt*pow(mdelta
```

```
                    ,coeficientesNaoAnaliticos[i][1]-1)-2*fi*(coeficientesNaoAnaliticos[i
                    ][1]-1)*pow(mdelta,coeficientesNaoAnaliticos[i][1]-2)*mdelta_dd+(
                    coeficientesNaoAnaliticos[i][1]-1)*pow(mdelta,
                    coeficientesNaoAnaliticos[i][1]-2)*2*mdelta_dt*mdelta_d-2*fi*(
                    coeficientesNaoAnaliticos[i][1]-2)*(coeficientesNaoAnaliticos[i
                    ][1]-1)*pow(mdelta,coeficientesNaoAnaliticos[i][1]-3)*mdelta_d*
                    mdelta_d);

                double ex_ddt=-4*coeficientesNaoAnaliticos[i][5]*
                    coeficientesNaoAnaliticos[i][4]*(tau-1)*(2*coeficientesNaoAnaliticos[
                    i][4]*pow(delta-1,2)-1)*ex;

                double a1=pow(mdelta,coeficientesNaoAnaliticos[i][1])*(2*ex_dt+delta*
                    ex_ddt);
                double a2=mdeltab_t*(2*ex_d+delta*ex_dd);
                double a3=2*mdeltab_d*(ex_t+delta*ex_dt);
                double a4=2*mdeltab_dt*(ex+delta*ex_d);
                double a5=mdeltab_dd*delta*ex_t;
                double a6=mdeltab_ddt*delta*ex;
                fResidual_ddt+=coeficientesNaoAnaliticos[i][3]*(a1+a2+a3+a4+a5+a6);
        }
}

void CHelmholtz::FR_dt()
{
        fResidual_dt=0;
        for(int i=0;i<numeroCoeficientes[1];i++){
                fResidual_dt+=coeficientesPolinomio[i][2]*coeficientesPolinomio[i][0]*
                    coeficientesPolinomio[i][1]*pow(delta,((coeficientesPolinomio[i
                    ][0]-1)))*pow(tau,coeficientesPolinomio[i][1]-1);
        }
        for(int i=0;i<numeroCoeficientes[2];i++){
                fResidual_dt+=coeficientesExponenciais[i][3]*coeficientesExponenciais[i
                    ][2]*pow(delta,(coeficientesExponenciais[i][1]-1))*pow(tau,
                    coeficientesExponenciais[i][2]-1)*(coeficientesExponenciais[i][1]-
                    coeficientesExponenciais[i][0]*pow(delta,coeficientesExponenciais[i
                    ][0]))*exp(-pow(delta,coeficientesExponenciais[i][0]));
        }
        for(int i=0;i<numeroCoeficientes[3];i++){
                double a=coeficientesGaussianos[i][2]*pow(delta,coeficientesGaussianos[i
                    ][0])*pow(tau,coeficientesGaussianos[i][1]);
                double b=exp(-coeficientesGaussianos[i][3]*pow((delta-
                    coeficientesGaussianos[i][6]),2)-coeficientesGaussianos[i][4]*pow((
                    tau-coeficientesGaussianos[i][5]),2));
                double c=(coeficientesGaussianos[i][0]/delta)-2*coeficientesGaussianos[i
                    ][3]*(delta-coeficientesGaussianos[i][6]);
                double d=(coeficientesGaussianos[i][1]/tau)-2*coeficientesGaussianos[i
                    ][4]*(tau-coeficientesGaussianos[i][5]);
                fResidual_dt+=a*b*c*d;
        }
        for(int i=0;i<numeroCoeficientes[4];i++){
                double fi=(1-tau)+coeficientesNaoAnaliticos[i][6]*pow(pow((delta-1),2)
                    ,(1/(2*coeficientesNaoAnaliticos[i][7])));
                double mdelta=pow(fi,2)+coeficientesNaoAnaliticos[i][2]*pow(pow((delta-1)
                    ,2),coeficientesNaoAnaliticos[i][0]);
                double mdelta_d=(delta-1)*(coeficientesNaoAnaliticos[i][6]*fi*(2/
                    coeficientesNaoAnaliticos[i][7])*pow(pow((delta-1),2),(1/(2*
                    coeficientesNaoAnaliticos[i][7]))-1)+2*coeficientesNaoAnaliticos[i
                    ][2]*coeficientesNaoAnaliticos[i][0]*pow(pow((delta-1),2),(
                    coeficientesNaoAnaliticos[i][0]-1)));
```

```
                double mdeltab_d=coeficientesNaoAnaliticos[i][1]*pow(mdelta,(
                    coeficientesNaoAnaliticos[i][1]-1))*mdelta_d;
                double ex=exp(-(coeficientesNaoAnaliticos[i][4]*pow((delta-1),2)+
                    coeficientesNaoAnaliticos[i][5]*pow((tau-1),2)));
                double ex_d=-2*coeficientesNaoAnaliticos[i][4]*(delta-1)*ex;
                double mdeltab_t=-2*fi*coeficientesNaoAnaliticos[i][1]*pow(mdelta,
                    coeficientesNaoAnaliticos[i][1]-1);
                double ex_t=-2*coeficientesNaoAnaliticos[i][5]*(tau-1)*ex;
                double mdeltab_dt=-coeficientesNaoAnaliticos[i][6]*
                    coeficientesNaoAnaliticos[i][1]*(2/coeficientesNaoAnaliticos[i][7])*
                    pow(mdelta,coeficientesNaoAnaliticos[i][1]-1)*(delta-1)*pow(pow(delta
                    -1,2),(1/(2*coeficientesNaoAnaliticos[i][7])-1))-2*fi*
                    coeficientesNaoAnaliticos[i][1]*(coeficientesNaoAnaliticos[i][1]-1)*
                    pow(mdelta,coeficientesNaoAnaliticos[i][1]-2)*mdelta_d;
                double ex_dt=4*coeficientesNaoAnaliticos[i][4]*coeficientesNaoAnaliticos[
                    i][5]*(delta-1)*(tau-1)*ex;
                fResidual_dt+=coeficientesNaoAnaliticos[i][3]*pow(mdelta,
                    coeficientesNaoAnaliticos[i][1])*(ex_t+delta*ex_dt);
                fResidual_dt+=coeficientesNaoAnaliticos[i][3]*delta*mdeltab_d*ex_t;
                fResidual_dt+=coeficientesNaoAnaliticos[i][3]*mdeltab_t*(ex+delta*ex_d);
                fResidual_dt+=coeficientesNaoAnaliticos[i][3]*mdeltab_dt*delta*ex;
        }
}

void CHelmholtz::FR_dtt()
{
        //falta terminar
        fResidual_dtt=0;
        for(int i=0;i<numeroCoeficientes[1];i++){
                fResidual_dtt+=coeficientesPolinomio[i][2]*coeficientesPolinomio[i][0]*
                    coeficientesPolinomio[i][1]*(coeficientesPolinomio[i][1]-1)*pow(delta
                    ,coeficientesPolinomio[i][0]-1)*pow(tau,coeficientesPolinomio[i
                    ][1]-2);
        }
        for(int i=0;i<numeroCoeficientes[2];i++){
                fResidual_dtt+=coeficientesExponenciais[i][3]*coeficientesExponenciais[i
                    ][2]*(coeficientesExponenciais[i][2]-1)*pow(delta,(
                    coeficientesExponenciais[i][1]-1))*pow(tau,coeficientesExponenciais[i
                    ][2]-2)*(coeficientesExponenciais[i][1]-coeficientesExponenciais[i
                    ][0]*pow(delta,coeficientesExponenciais[i][0]))*exp(-pow(delta,
                    coeficientesExponenciais[i][0]));
        }
        for(int i=0;i<numeroCoeficientes[3];i++){
                double a=coeficientesGaussianos[i][2]*pow(delta,coeficientesGaussianos[i
                    ][0])*pow(tau,coeficientesGaussianos[i][1]);
                double b=exp(-coeficientesGaussianos[i][3]*pow((delta-
                    coeficientesGaussianos[i][6]),2)-coeficientesGaussianos[i][4]*pow((
                    tau-coeficientesGaussianos[i][5]),2));
                double c=(coeficientesGaussianos[i][0]/delta)-2*coeficientesGaussianos[i
                    ][3]*(delta-coeficientesGaussianos[i][6]);
                double d=pow((coeficientesGaussianos[i][1]/tau)-2*coeficientesGaussianos[
                    i][4]*(tau-coeficientesGaussianos[i][5]),2)-(coeficientesGaussianos[i
                    ][1]/pow(tau,2))-2*coeficientesGaussianos[i][4];
                fResidual_dtt+=a*b*c*d;
        }
}

void CHelmholtz::FR_t()
{
        fResidual_t=0;
```

```
        for(int i=0;i<numeroCoeficientes[1];i++){
                fResidual_t+=coeficientesPolinomio[i][2]*coeficientesPolinomio[i][1]*pow(
                    delta,coeficientesPolinomio[i][0])*pow(tau,(coeficientesPolinomio[i
                    ][1]-1));
        }
        for(int i=0;i<numeroCoeficientes[2];i++){
                fResidual_t+=coeficientesExponenciais[i][3]*coeficientesExponenciais[i
                    ][2]*pow(delta,coeficientesExponenciais[i][1])*pow(tau,
                    coeficientesExponenciais[i][2]-1)*exp(-pow(delta,
                    coeficientesExponenciais[i][0]));
        }
        for(int i=0;i<numeroCoeficientes[3];i++){
                double a=coeficientesGaussianos[i][2]*pow(delta,coeficientesGaussianos[i
                    ][0])*pow(tau,coeficientesGaussianos[i][1]);
                double b=exp(-coeficientesGaussianos[i][3]*pow((delta-
                    coeficientesGaussianos[i][6]),2)-coeficientesGaussianos[i][4]*pow((
                    tau-coeficientesGaussianos[i][5]),2));
                double c=(coeficientesGaussianos[i][1]/tau)-2*coeficientesGaussianos[i
                    ][4]*(tau-coeficientesGaussianos[i][5]);
                fResidual_t+=a*b*c;
        }
        for(int i=0;i<numeroCoeficientes[4];i++){
                double fi=(1-tau)+coeficientesNaoAnaliticos[i][6]*pow(pow((delta-1),2)
                    ,(1/(2*coeficientesNaoAnaliticos[i][7])));
                double mdelta=pow(fi,2)+coeficientesNaoAnaliticos[i][2]*pow(pow((delta-1)
                    ,2),coeficientesNaoAnaliticos[i][0]);
                double ex=exp(-(coeficientesNaoAnaliticos[i][4]*pow((delta-1),2)+
                    coeficientesNaoAnaliticos[i][5]*pow((tau-1),2)));
                double mdeltab_t=-2*fi*coeficientesNaoAnaliticos[i][1]*pow(mdelta,
                    coeficientesNaoAnaliticos[i][1]-1);
                double ex_t=-2*coeficientesNaoAnaliticos[i][5]*(tau-1)*ex;

                fResidual_t+=coeficientesNaoAnaliticos[i][3]*delta*(pow(mdelta,
                    coeficientesNaoAnaliticos[i][1])*ex_t+mdeltab_t*ex);
        }
}

void CHelmholtz::FR_tt()
{
        fResidual_tt=0;
        for(int i=0;i<numeroCoeficientes[1];i++){
                fResidual_tt+=coeficientesPolinomio[i][2]*coeficientesPolinomio[i][1]*(
                    coeficientesPolinomio[i][1]-1)*pow(delta,coeficientesPolinomio[i][0])
                    *pow(tau,(coeficientesPolinomio[i][1]-2));
        }
        for(int i=0;i<numeroCoeficientes[2];i++){
                fResidual_tt+=coeficientesExponenciais[i][3]*coeficientesExponenciais[i
                    ][2]*(coeficientesExponenciais[i][2]-1)*pow(delta,
                    coeficientesExponenciais[i][1])*pow(tau,coeficientesExponenciais[i
                    ][2]-2)*exp(-pow(delta,coeficientesExponenciais[i][0]));
        }
        for(int i=0;i<numeroCoeficientes[3];i++){
                double a=coeficientesGaussianos[i][2]*pow(delta,coeficientesGaussianos[i
                    ][0])*pow(tau,coeficientesGaussianos[i][1]);
                double b=exp(-coeficientesGaussianos[i][3]*pow(delta-
                    coeficientesGaussianos[i][6],2)-coeficientesGaussianos[i][4]*pow(tau-
                    coeficientesGaussianos[i][5],2));
                double c=pow((coeficientesGaussianos[i][1]/tau)-2*coeficientesGaussianos[
                    i][4]*(tau-coeficientesGaussianos[i][5]),2)-coeficientesGaussianos[i
                    ][1]*pow(tau,-2)-2*coeficientesGaussianos[i][4];
```

```
                fResidual_tt+=a*b*c;
        }
        for(int i=0;i<numeroCoeficientes[4];i++){
                double fi=(1-tau)+coeficientesNaoAnaliticos[i][6]*pow(pow((delta-1),2)
                    ,(1/(2*coeficientesNaoAnaliticos[i][7])));
                double mdelta=pow(fi,2)+coeficientesNaoAnaliticos[i][2]*pow(pow((delta-1)
                    ,2),coeficientesNaoAnaliticos[i][0]);
                double ex=exp(-(coeficientesNaoAnaliticos[i][4]*pow((delta-1),2)+
                    coeficientesNaoAnaliticos[i][5]*pow((tau-1),2)));
                double mdeltab_t=-2*fi*coeficientesNaoAnaliticos[i][1]*pow(mdelta,
                    coeficientesNaoAnaliticos[i][1]-1);
                double ex_t=-2*coeficientesNaoAnaliticos[i][5]*(tau-1)*ex;
                double ex_tt=(2*coeficientesNaoAnaliticos[i][5]*pow(tau-1,2)-1)*2*
                    coeficientesNaoAnaliticos[i][5]*ex;
                double mdeltab_tt=2*coeficientesNaoAnaliticos[i][1]*pow(mdelta,
                    coeficientesNaoAnaliticos[i][1]-1)+4*pow(fi,2)*
                    coeficientesNaoAnaliticos[i][1]*(coeficientesNaoAnaliticos[i][1]-1)*
                    pow(mdelta,coeficientesNaoAnaliticos[i][1]-2);

                fResidual_tt+=coeficientesNaoAnaliticos[i][3]*delta*(2*mdeltab_t*ex_t+
                    mdeltab_tt*ex+pow(mdelta,coeficientesNaoAnaliticos[i][1])*ex_tt);
        }
}

void CHelmholtz::FR_ttt()
{
        fResidual_ttt=0;
        for(int i=0;i<numeroCoeficientes[1];i++){
                fResidual_ttt+=coeficientesPolinomio[i][2]*coeficientesPolinomio[i][1]*(
                    coeficientesPolinomio[i][1]-1)*(coeficientesPolinomio[i][1]-2)*pow(
                    delta,coeficientesPolinomio[i][0])*pow(tau,coeficientesPolinomio[i
                    ][1]-3);
        }
        for(int i=0;i<numeroCoeficientes[2];i++){
                fResidual_ttt+=coeficientesExponenciais[i][3]*coeficientesExponenciais[i
                    ][2]*(coeficientesExponenciais[i][2]-1)*(coeficientesExponenciais[i
                    ][2]-2)*pow(delta,coeficientesExponenciais[i][1])*pow(tau,
                    coeficientesExponenciais[i][2]-3)*exp(-pow(delta,
                    coeficientesExponenciais[i][0]));
        }
        for(int i=0;i<numeroCoeficientes[3];i++){
                double a=coeficientesGaussianos[i][2]*pow(delta,coeficientesGaussianos[i
                    ][0])*pow(tau,coeficientesGaussianos[i][1]);
                double b=exp(-coeficientesGaussianos[i][3]*pow(delta-
                    coeficientesGaussianos[i][6],2)-coeficientesGaussianos[i][4]*pow(tau-
                    coeficientesGaussianos[i][5],2));
                double c=coeficientesGaussianos[i][1]*pow(tau,-3)+(coeficientesGaussianos
                    [i][1]/tau-2*coeficientesGaussianos[i][4]*(tau-coeficientesGaussianos
                    [i][5]))*(pow(coeficientesGaussianos[i][1]/tau-2*
                    coeficientesGaussianos[i][4]*(tau-coeficientesGaussianos[i][5]),2)
                    -3*(coeficientesGaussianos[i][1]/pow(tau,2)+2*coeficientesGaussianos[
                    i][4]));
                fResidual_ttt+=a*b*c;//erro
}
        for(int i=0;i<numeroCoeficientes[4];i++){
                double fi=(1-tau)+coeficientesNaoAnaliticos[i][6]*pow(pow((delta-1),2)
                    ,(1/(2*coeficientesNaoAnaliticos[i][7])));
                double mdelta=pow(fi,2)+coeficientesNaoAnaliticos[i][2]*pow(pow((delta-1)
                    ,2),coeficientesNaoAnaliticos[i][0]);
                double ex=exp(-(coeficientesNaoAnaliticos[i][4]*pow((delta-1),2)+
```

```
                    coeficientesNaoAnaliticos[i][5]*pow((tau-1),2)));
                double mdeltab_t=-2*fi*coeficientesNaoAnaliticos[i][1]*pow(mdelta,
                    coeficientesNaoAnaliticos[i][1]-1);
                double ex_t=-2*coeficientesNaoAnaliticos[i][5]*(tau-1)*ex;
                double ex_tt=(2*coeficientesNaoAnaliticos[i][5]*pow(tau-1,2)-1)*2*
                    coeficientesNaoAnaliticos[i][5]*ex;
                double mdeltab_tt=2*coeficientesNaoAnaliticos[i][1]*pow(mdelta,
                    coeficientesNaoAnaliticos[i][1]-1)+4*pow(fi,2)*
                    coeficientesNaoAnaliticos[i][1]*(coeficientesNaoAnaliticos[i][1]-1)*
                    pow(mdelta,coeficientesNaoAnaliticos[i][1]-2);
                double mdeltab_ttt=-12*coeficientesNaoAnaliticos[i][1]*(
                    coeficientesNaoAnaliticos[i][1]-1)*fi*pow(mdelta,
                    coeficientesNaoAnaliticos[i][1]-2)-8*coeficientesNaoAnaliticos[i
                    ][1]*(coeficientesNaoAnaliticos[i][1]-1)*(coeficientesNaoAnaliticos[i
                    ][1]-2)*pow(fi,3)*pow(mdelta,coeficientesNaoAnaliticos[i][1]-3);
                double ex_ttt=2*coeficientesNaoAnaliticos[i][5]*ex_t*(2*
                    coeficientesNaoAnaliticos[i][5]*pow(tau-1,2)-1)+8*pow(
                    coeficientesNaoAnaliticos[i][5],2)*(tau-1)*ex;
                fResidual_ttt+=coeficientesNaoAnaliticos[i][3]*delta*(mdeltab_ttt*ex+3*
                    mdeltab_tt*ex_t+3*mdeltab_t*ex_tt+pow(mdelta,
                    coeficientesNaoAnaliticos[i][1])*ex_ttt);
        }
}

void CHelmholtz::FI()
{
        fIdeal=log(delta)+coeficientesIdeais1[0]+coeficientesIdeais1[1]*tau+
            coeficientesIdeais1[2]*log(tau);
        for(int i=0;i<numeroCoeficientes[0];i++){
                fIdeal+=coeficientesIdeais2[i][0]*log(1-exp(-coeficientesIdeais2[i][1]*
                    tau));
        }
        if(tCritica==126.192)fIdeal+=coeficientesIdeais1[3]/tau+coeficientesIdeais1[4]/
            pow(tau,2)+coeficientesIdeais1[5]/pow(tau,3);
}

void CHelmholtz::FI_t()
{
        fIdeal_t=coeficientesIdeais1[1]+(coeficientesIdeais1[2]/tau);
        for(int i=0;i<numeroCoeficientes[0];i++){
                fIdeal_t+=coeficientesIdeais2[i][0]*coeficientesIdeais2[i][1]*(pow((1-exp
                    (-coeficientesIdeais2[i][1]*tau)),-1)-1);
        }
        if(tCritica==126.192)fIdeal_t-=coeficientesIdeais1[3]/pow(tau,2)+2*
            coeficientesIdeais1[4]/pow(tau,3)+3*coeficientesIdeais1[5]/pow(tau,4);
}

void CHelmholtz::FI_tt()
{
        fIdeal_tt=-coeficientesIdeais1[2]*pow(tau,-2);
        for(int i=0;i<numeroCoeficientes[0];i++){
                fIdeal_tt-=coeficientesIdeais2[i][0]*pow(coeficientesIdeais2[i][1],2)*exp
                    (-coeficientesIdeais2[i][1]*tau)*pow(1-exp(-coeficientesIdeais2[i
                    ][1]*tau),-2);
        }
        if(tCritica==126.192)fIdeal_tt+=2*coeficientesIdeais1[3]/pow(tau,3)+6*
            coeficientesIdeais1[4]/pow(tau,4)+12*coeficientesIdeais1[5]/pow(tau,5);
}

void CHelmholtz::FI_ttt()
```

```
{
        fIdeal_ttt=2*coeficientesIdeais1[2]*pow(tau,-3);
        for(int i=0;i<numeroCoeficientes[0];i++){
                fIdeal_ttt+=coeficientesIdeais2[i][0]*pow(coeficientesIdeais2[i][1],3)*
                    exp(-coeficientesIdeais2[i][1]*tau)*pow(1-exp(-coeficientesIdeais2[i
                    ][1]*tau),-2)*(1+2*exp(-coeficientesIdeais2[i][1]*tau)*(1-exp(-
                    coeficientesIdeais2[i][1]*tau)));
        }
        if(tCritica==126.192)fIdeal_tt-=6*coeficientesIdeais1[3]/pow(tau,4)+24*
            coeficientesIdeais1[4]/pow(tau,5)+60*coeficientesIdeais1[5]/pow(tau,6);
}
```

Apresenta-se na listagem 6.5 o arquivo com código da classe `CSubstancia`.

Listing 6.5: Arquivo de cabeçalho da classe CSubstancia.

```
#ifndef CSUBSTANCIA_H
#define CSUBSTANCIA_H
#include <ostream>
#include <string>

/// @struct SPressao
/// @brief Struct com os valores relacionados à pressão.
struct SPressao
{
    /// Valor da pressão.
        double valor;

    /// Derivada da pressão em relação à densidade.
    double d;

    /// Derivada da pressão em relação à temperatura.
    double t;
};

/// @struct SEntalpia
/// @brief Struct com os valores relacionados à entalpia.
struct SEntalpia
{
    ///Valor da entalpia.
        double valor;

    /// Derivada da entalpia em relação à densidade.
    double d;

    /// Derivada da entalpia em relação à temperatura.
    double t;

    /// Derivada da entalpia em relação à pressão.
    double p;
};

/// @struct SEntropia
/// @brief Struct com os valores relacionados à entropia.
struct SEntropia
{
    ///Valor da entropia.
    double valor;

    /// Derivada da entalpia em relação à densidade.
    double d;
```

```
    /// Derivada da entalpia em relação à temperatura.
    double t;

    /// Derivada da entalpia em relação à pressão.
    double p;
};

/// @struct SEnergiaInterna
/// @brief Struct com os valores relacionados à energia interna.
struct SEnergiaInterna
{
    ///Valor da energia interna.
    double valor;

    /// Derivada da entalpia em relação à densidade.
    double d;

    /// Derivada da entalpia em relação à temperatura.
    double t;

    /// Derivada da entalpia em relação à pressão.
    double p;
};

/// @struct SEnergiaGibbs
/// @brief Struct com os valores relacionados à energia de Gibbs.
struct SEnergiaGibbs
{
    ///Valor da energia de Gibbs.
    double valor;

    /// Derivada da entalpia em relação à densidade.
    double d;

    /// Derivada da entalpia em relação à temperatura.
    double t;

    /// Derivada da entalpia em relação à pressão.
    double p;
};

/// @struct SCapCalorificaIsobarica
/// @brief Struct com os valores relacionados à capacidade calorífica isobárica.
struct SCapCalorificaIsobarica
{
    ///Valor da capacidade calorífica isobárica.
    double valor;

    /// Derivada da entalpia em relação à densidade.
    double d;

    /// Derivada da entalpia em relação à temperatura.
    double t;

    /// Derivada da entalpia em relação à pressão.
    double p;
};

/// @struct SCapCalorificaIsovolumetrica
```

```
/// @brief Struct com os valores relacionados à capacidade calorífica isovolumétrica.
struct SCapCalorificaIsovolumetrica
{
    ///Valor da capacidade calorífica isovolumétrica.
    double valor;

    /// Derivada da entalpia em relação à densidade.
    double d;

    /// Derivada da entalpia em relação à temperatura.
    double t;

    /// Derivada da entalpia em relação à pressão.
    double p;
};

/// @struct SCoefJouleThomson
/// @brief Struct com os valores relacionados ao coeficiente de Joule-Thomson.
struct SCoefJouleThomson
{
    ///Valor do coeficiente de Joule-Thomson.
    double valor;

    /// Derivada da entalpia em relação à densidade.
    double d;

    /// Derivada da entalpia em relação à temperatura.
    double t;

    /// Derivada da entalpia em relação à pressão.
    double p;
};

/// @struct SCoefEnforcamento
/// @brief Struct com os valores relacionados ao coeficiente de enforcamento.
struct SCoefEnforcamento
{
    ///Valor do coeficiente de enforcamento.
    double valor;

    /// Derivada da entalpia em relação à densidade.
    double d;

    /// Derivada da entalpia em relação à temperatura.
    double t;

    /// Derivada da entalpia em relação à pressão.
    double p;
};

/// @struct SVelocidadeSom
/// @brief Struct com os valores relacionados à velocidade do som no meio.
struct SVelocidadeSom
{
    ///Valor da velocidade do som no meio.
    double valor;

    /// Derivada da entalpia em relação à densidade.
    double d;
```

```
    /// Derivada da entalpia em relação à temperatura.
    double t;

    /// Derivada da entalpia em relação à pressão.
    double p;
};

/// @struct SIsentropico
/// @brief Struct com os valores relacionados ao coeficiente de temperatura-pressão
    isentrópico.
struct SIsentropico
{
    ///Valor do coeficiente de temperatura-pressão isentrópico.
    double valor;

    /// Derivada da entalpia em relação à densidade.
    double d;

    /// Derivada da entalpia em relação à temperatura.
    double t;

    /// Derivada da entalpia em relação à pressão.
    double p;
};

////////////////////////////////////////////////////////////////////////
/// @author Thomas Augusto Menegazzo
/// @class CSubstancia
/// @file CSubstancia.h
/// @brief Classe que armazena os valores calculados.
////////////////////////////////////////////////////////////////////////
class CSubstancia
{
        public:
                ///Temperatura da substância.
                double temperatura;

                ///Densidade da substância.
                double densidade;

                ///Constante especifica dos gases.
                double R;

                ///Derivada da densidade com relacao à pressão.
                double d_p;

                ///Massa molar.
                double massaMolar;

                ///Formula química da substância.
                std::string nome;

                ///Struct para armazenamento da pressão.
                SPressao pressao;

                ///Struct para armazenamento da entalpia.
                SEntalpia entalpia;

                ///Struct para armazenamento da entropia.
                SEntropia entropia;
```

```
                ///Struct para armazenamento da energia interna.
                SEnergiaInterna energiaInterna;

                ///Struct para armazenamento da energia de Gibbs.
                SEnergiaGibbs energiaGibbs;

                ///Struct para armazenamento da capacidade calorífica isobárica.
                SCapCalorificaIsobarica capCalorificaIsobarica;

                ///Struct para armazenamento da capacidade calorífica isocórica.
                SCapCalorificaIsovolumetrica capCalorificaIsovolumetrica;

                ///Struct para armazenamento do coeficiente de Joule-Thomson.
                SCoefJouleThomson coefJouleThomson;

                ///Struct para armazenamento do coeficiente de enforcamento.
                SCoefEnforcamento coefEnforcamento;

                ///Struct para armazenamento da velocidade do som.
                SVelocidadeSom velocidadeSom;

                ///Struct para armazenamento do coeficiente de temperatura-pressão
                    isentrópico.
                SIsentropico coefIsentropico;
        public:

                ///Construtor da classe.
                CSubstancia();

                ///Seta o valor da temperatura.
                ///@param t Temperatura.
                void SetTemperatura(double t){temperatura=t;}

                ///Seta o valor da densidade
                ///@param d Densidade.
                void SetDensidade(double d){densidade=d;}

                ///Seta o valor da pressão.
                ///@param p pressão.
                void SetPressao(double p){pressao.valor=p;}

                ///função para mostrar os dados calculados na tela
                void Display();
                };
#endif
```

Apresenta-se na listagem 6.6 o arquivo de implementação da classe `CSubstancia`.

Listing 6.6: Arquivo de implementação da classe CSubstancia.

```
#include "CSubstancia.h"
#include <iostream>
void CSubstancia::Display()
{
        if (pressao.valor!=0){
                std::cout<<"**************************************";
                std::cout<<std::endl<<"Pressao:␣"<<pressao.valor;
                if(pressao.t!=0.0){
                std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣temperatura␣=␣"<<pressao.t
                    ;}
```

```
        if(pressao.d!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣densidade␣=␣"<<pressao.d;}
        std::cout<<std::endl;
}
if (entalpia.valor!=0){
        std::cout<<"************************************";
        std::cout<<std::endl<<"Entalpia:␣"<<entalpia.valor;
        if(entalpia.t!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣temperatura␣=␣"<<entalpia.t
            ;}
        if(entalpia.d!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣densidade␣=␣"<<entalpia.d;}
        if(entalpia.p!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣pressao␣=␣"<<entalpia.p;}
        std::cout<<std::endl;
}
if (entropia.valor!=0){
        std::cout<<"************************************";
        std::cout<<std::endl<<"Entropia:␣"<<entropia.valor;
        if(entropia.t!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣temperatura␣=␣"<<entropia.t
            ;}
        if(entropia.d!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣densidade␣=␣"<<entropia.d;}
        if(entropia.p!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣pressao␣=␣"<<entropia.p;}
        std::cout<<std::endl;
}
if (energiaInterna.valor!=0){
        std::cout<<"************************************";
        std::cout<<std::endl<<"Energia␣Interna:␣"<<energiaInterna.valor;
        if(energiaInterna.t!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣temperatura␣=␣"<<
            energiaInterna.t;}
        if(energiaInterna.d!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣densidade␣=␣"<<
            energiaInterna.d;}
        if(energiaInterna.p!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣pressao␣=␣"<<energiaInterna
            .p;}
        std::cout<<std::endl;
}
if (energiaGibbs.valor!=0){
        std::cout<<"************************************";
        std::cout<<std::endl<<"Energia␣de␣Gibbs:␣"<<energiaGibbs.valor;
        if(energiaGibbs.t!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣temperatura␣=␣"<<
            energiaGibbs.t;}
        if(energiaGibbs.d!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣densidade␣=␣"<<energiaGibbs
            .d;}
        if(energiaGibbs.p!=0.0){
        std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣pressao␣=␣"<<energiaGibbs.p
            ;}
        std::cout<<std::endl;
}
if (capCalorificaIsobarica.valor!=0){
        std::cout<<"************************************";
        std::cout<<std::endl<<"Capacidade␣Calorifica␣Isobarica:␣"<<
            capCalorificaIsobarica.valor;
```

```
            if(capCalorificaIsobarica.t!=0.0){
            std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣temperatura␣=␣"<<
                capCalorificaIsobarica.t;}
            if(capCalorificaIsobarica.d!=0.0){
            std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣densidade␣=␣"<<
                capCalorificaIsobarica.d;}
            if(capCalorificaIsobarica.p!=0.0){
            std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣pressao␣=␣"<<
                capCalorificaIsobarica.p;}
            std::cout<<std::endl;
        }
        if (capCalorificaIsovolumetrica.valor!=0){
            std::cout<<"************************************";
            std::cout<<std::endl<<"Capacidade␣Calorifica␣Isovolumetrica:␣"<<
                capCalorificaIsovolumetrica.valor;
            if(capCalorificaIsovolumetrica.t!=0.0){
            std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣temperatura␣=␣"<<
                capCalorificaIsovolumetrica.t;}
            if(capCalorificaIsovolumetrica.d!=0.0){
            std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣densidade␣=␣"<<
                capCalorificaIsovolumetrica.d;}
            if(capCalorificaIsovolumetrica.p!=0.0){
            std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣pressao␣=␣"<<
                capCalorificaIsovolumetrica.p;}
            std::cout<<std::endl;
        }
        if (coefJouleThomson.valor!=0){
            std::cout<<"************************************";
            std::cout<<std::endl<<"Coeficiente␣de␣Joule-Thomson:␣"<<coefJouleThomson.
                valor;
            if(coefJouleThomson.t!=0.0){
            std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣temperatura␣=␣"<<
                coefJouleThomson.t;}
            if(coefJouleThomson.d!=0.0){
            std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣densidade␣=␣"<<
                coefJouleThomson.d;}
            if(coefJouleThomson.p!=0.0){
            std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣pressao␣=␣"<<
                coefJouleThomson.p;}
            std::cout<<std::endl;
        }
        if (coefEnforcamento.valor!=0){
            std::cout<<"************************************";
            std::cout<<std::endl<<"Coeficiente␣de␣Enforcamento:␣"<<coefEnforcamento.
                valor;
            if(coefEnforcamento.t!=0){
            std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣temperatura␣=␣"<<
                coefEnforcamento.t;}
            if(coefEnforcamento.d!=0){
            std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣densidade␣=␣"<<
                coefEnforcamento.d;}
            if(coefEnforcamento.p!=0){
            std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣pressao␣=␣"<<
                coefEnforcamento.p;}
            std::cout<<std::endl;
        }
        if (velocidadeSom.valor!=0){
            std::cout<<"************************************";
            std::cout<<std::endl<<"Velocidade␣do␣Som:␣"<<velocidadeSom.valor;
            if(velocidadeSom.t!=0){
```

```
                std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣temperatura␣=␣"<<
                    velocidadeSom.t;}
                if(velocidadeSom.d!=0){
                std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣densidade␣=␣"<<
                    velocidadeSom.d;}
                if(velocidadeSom.p!=0){
                std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣pressao␣=␣"<<velocidadeSom.
                    p;}
                std::cout<<std::endl;
        }
                if (coefIsentropico.valor!=0){
                std::cout<<"************************************";
                std::cout<<std::endl<<"Coeficiente␣de␣pressao-temperatura␣isentropico:␣"
                    <<coefIsentropico.valor;
                if(coefIsentropico.t!=0){
                std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣temperatura␣=␣"<<
                    coefIsentropico.t;}
                if(coefIsentropico.d!=0){
                std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣densidade␣=␣"<<
                    coefIsentropico.d;}
                if(coefIsentropico.p!=0){
                std::cout<<std::endl<<"Derivada␣com␣relacao␣a␣pressao␣=␣"<<
                    coefIsentropico.p;}
                std::cout<<std::endl;
        }
        std::cout<<"************************************";
        std::cout<<std::endl<<std::endl;
}
CSubstancia::CSubstancia()
{
                nome="␣";
                temperatura=0;
                densidade=0;
                R=0;
                d_p=0;
                pressao.valor=0;
                pressao.d=0;
                pressao.t=0;
                entalpia.valor=0;
                entalpia.d=0;
                entalpia.p=0;
                entalpia.t=0;
                entropia.valor=0;
                entropia.d=0;
                entropia.t=0;
                entropia.p=0;
                energiaInterna.valor=0;
                energiaInterna.d=0;
                energiaInterna.t=0;
                energiaInterna.p=0;
                energiaGibbs.valor=0;
                energiaGibbs.d=0;
                energiaGibbs.t=0;
                energiaGibbs.p=0;
                capCalorificaIsobarica.valor=0;
                capCalorificaIsobarica.d=0;
                capCalorificaIsobarica.t=0;
                capCalorificaIsobarica.p=0;
                capCalorificaIsovolumetrica.valor=0;
                capCalorificaIsovolumetrica.d=0;
```

```
                capCalorificaIsovolumetrica.t=0;
                capCalorificaIsovolumetrica.p=0;
                coefJouleThomson.valor=0;
                coefJouleThomson.d=0;
                coefJouleThomson.t=0;
                coefJouleThomson.p=0;
                coefEnforcamento.valor=0;
                coefEnforcamento.d=0;
                coefEnforcamento.t=0;
                coefEnforcamento.p=0;
                velocidadeSom.valor=0;
                velocidadeSom.d=0;
                velocidadeSom.p=0;
                velocidadeSom.t=0;
                coefIsentropico.valor=0;
                coefIsentropico.t=0;
                coefIsentropico.d=0;
                coefIsentropico.p=0;
                massaMolar=0;
}
```

Apresenta-se na listagem 6.7 o arquivo com código das classes derivadas de `CHelmholtz`.

Listing 6.7: Arquivo de cabeçalho das classes derivadas de CHelmholtz.

```
#include "CHelmholtz.h"
////////////////////////////////////////////////////////////////////////
/// @author Thomas Augusto Menegazzo
/// @class CHelmholtzAgua
/// @file CHelmholtzAgua.h
/// @brief Classe para passar os coeficientes da água para CHelmholtz.
////////////////////////////////////////////////////////////////////////
class CHelmholtzAgua: public CHelmholtz{
        public:
                CHelmholtzAgua();
                ~CHelmholtzAgua();
};

/// @class CHelmholtzMetano
/// @file CHelmholtzAgua.h
/// @brief Classe para passar os coeficientes do metano para CHelmholtz.
class CHelmholtzMetano: public CHelmholtz{
        public:
                CHelmholtzMetano();
                ~CHelmholtzMetano();
};

/// @class CHelmholtzEtano
/// @file CHelmholtzAgua.h
/// @brief Classe para passar os coeficientes do etano para CHelmholtz.
class CHelmholtzEtano: public CHelmholtz{
        public:
                CHelmholtzEtano();
                ~CHelmholtzEtano();
};

/// @class CHelmholtzIsobutano
/// @file CHelmholtzAgua.h
/// @brief Classe para passar os coeficientes do isobutano para CHelmholtz.
class CHelmholtzIsobutano: public CHelmholtz{
        public:
```

```
                CHelmholtzIsobutano();
                ~CHelmholtzIsobutano();
};

/// @class CHelmholtzNbutano
/// @file CHelmholtzAgua.h
/// @brief Classe para passar os coeficientes do n-butano para CHelmholtz.
class CHelmholtzNbutano: public CHelmholtz{
        public:
                CHelmholtzNbutano();
                ~CHelmholtzNbutano();
};

/// @class CHelmholtzDioxidoCarbono
/// @file CHelmholtzAgua.h
/// @brief Classe para passar os coeficientes do dioxido de carbono para CHelmholtz.
class CHelmholtzDioxidoCarbono: public CHelmholtz{
        public:
                CHelmholtzDioxidoCarbono();
                ~CHelmholtzDioxidoCarbono();
};

/// @class CHelmholtzNitrogenio
/// @file CHelmholtzAgua.h
/// @brief Classe para passar os coeficientes do nitrogenio para CHelmholtz.
class CHelmholtzNitrogenio: public CHelmholtz{
        public:
                CHelmholtzNitrogenio();
                ~CHelmholtzNitrogenio();
};
```

Apresenta-se na listagem 6.8 o arquivo de implementação da classe `CHelmholtzAgua`.

Listing 6.8: Arquivo de implementação da classe CHelmholtzAgua.

```
#include "CHelmholtzAgua.h"
CHelmholtzAgua::CHelmholtzAgua()
{
    tCritica=647.096;
    dCritica=322;
    rEspecifico=0.46151805;
    int a[5]={5,7,44,3,2};
    for(int i=0;i<5;i++)numeroCoeficientes[i]=a[i];
        double b[3]={-8.32044648201,6.6832105268,3.00632};
        coeficientesIdeais1 = new double[3];
        for(int i=0;i<3;i++)coeficientesIdeais1[i]=b[i];
        double c[5][2]={
        {0.012436,1.28728967},
    {0.97315,3.53734222},
    {1.27950,7.74073708},
    {0.96956,9.24437796},
    {0.24873,27.5075105}
        };
        coeficientesIdeais2=Aloca(numeroCoeficientes[0],2);
        for(int i=0;i<5;i++){
                for(int j=0;j<2;j++)coeficientesIdeais2[i][j]=c[i][j];
        }
        double d[7][3]={
        {1,-0.5,0.12533547935523E-1},
    {1,0.875,0.78957634722828E1},
    {1,1,-0.87803203303561E1},
```

```
{2,0.5,0.31802509345418},
{2,0.75,-0.26145533859358},
{3,0.375,-0.78199751687981E-2},
{4,1,0.88089493102134E-2},
    };
    coeficientesPolinomio=Aloca(numeroCoeficientes[1],3);
    for(int i=0;i<7;i++){
            for(int j=0;j<3;j++)coeficientesPolinomio[i][j]=d[i][j];
    }
    double e[44][4]={
            {1,1,4,-0.66856572307965},
            {1,1,6,0.20433810950965},
            {1,1,12,-0.66212605039687E-4},
            {1,2,1,-0.19232721156002},
            {1,2,5,-0.25709043003438},
            {1,3,4,0.16074868486251},
            {1,4,2,-0.40092828925807E-1},
            {1,4,13,0.39343422603254E-6},
            {1,5,9,-0.75941377088144E-5},
            {1,7,3,0.56250979351888E-3},
            {1,9,4,-0.15608652257135E-4},
            {1,10,11,0.11537996422951E-8},
            {1,11,4,0.36582165144204E-6},
            {1,13,13,-0.13251180074668E-11},
            {1,15,1,-0.62639586912454E-9},
            {2,1,7,-0.10793600908932},
            {2,2,1,0.17611491008752E-1},
            {2,2,9,0.22132295167546},
            {2,2,10,-0.40247669763528},
            {2,3,10,0.58083399985759},
            {2,4,3,0.49969146990806E-2},
            {2,4,7,-0.31358700712549E-1},
            {2,4,10,-0.74315929710341},
            {2,5,10,0.47807329915480},
            {2,6,6,0.20527940895948E-1},
            {2,6,10,-0.13636435110343},
            {2,7,10,0.14180634400617E-1},
            {2,9,1,0.83326504880713E-2},
            {2,9,2,-0.29052336009585E-1},
            {2,9,3,0.38615085574206E-1},
            {2,9,4,-0.20393486513704E-1},
            {2,9,8,-0.16554050063734E-2},
            {2,10,6,0.19955571979541E-2},
            {2,10,9,0.15870308324157E-3},
            {2,12,8,-0.16388568342530E-4},
            {3,3,16,0.43613615723811E-1},
            {3,4,22,0.34994005463765E-1},
            {3,4,23,-0.76788197844621E-1},
            {3,5,23,0.22446277332006E-1},
            {4,14,10,-0.62689710414685E-4},
            {6,3,50,-0.55711118565645E-9},
            {6,6,44,-0.19905718354408},
            {6,6,46,0.31777497330738},
            {6,6,50,-0.11841182425981}};
    coeficientesExponenciais=Aloca(numeroCoeficientes[2],4);
    for(int i=0;i<44;i++){
            for(int j=0;j<4;j++)coeficientesExponenciais[i][j]=e[i][j];
    }
    double f[3][7]={
            {3,0,-0.31306260323435E2,20,150,1.21,1},
```

```
                {3,1,0.31546140237781E2,20,150,1.21,1},
                {3,4,-0.25213154341695E4,20,250,1.25,1}};
        coeficientesGaussianos=Aloca(numeroCoeficientes[3],7);
        for(int i=0;i<3;i++){
                for(int j=0;j<7;j++)coeficientesGaussianos[i][j]=f[i][j];
        }
        double g[2][8]={
                {3.5,0.85,0.2,-0.14874640856724,28,700,0.32,0.3},
                {3.5,0.95,0.2,0.31806110878444,32,800,0.32,0.3}};
        coeficientesNaoAnaliticos=Aloca(numeroCoeficientes[4],8);
        for(int i=0;i<2;i++){
                for(int j=0;j<8;j++)coeficientesNaoAnaliticos[i][j]=g[i][j];
        }
        fIdeal=0;
        fIdeal_t=0;
        fIdeal_tt=0;
        fIdeal_ttt=0;
        fResidual=0;
        fResidual_t=0;
        fResidual_d=0;
        fResidual_dd=0;
        fResidual_dt=0;
        fResidual_tt=0;
        fResidual_ddd=0;
        fResidual_ddt=0;
        fResidual_dtt=0;
        fResidual_ttt=0;
}
```

Apresenta-se na listagem 6.9 o arquivo de implementação da classe `CHelmholtzDioxidoCarbono`.

Listing 6.9: Arquivo de implementação da classe CHelmholtzDioxidoCarbono.

```
#include "CHelmholtzAgua.h"
CHelmholtzDioxidoCarbono::CHelmholtzDioxidoCarbono()
{
        tCritica=304.1282;
        dCritica=467.6;
        rEspecifico=0.1889241;
        int a[5]={5,7,27,5,3};
        for(int i=0;i<5;i++)numeroCoeficientes[i]=a[i];
        double b[3]={8.37304456,-3.70454304,2.50000000};
        coeficientesIdeais1 = new double[3];
        for(int i=0;i<3;i++)coeficientesIdeais1[i]=b[i];
        double c[5][2]={
        {1.99427042,3.15163},
        {0.62105248,6.11190},
        {0.41195293,6.77708},
        {1.04028922,11.32384},
        {0.08327678,27.08792}};
        coeficientesIdeais2=Aloca(numeroCoeficientes[0],2);
        for(int i=0;i<numeroCoeficientes[0];i++){
                for(int j=0;j<2;j++)coeficientesIdeais2[i][j]=c[i][j];
        }
        double d[7][3]={
        {1,0,0.38856823203161},
        {1,0.75,0.29385475942740E1},
        {1,1,-0.55867188534934E1},
        {1,2,-0.76753199592477},
        {2,0.75,0.31729005580416},
        {2,2,0.54803315897767},
```

```
{3,0.75,0.12279411220335}};
coeficientesPolinomio=Aloca(numeroCoeficientes[1],3);
for(int i=0;i<numeroCoeficientes[1];i++){
        for(int j=0;j<3;j++)coeficientesPolinomio[i][j]=d[i][j];
}
double e[27][4]={
{1,1,1.5,0.21658961543220E1},
{1,2,1.5,0.15841735109724E1},
{1,4,2.5,-0.23132705405503},
{1,5,0,0.58116916431436E-1},
{1,5,1.5,-0.55369137205382},
{1,5,2,0.48946615909422},
{1,6,0,-0.24275739843501E-1},
{1,6,1,0.62494790501678E-1},
{1,6,2,-0.12175860225246},
{2,1,3,-0.37055685270086},
{2,1,6,-0.16775879700426E-1},
{2,4,3,-0.11960736637987},
{2,4,6,-0.45619362508778E-1},
{2,4,8,0.35612789270346E-1},
{2,7,6,-0.74427727132052E-2},
{2,8,0,-0.17395704902432E-2},
{3,2,7,-0.21810121289527E-1},
{3,3,12,0.24332166559236E-1},
{3,3,16,-0.37440133423463E-1},
{4,5,22,0.14338715756878},
{4,5,24,-0.13491969083286},
{4,6,16,-0.23151225053480E-1},
{4,7,24,0.12363125492901E-1},
{4,8,8,0.21058321972940E-2},
{4,10,2,-0.33958519026368E-3},
{5,4,28,0.55993651771592E-2},
{6,8,14,-0.30335118055646E-3}};
coeficientesExponenciais=Aloca(numeroCoeficientes[2],4);
for(int i=0;i<numeroCoeficientes[2];i++){
        for(int j=0;j<4;j++)coeficientesExponenciais[i][j]=e[i][j];
}
double f[5][7]={
        {2,1,-0.21365488688320E3,25,325,1.16,1},
        {2,0,0.26641569149272E5,25,300,1.19,1},
        {2,1,-0.24027212204557E5,25,300,1.19,1},
        {3,3,-0.28341603423999E3,15,275,1.25,1},
        {3,3,0.21247284400179E3,20,275,1.22,1}};
coeficientesGaussianos=Aloca(numeroCoeficientes[3],7);
for(int i=0;i<numeroCoeficientes[3];i++){
        for(int j=0;j<7;j++)coeficientesGaussianos[i][j]=f[i][j];
}
double g[3][8]={
        {3.5,0.875,0.3,-0.66642276540751,10,275,0.7,0.3},
        {3.5,0.925,0.3,0.72608632349897,10,275,0.7,0.3},
        {3,0.875,1,0.55068668612842E-1,12.5,275,0.7,0.3}};
coeficientesNaoAnaliticos=Aloca(numeroCoeficientes[4],8);
for(int i=0;i<numeroCoeficientes[4];i++){
        for(int j=0;j<8;j++)coeficientesNaoAnaliticos[i][j]=g[i][j];
}
fIdeal=0;
fIdeal_t=0;
fIdeal_tt=0;
fIdeal_ttt=0;
fResidual=0;
```

```
        fResidual_t=0;
        fResidual_d=0;
        fResidual_dd=0;
        fResidual_dt=0;
        fResidual_tt=0;
        fResidual_ddd=0;
        fResidual_ddt=0;
        fResidual_dtt=0;
        fResidual_ttt=0;
}
```

Apresenta-se na listagem 6.10 o arquivo de implementação da classe `CHelmholtzEtano`.

Listing 6.10: Arquivo de implementação da classe CHelmholtzEtano.

```
#include "CHelmholtzAgua.h"
CHelmholtzEtano::CHelmholtzEtano()
{
        tCritica=305.322;
        dCritica=206.18;
        rEspecifico=0.27651272;
        int a[5]={4,5,34,5,0};
        for(int i=0;i<5;i++)numeroCoeficientes[i]=a[i];
        double b[3]={9.212802589,-4.682248550,3.003039265};
        coeficientesIdeais1 = new double[3];
        for(int i=0;i<3;i++)coeficientesIdeais1[i]=b[i];
        double c[4][2]={
        {1.117433359,1.4091052332},
    {3.467773215,4.0099170712},
    {6.941944640,6.5967098342},
    {5.970850948,13.9798102659}};
        coeficientesIdeais2=Aloca(numeroCoeficientes[0],2);
        for(int i=0;i<4;i++){
                for(int j=0;j<2;j++)coeficientesIdeais2[i][j]=c[i][j];
        }
        double d[5][3]={
        {1,0.25,0.83440745735241},
        {1,1,-0.14287360607171E1},
        {2,0.25,0.34430242210927},
        {2,0.75,-0.42096677920265},
        {4,0.75,0.12094500886549E-1}};
        coeficientesPolinomio=Aloca(numeroCoeficientes[1],3);
        for(int i=0;i<5;i++){
                for(int j=0;j<3;j++)coeficientesPolinomio[i][j]=d[i][j];
        }
        double e[34][4]={
        {1,1,2,-0.57976201597341},
        {1,1,4.25,-0.33127037870838E-1},
        {1,2,0.75,-0.11751654894130},
        {1,2,2.25,-0.11160957833067},
        {1,3,3,0.62181592654406E-1},
        {1,6,1,0.98481795434443E-1},
        {1,6,1.25,-0.98268582682358E-1},
        {1,7,2.75,-0.23977831007049E-3},
        {1,9,1,0.69885663328821E-3},
        {1,10,2,0.19665987803305E-4},
        {2,2,2.5,-0.14586152207928E-1},
        {2,4,5.5,0.46354100536781E-1},
        {2,4,7,0.60764622180645E-2},
        {2,5,0.5,-0.26447330147828E-2},
        {2,5,5.5,-0.42931872689904E-1},
```

```
        {2,6,2.5,0.29987786517263E-2},
        {2,8,4,0.52919335175010E-2},
        {2,9,2,-0.10383897798198E-2},
        {3,2,10,-0.54260348214694E-1},
        {3,3,16,-0.21959362918493},
        {3,3,18,0.35362456650354},
        {3,3,20,-0.12477390173714},
        {3,4,14,0.18425693591517},
        {3,4,18,-0.16192256436754},
        {3,5,12,-0.82770876149064E-1},
        {3,5,19,0.50160758096437E-1},
        {3,6,7,0.93614326336655E-2},
        {3,11,15,-0.27839186242864E-3},
        {3,14,9,0.23560274071481E-4},
        {4,3,26,0.39238329738527E-2},
        {4,3,28,-0.76488325813618E-3},
        {4,4,28,-0.49944304440730E-2},
        {4,8,22,0.18593386407186E-2},
        {4,10,13,-0.61404353331199E-3}};
        coeficientesExponenciais=Aloca(numeroCoeficientes[2],4);
        for(int i=0;i<34;i++){
                for(int j=0;j<4;j++)coeficientesExponenciais[i][j]=e[i][j];
        }
        double f[5][7]={
                {1,0,-0.23312179367924E-2,15,150,1.05,1},
                {1,3,0.29301047908760E-2,15,150,1.05,1},
                {3,3,-0.26912472842883E-3,15,150,1.05,1},
                {3,0,0.18413834111814E3,20,275,1.22,1},
                {2,3,-0.10397127984854E2,20,400,1.16,1}};
        coeficientesGaussianos=Aloca(numeroCoeficientes[3],7);
        for(int i=0;i<5;i++){
                for(int j=0;j<7;j++)coeficientesGaussianos[i][j]=f[i][j];
        }
        fIdeal=0;
        fIdeal_t=0;
        fIdeal_tt=0;
        fIdeal_ttt=0;
        fResidual=0;
        fResidual_t=0;
        fResidual_d=0;
        fResidual_dd=0;
        fResidual_dt=0;
        fResidual_tt=0;
        fResidual_ddd=0;
        fResidual_ddt=0;
        fResidual_dtt=0;
        fResidual_ttt=0;
}
```

Apresenta-se na listagem 6.11 o arquivo de implementação da classe `CHelmholtzIsobutano`.

Listing 6.11: Arquivo de implementação da classe CHelmholtzIsobutano.

```
#include "CHelmholtzAgua.h"
CHelmholtzIsobutano::CHelmholtzIsobutano()
{
        tCritica=407.81;
        dCritica=225.5;
        rEspecifico=0.14305157;
        int a[5]={4,7,16,2,0};
        for(int i=0;i<5;i++)numeroCoeficientes[i]=a[i];
```

```
double b[3]={11.60865546,-5.29450411,3.05956619};
coeficientesIdeais1 = new double[3];
for(int i=0;i<3;i++)coeficientesIdeais1[i]=b[i];
double c[4][2]={
{4.94641014,0.9512779015},
{4.09475197,2.3878958853},
{15.6632824,4.3469042691},
{9.73918122,10.3688586351}};
coeficientesIdeais2=Aloca(numeroCoeficientes[0],2);
for(int i=0;i<numeroCoeficientes[0];i++){
        for(int j=0;j<2;j++)coeficientesIdeais2[i][j]=c[i][j];
}
double d[7][3]={
{1,0.5,0.20686820727966E1},
{1,1,-0.36400098615204E1},
{1,1.5,0.51968754427244},
{2,0,0.17745845870123},
{3,0.5,-0.12361807851599},
{4,0.5,0.45145314010528E-1},
{4,0.75,0.30476479965980E-1}};
coeficientesPolinomio=Aloca(numeroCoeficientes[1],3);
for(int i=0;i<numeroCoeficientes[1];i++){
        for(int j=0;j<3;j++)coeficientesPolinomio[i][j]=d[i][j];
}
double e[16][4]={
{1,1,2,0.75508387706302},
{1,1,2.5,-0.85885381015629},
{1,2,2.5,0.36324009830684E-1},
{1,7,1.5,-0.19548799450550E-1},
{1,8,1,-0.44452392904960E-2},
{1,8,1.5,0.46410763666460E-2},
{2,1,4,-0.71444097992825E-1},
{2,2,7,-0.80765060030713E-1},
{2,3,3,0.15560460945053},
{2,3,7,0.20318752160332E-2},
{2,4,3,-0.10624883571689},
{2,5,1,0.39807690546305E-1},
{2,5,6,0.16371431292386E-1},
{2,10,0,0.53212200682628E-3},
{3,2,6,-0.78681561156387E-2},
{3,6,13,-0.30981191888963E-2}};
coeficientesExponenciais=Aloca(numeroCoeficientes[2],4);
for(int i=0;i<numeroCoeficientes[2];i++){
        for(int j=0;j<4;j++)coeficientesExponenciais[i][j]=e[i][j];
}
double f[2][7]={
        {1,2,-0.42276036810382E-1,10,150,1.16,0.85},
        {2,0,-0.53001044558079E-2,10,200,1.13,1}};
coeficientesGaussianos=Aloca(numeroCoeficientes[3],7);
for(int i=0;i<numeroCoeficientes[3];i++){
        for(int j=0;j<7;j++)coeficientesGaussianos[i][j]=f[i][j];
}
fIdeal=0;
fIdeal_t=0;
fIdeal_tt=0;
fIdeal_ttt=0;
fResidual=0;
fResidual_t=0;
fResidual_d=0;
fResidual_dd=0;
```

```
        fResidual_dt=0;
        fResidual_tt=0;
        fResidual_ddd=0;
        fResidual_ddt=0;
        fResidual_dtt=0;
        fResidual_ttt=0;
}
```

Apresenta-se na listagem 6.12 o arquivo de implementação da classe `CHelmholtzMetano`.

Listing 6.12: Arquivo de implementação da classe CHelmholtzMetano.

```
#include "CHelmholtzAgua.h"
CHelmholtzMetano::CHelmholtzMetano()
{
        tCritica=190.564;
        dCritica=162.66;
        rEspecifico=0.5182705;
        int a[5]={5,13,23,4,0};
        for(int i=0;i<5;i++)numeroCoeficientes[i]=a[i];
        double b[3]={9.91243972,-6.33270087,3.0016};
        coeficientesIdeais1 = new double[3];
        for(int i=0;i<3;i++)coeficientesIdeais1[i]=b[i];
        double c[5][2]={
        {0.008449,3.40043240},
        {4.6942,10.26951575},
        {3.4865,20.43932747},
        {1.6572,29.93744884},
        {1.4115,79.13351945}};
        coeficientesIdeais2=Aloca(numeroCoeficientes[0],2);
        for(int i=0;i<5;i++){
                for(int j=0;j<2;j++)coeficientesIdeais2[i][j]=c[i][j];
        }
        double d[13][3]={
        {1,-0.5,0.4367901028E-1},
        {1,0.5,0.6709236199},
        {1,1,-0.1765577859E1},
        {2,0.5,0.8582330241},
        {2,1,-0.1206513052E1},
        {2,1.5,0.5120467220},
        {2,4.5,-0.4000010791E-3},
        {3,0,-0.1247842423E-1},
        {4,1,0.3100269701E-1},
        {4,3,0.1754748522E-2},
        {8,1,-0.3171921605E-5},
        {9,3,-0.2240346840E-5},
        {10,3,0.2947056156E-6}};
        coeficientesPolinomio=Aloca(numeroCoeficientes[1],3);
        for(int i=0;i<13;i++){
                for(int j=0;j<3;j++)coeficientesPolinomio[i][j]=d[i][j];
        }
        double e[23][4]={
        {1,1,0,0.1830487909},
        {1,1,1,0.1511883679},
        {1,1,2,-0.4289363877},
        {1,2,0,0.6894002446E-1},
        {1,4,0,-0.1408313996E-1},
        {1,5,2,-0.3063054830E-1},
        {1,6,2,-0.2969906708E-1},
        {2,1,5,-0.1932040831E-1},
        {2,2,5,-0.1105739959},
```

```
        {2,3,5,0.9952548995E-1},
        {2,4,2,0.8548437825E-2},
        {2,4,4,-0.6150555662E-1},
        {3,3,12,-0.4291792423E-1},
        {3,5,8,-0.1813207290E-1},
        {3,5,10,0.3445904760E-1},
        {3,8,10,-0.2385919450E-2},
        {4,2,10,-0.1159094939E-1},
        {4,3,14,0.6641693602E-1},
        {4,4,12,-0.2371549590E-1},
        {4,4,18,-0.3961624905E-1},
        {4,4,22,-0.1387292044E-1},
        {4,5,18,0.3389489599E-1},
        {4,6,14,-0.2927378753E-2}};
        coeficientesExponenciais=Aloca(numeroCoeficientes[2],4);
        for(int i=0;i<23;i++){
                for(int j=0;j<4;j++)coeficientesExponenciais[i][j]=e[i][j];
        }
        double f[4][7]={
                {2,2,0.9324799946E-4,20,200,1.07,1},
                {0,0,-0.6287171518E1,40,250,1.11,1},
                {0,1,0.1271069467E2,40,250,1.11,1},
                {0,2,-0.6423953466E1,40,250,1.11,1}};
        coeficientesGaussianos=Aloca(numeroCoeficientes[3],7);
        for(int i=0;i<4;i++){
                for(int j=0;j<7;j++)coeficientesGaussianos[i][j]=f[i][j];
        }
        fIdeal=0;
        fIdeal_t=0;
        fIdeal_tt=0;
        fIdeal_ttt=0;
        fResidual=0;
        fResidual_t=0;
        fResidual_d=0;
        fResidual_dd=0;
        fResidual_dt=0;
        fResidual_tt=0;
        fResidual_ddd=0;
        fResidual_ddt=0;
        fResidual_dtt=0;
        fResidual_ttt=0;
}
```

Apresenta-se na listagem 6.13 o arquivo de implementação da classe `CHelmholtzNbutano`.

Listing 6.13: Arquivo de implementação da classe CHelmholtzNbutano.

```
#include "CHelmholtzAgua.h"
CHelmholtzNbutano::CHelmholtzNbutano()
{
        tCritica=425.125;
        dCritica=228;
        rEspecifico=0.14305157;
        int a[5]={4,7,16,2,0};
        for(int i=0;i<5;i++)numeroCoeficientes[i]=a[i];
        double b[3]={12.54882924,-5.46976878,3.24680487};
        coeficientesIdeais1 = new double[3];
        for(int i=0;i<3;i++)coeficientesIdeais1[i]=b[i];
        double c[4][2]={
        {5.54913289,0.7748404445},
        {11.4648996,3.3406025522},
```

```
{7.59987584,4.9705130961},
{9.66033239,9.9755537783}};
coeficientesIdeais2=Aloca(numeroCoeficientes[0],2);
for(int i=0;i<numeroCoeficientes[0];i++){
        for(int j=0;j<2;j++)coeficientesIdeais2[i][j]=c[i][j];
}
double d[7][3]={
{1,0.5,0.25536998241635E1},
{1,1,-0.44585951806696E1},
{1,1.5,0.82425886369063},
{2,0,0.11215007011442},
{3,0.5,-0.35910933680333E-1},
{4,0.5,0.16790508518103E-1},
{4,0.75,0.32734072508724E-1}};
coeficientesPolinomio=Aloca(numeroCoeficientes[1],3);
for(int i=0;i<numeroCoeficientes[1];i++){
        for(int j=0;j<3;j++)coeficientesPolinomio[i][j]=d[i][j];
}
double e[16][4]={
{1,1,2,0.95571232982005},
{1,1,2.5,-0.10003385753419E1},
{1,2,2.5,0.85581548803855E-1},
{1,7,1.5,-0.25147918369616E-1},
{1,8,1,-0.15202958578918E-2},
{1,8,1.5,0.47060682326420E-2},
{2,1,4,-0.97845414174006E-1},
{2,2,7,-0.48317904158760E-1},
{2,3,3,0.17841271865468},
{2,3,7,0.18173836739334E-1},
{2,4,3,-0.11399068074953},
{2,5,1,0.19329896666669E-1},
{2,5,6,0.11575877401010E-2},
{2,10,0,0.15253808698116E-3},
{3,2,6,-0.43688558458471E-1},
{3,6,13,-0.82403190629989E-2}};
coeficientesExponenciais=Aloca(numeroCoeficientes[2],4);
for(int i=0;i<numeroCoeficientes[2];i++){
        for(int j=0;j<4;j++)coeficientesExponenciais[i][j]=e[i][j];
}
double f[2][7]={
        {1,2,-0.28390056949441E-1,10,150,1.16,0.85},
        {2,0,0.14904666224681E-2,10,200,1.13,1}};
coeficientesGaussianos=Aloca(numeroCoeficientes[3],7);
for(int i=0;i<numeroCoeficientes[3];i++){
        for(int j=0;j<7;j++)coeficientesGaussianos[i][j]=f[i][j];
}
fIdeal=0;
fIdeal_t=0;
fIdeal_tt=0;
fIdeal_ttt=0;
fResidual=0;
fResidual_t=0;
fResidual_d=0;
fResidual_dd=0;
fResidual_dt=0;
fResidual_tt=0;
fResidual_ddd=0;
fResidual_ddt=0;
fResidual_dtt=0;
fResidual_ttt=0;
```

```
}
```

Apresenta-se na listagem 6.14 o arquivo de implementação da classe `CHelmholtzNitrogenio`.

Listing 6.14: Arquivo de implementação da classe CHelmholtzNitrogenio.

```
#include "CHelmholtzAgua.h"
CHelmholtzNitrogenio::CHelmholtzNitrogenio()
{
        tCritica=126.192;
        dCritica=313.3;
        rEspecifico=0.2968038958;
        int a[5]={1,6,26,4,0};
        for(int i=0;i<5;i++)numeroCoeficientes[i]=a[i];
        double b[6]={-12.76952708,-0.00784163,2.5,-1.934819E-4,-1.247742E-5,6.678326E-8};
        coeficientesIdeais1 = new double[6];
        for(int i=0;i<6;i++)coeficientesIdeais1[i]=b[i];
        double c[1][2]={
        {1.012941,26.65788}};
        coeficientesIdeais2=Aloca(numeroCoeficientes[0],2);
        for(int i=0;i<numeroCoeficientes[0];i++){
                for(int j=0;j<2;j++)coeficientesIdeais2[i][j]=c[i][j];
        }
        double d[6][3]={
        {1,0.25,0.924803575275},
        {1,0.875,-0.492448489428},
        {2,0.5,0.661883336938},
        {2,0.875,-0.192902649201E1},
        {3,0.375,-0.622469309629E-1},
        {3,0.75,0.349943957581}};
        coeficientesPolinomio=Aloca(numeroCoeficientes[1],3);
        for(int i=0;i<numeroCoeficientes[1];i++){
                for(int j=0;j<3;j++)coeficientesPolinomio[i][j]=d[i][j];
        }
        double e[26][4]={
        {1,1,0.5,0.564857472498},
        {1,1,0.75,-0.161720005987E1},
        {1,1,2,-0.481395031883},
        {1,3,1.25,0.421150636384},
        {1,3,3.5,-0.161962230825E-1},
        {1,4,1,0.172100994165},
        {1,6,0.5,0.735448924933E-2},
        {1,6,3,0.168077305479E-1},
        {1,7,0,-0.107626664179E-2},
        {1,7,2.75,-0.137318088513E-1},
        {1,8,0.75,0.635466899859E-3},
        {1,8,2.5,0.304432279419E-2},
        {2,1,4,-0.435762336045E-1},
        {2,2,6,-0.723174889316E-1},
        {2,3,6,0.389644315272E-1},
        {2,4,3,-0.212201363910E-1},
        {2,5,3,0.408822981509E-2},
        {2,8,6,-0.551990017984E-4},
        {3,4,16,-0.462016716479E-1},
        {3,5,11,-0.300311716011E-2},
        {3,5,15,0.368825891208E-1},
        {3,8,12,-0.255856846220E-2},
        {4,3,12,0.896915264558E-2},
        {4,5,7,-0.441513370350E-2},
        {4,6,4,0.133722924858E-2},
        {4,9,16,0.264832491957E-3}};
```

```
        coeficientesExponenciais=Aloca(numeroCoeficientes[2],4);
        for(int i=0;i<numeroCoeficientes[2];i++){
                for(int j=0;j<4;j++)coeficientesExponenciais[i][j]=e[i][j];
        }
        double f[4][7]={
                {1,0,0.196688194015E2,20,325,1.16,1},
                {1,1,-0.209115600730E2,20,325,1.16,1},
                {3,2,0.167788306989E-2,15,300,1.13,1},
                {2,3,0.262767566274E4,25,275,1.25,1}};
        coeficientesGaussianos=Aloca(numeroCoeficientes[3],7);
        for(int i=0;i<numeroCoeficientes[3];i++){
                for(int j=0;j<7;j++)coeficientesGaussianos[i][j]=f[i][j];
        }
        fIdeal=0;
        fIdeal_t=0;
        fIdeal_tt=0;
        fIdeal_ttt=0;
        fResidual=0;
        fResidual_t=0;
        fResidual_d=0;
        fResidual_dd=0;
        fResidual_dt=0;
        fResidual_tt=0;
        fResidual_ddd=0;
        fResidual_ddt=0;
        fResidual_dtt=0;
        fResidual_ttt=0;
}
```

Apresenta-se na listagem 6.15 o arquivo com código da classe `Botao`.

Listing 6.15: Arquivo de cabeçalho da classe Botao.

```
#ifndef BOTAO_H
#define BOTAO_H
#include <QToolButton>
////////////////////////////////////////////////////////////////////////
/// @author Thomas Augusto Menegazzo
/// @class Botao
/// @file Botao.h
/// @brief Classe para criação do botão com as propriedades corretas.
////////////////////////////////////////////////////////////////////////
class Botao : public QToolButton
{
    Q_OBJECT

public:
    ///Construtor.
    ///@param texto Texto no botão.
    explicit Botao(const QString &texto, QWidget *pai = 0);

    ///Tamanho do botão.
    QSize sizeHint() const;
};
#endif
```

Apresenta-se na listagem 6.16 o arquivo de implementação da classe `Botao`.

Listing 6.16: Arquivo de implementação da classe Botao.

```
/*************************************************************************
**
```

```
** Copyright (C) 2013 Digia Plc and/or its subsidiary(-ies).
** Contact: http://www.qt-project.org/legal
**
** This file is part of the examples of the Qt Toolkit.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**     notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**     notice, this list of conditions and the following disclaimer in
**     the documentation and/or other materials provided with the
**     distribution.
**   * Neither the name of Digia Plc and its Subsidiary(-ies) nor the names
**     of its contributors may be used to endorse or promote products derived
**     from this software without specific prior written permission.
**
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
**
** $QT_END_LICENSE$
**
****************************************************************************/

#include <QtWidgets>
#include "botao.h"


Botao::Botao(const QString &texto, QWidget *pai)
    : QToolButton(pai)
{
    setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Preferred);
    setText(texto);
}


QSize Botao::sizeHint() const
{
    QSize size = QToolButton::sizeHint();
    size.rheight() += 20;
    size.rwidth() = qMax(size.width(), size.height());
    return size;
}
```

Apresenta-se na listagem 6.17 o arquivo com código da classe SPTSPGUI.

Listing 6.17: Arquivo de cabeçalho da classe SPTSPGUI.

```
/****************************************************************************
**
** Copyright (C) 2013 Digia Plc and/or its subsidiary(-ies).
** Contact: http://www.qt-project.org/legal
**
** This file is part of the examples of the Qt Toolkit.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**     notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**     notice, this list of conditions and the following disclaimer in
**     the documentation and/or other materials provided with the
**     distribution.
**   * Neither the name of Digia Plc and its Subsidiary(-ies) nor the names
**     of its contributors may be used to endorse or promote products derived
**     from this software without specific prior written permission.
**
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
**
** $QT_END_LICENSE$
**
****************************************************************************/


#ifndef SPTSPGUI_H
#define SPTSPGUI_H
#include <QCheckBox>
#include <QWidget>
#include <QLabel>
#include <QComboBox>
#include <QDialog>
#include <QTableView>
#include <QFile>
#include <QTextStream>
#include <QAbstractItemModel>
#include "CSimuladorPropriedades.h"
#include <QDoubleValidator>
QT_BEGIN_NAMESPACE
class QLineEdit;
QT_END_NAMESPACE
class Botao;
```

```
////////////////////////////////////////////////////////////////////////
/// @author Thomas Augusto Menegazzo
/// @class SPTSPGUI
/// @file SPTSPGUI.h
/// @brief Classe para a implementação da interface gráfica.
////////////////////////////////////////////////////////////////////////

class SPTSPGUI : public QWidget
{
    Q_OBJECT

public:
    /// Construtor da classe.
    SPTSPGUI(QWidget *pai = 0);

private slots:
    /// Função chamada quando o botão calcular é clicado.
    void calcularClicado();

    /// Função chamada quando checkTodos é clicado.
    void checkTodosClicado();

    /// Função chamada quando checkPropriedades é clicado.
    void checkPropriedadesClicado();

    /// Função chamada quando checkDerTemp é clicado.
    void checkDerTempClicado();

    /// Função chamada quando checkDerDens é clicado.
    void checkDerDensClicado();

    /// Função chamada quando checkDerPressao é clicado.
    void checkDerPressaoClicado();

    ///Função chamada quando o botão de salvar em disco é clicado.
    void SalvarEmDisco();

private:
    ///Função para criar um botão conectado a um slot.
    /// @param texto Texto do botão.
    /// @param membro Slot a ser conectado.
    /// @return Botão que realiza o slot escolhido quando clicado.
    Botao *criarBotao(const QString &texto, const char *membro);

    ///Função para criar um CheckBox conectado a um slot.
    /// @param texto Texto do CheckBox.
    /// @param membro Slot a ser conectado.
    /// @return CheckBox que realiza o slot escolhido quando clicado.
    QCheckBox *criarCheck(const QString &texto, const char *membro);

    ///Checkbox para seleção das propriedades a serem calculadas.
    QCheckBox *checkTodos, *checkPropriedades, *checkDerTemp, *checkDerDens, *
        checkDerPressao;
    QCheckBox *checkPressao,*checkEntalpia, *checkEntropia, *checkEnergiaInterna, *
        checkEnergiaGibbs, *checkCapCalorificaIsobarica;
    QCheckBox *checkPressao_t,*checkEntalpia_t, *checkEntropia_t, *checkEnergiaInterna_t,
         *checkEnergiaGibbs_t, *checkCapCalorificaIsobarica_t;
    QCheckBox *checkPressao_d, *checkEntalpia_d, *checkEntropia_d, *checkEnergiaInterna_d
        , *checkEnergiaGibbs_d, *checkCapCalorificaIsobarica_d;
    QCheckBox *checkEntalpia_p, *checkEntropia_p, *checkEnergiaInterna_p, *
```

```
        checkEnergiaGibbs_p, *checkCapCalorificaIsobarica_p;
    QCheckBox *checkCapCalorificaIsovolumetrica, *checkCoefJouleThomson, *
        checkCoefEnforcamento, *checkVelocidadeSom, *checkCoefIsentropico;
    QCheckBox *checkCapCalorificaIsovolumetrica_t, *checkCoefJouleThomson_t, *
        checkCoefEnforcamento_t, *checkVelocidadeSom_t, *checkCoefIsentropico_t;
    QCheckBox *checkCapCalorificaIsovolumetrica_d, *checkCoefJouleThomson_d, *
        checkCoefEnforcamento_d, *checkVelocidadeSom_d, *checkCoefIsentropico_d;
    QCheckBox *checkCapCalorificaIsovolumetrica_p, *checkCoefJouleThomson_p, *
        checkCoefEnforcamento_p, *checkVelocidadeSom_p, *checkCoefIsentropico_p;

    ///Função para passar a substância escolhida.
    /// @return String usado no construtor do simulador.
    std::string Substancia();/*Funcao para passar a substancia correta para calculo*/

    ///Função para calcular apenas as propriedades selecionadas.
    void EscolherCalculos();

    ///Popup com os resultados calculados.
    QDialog *popup;

    ///Função para criação do popup.
    void CriadorPopup();

    ///Simulador que realizará os calculos.
    CSimuladorPropriedades *sim;

    ///Caixa de entrada da densidade.
    QLineEdit *displayDensidade;

    ///Caixa de entrada da temperatura.
    QLineEdit *displayTemperatura;

    ///Caixa de seleção da substância
    QComboBox *substanciaBox;
};

#endif
```

Apresenta-se na listagem 6.18 o programa que usa a classe `SPTSPGUI`.

Listing 6.18: Arquivo de implementação da função `main()`.

```
/****************************************************************************
**
** Copyright (C) 2013 Digia Plc and/or its subsidiary(-ies).
** Contact: http://www.qt-project.org/legal
**
** This file is part of the examples of the Qt Toolkit.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**     notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**     notice, this list of conditions and the following disclaimer in
**     the documentation and/or other materials provided with the
**     distribution.
```

```
**   * Neither the name of Digia Plc and its Subsidiary(-ies) nor the names
**     of its contributors may be used to endorse or promote products derived
**     from this software without specific prior written permission.
**
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
**
** $QT_END_LICENSE$
**
****************************************************************************/

#include <QApplication>
#include "SPTSPGUI.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    SPTSPGUI gui;
    gui.show();
    return app.exec();
}
```

Capítulo 7

Teste

Todo projeto de engenharia passa por uma etapa de testes. Neste capítulo apresentamos alguns testes do software desenvolvido. Estes testes devem dar resposta aos diagramas de caso de uso inicialmente apresentados (diagramas de caso de uso geral e específicos).

7.1 Teste 1: Teste dos cálculos

Neste teste são testados os valores das diferentes propriedades, suas derivadas e as derivadas da energia de Helmholtz. Estes valores são comparados com os encontrados na literatura ou, no caso das derivadas, com valores das derivadas calculadas numéricamente.

- Sequência do teste:
  - Rodar programa
  - Selecionar Água
  - Entrar com temperatura=500 [K] e densidade=838.025 [kg/m^3]
  - Veja resultados na Figura 7.1.

Figura 7.1: Tela do programa mostrando os valores calculados no Teste 1

7.2 Teste 2: Teste de funcionamento da interface gráfica

Neste teste verifica-se o funcionamento da interface gráfica. Para sua realização, todas as propriedades serão calculadas para diversas substâncias, temperaturas e densidades. Os valores mostrados na tela serão comparados com os valores adquiridos no programa em interface de texto.

- Sequência do teste:
  - Rodar programa
  - Selecionar Água
  - Entrar com temperatura=500 [K] e densidade=838.025 [kg/m^3]
  - Veja resultados na Figura 7.2.

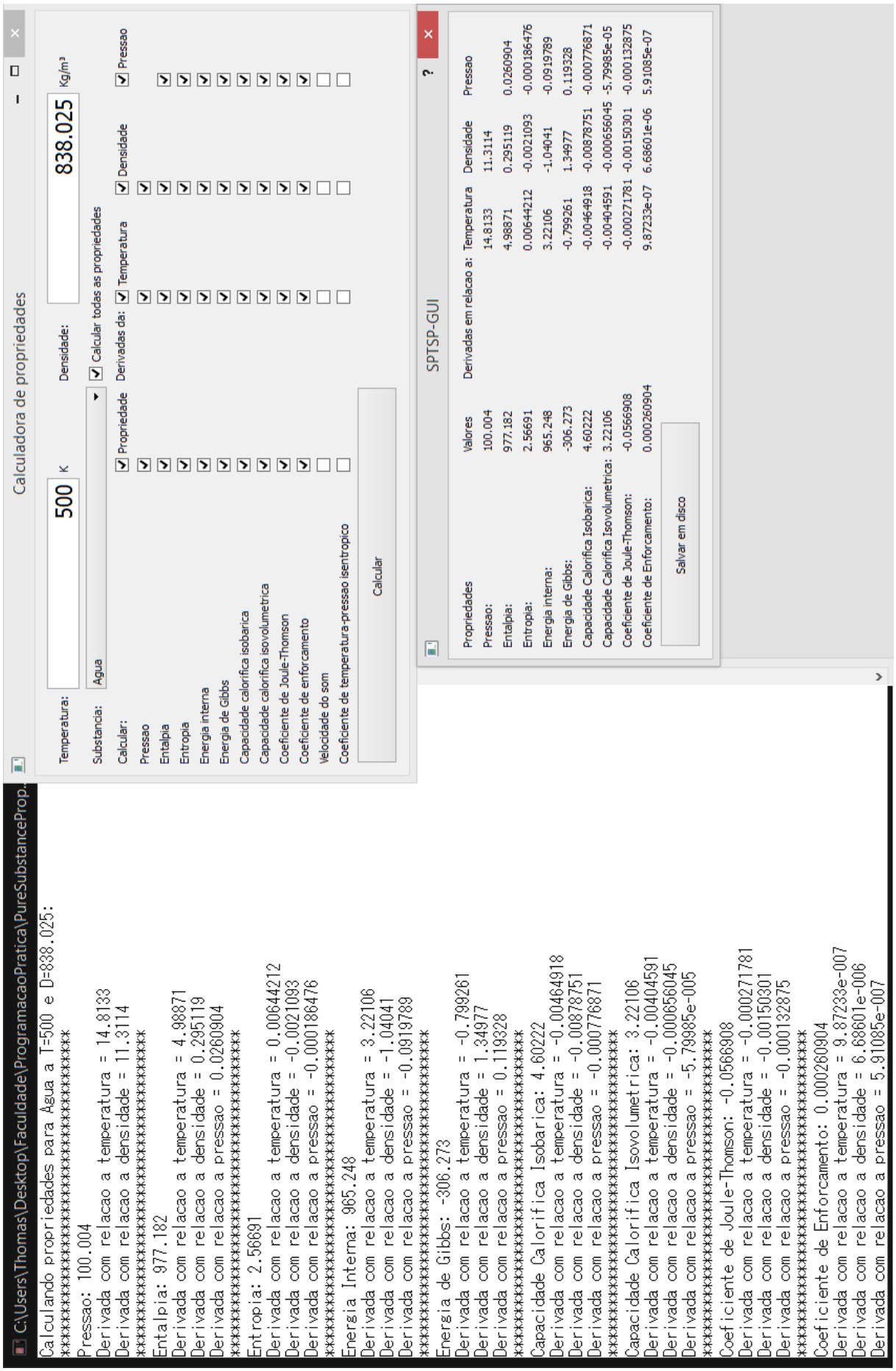

Figura 7.2: Tela do programa mostrando o teste 2 para a água a T=500 e d=838.025

Capítulo 8

Documentação

Todo projeto de engenharia precisa ser bem documentado. Neste sentido, apresenta-se neste capítulo a documentação de uso do software SPTSP. Esta documentação tem o formato de uma apostila que explica passo a passo como usar o software.

8.1 Documentação do usuário

Descreve-se aqui o manual do usuário, um guia que explica, passo a passo, a forma de instalação e uso do software desenvolvido.

8.1.1 Como instalar o software

Para instalar o software execute o seguinte passo a passo:

- Rode o programa pelo executável SPTSP-GUI.exe disponivel.

Alternativamente:

- Instale a biblioteca Qt 5.3, disponivel em http://qt-project.org/.
- Abra o projeto pelo software QtCreator instalado.
- Compile e rode o programa apertando no botão *Run*(ctrl+R).

8.1.2 Como rodar o software

Para rodar o software por meio da interface gráfica apenas escolha a substância, sua temperatura e densidade, e o que deseja calcular. Pressione o botão calcular para receber os resultados.

Para usar o software pela interface texto modifique o arquivo main.cpp de acordo com o que deseja e rode o programa.

8.2 Documentação para desenvolvedor

Apresenta-se nesta seção a documentação para o desenvolvedor, isto é, informações para usuários que queiram modificar, aperfeiçoar ou ampliar este software.

8.2.1 Dependências

Para compilar o software é necessário atender as seguintes dependências:

- Instalar o compilador g++ da GNU disponível em http://gcc.gnu.org.
- O software `Qt 5.3`, disponível no endereço http://qt-project.org/, deve estar instalado.

Referências Bibliográficas

[Bucker and Wagner, 2006a] Bucker, D. and Wagner, W. (2006a). A reference equation of state for the thermodynamic properties of ethane for temperatures from the melting line to 675 k and pressures up to 900 mpa. *J. Phys. Chem. Ref. Data*, 35:205–266. 7

[Bucker and Wagner, 2006b] Bucker, D. and Wagner, W. (2006b). Reference equations of state for the thermodynamic properties of fluid phase n butane and isobutane. *J. Phys. Chem. Ref. Data*, 35:929–1019. 7

[Bueno, 2003] Bueno, A. D. (2003). *ProgramaÃ§Ã£o Orientada a Objeto com C++ - Aprenda a Programar em Ambiente Multiplataforma com Software Livre.* Novatec, SÃ£o Paulo. 17

[Setzmann and Wagner, 1991] Setzmann, U. and Wagner, W. (1991). A new equation of state and tables of thermodynamic properties for methane covering the range from the melting line to 625 k at pressures up to 100 mpa. *J. Phys. Chem. Ref. Data*, 20:1061–1155. 7

[Span and Wagner, 1996] Span, R. and Wagner, W. (1996). A new equation of state for carbon dioxide covering the fluid region from the triple point temperature to 1100 k at pressures up to 800 mpa. *J. Phys. Chem. Ref. Data*, 25:1509–1596. 7

[Span et al., 2000] Span, R., Wagner, W., Lemmon, E. W., Jacobsen, R. T., and Yokozeki, A. (2000). A reference equation of state for the thermodynamic properties of nitrogen for temperatures from 63.151 to 1000 k and pressures to 2200 mpa. *J. Phys. Chem. Ref. Data*, 29:1361–1433. 7

[Wagner and Prub, 2002] Wagner, W. and Prub, A. (2002). The iapws formulation 1995 for the thermodynamic properties of ordinary water substance for general and scientific use. *J. Phys. Chem. Ref. Data*, 31:387–535. 7

Índice Remissivo