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

# PROJETO ENGENHARIA
# DESENVOLVIMENTO DO SOFTWARE
# SIMULAÇÃO DE PROPRIEDADES TERMODINÂMICAS DE SUBSTÂNCIAS SIMPLES E COMPOSTAS
# TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA

Versão 1:

Daniel Guimarães Alvarenga
Prof. André Duarte Bueno

MACAÉ - RJ
Março - 2017

# Sumário

# Capítulo 1

# Introdução

No presente projeto de engenharia desenvolve-se o *software de simulação de propriedades termodinâmicas de substâncias simples e compostas*, um software aplicado a engenharia de petróleo e que utiliza o paradigma da orientação a objetos.

Este software tem como finalidade obter propriedades termodinâmicas de substâncias simples ou misturas de duas substâncias. Para isso, será necessário um banco de dados para armazenar propriedades das principais substâncias relacionadas à produção de petróleo, um sistema que permita ao usuário escolher entre usar a equação de estado de Peng-Robinson ou de Soave-Redlich-Kwong, um sistema de resolução de equação de terceiro grau de modo exato (para que não haja erros em métodos numéricos que não convergem em determinada situação), e, enfim, o cálculo das propriedades termodinâmicas para cada fase e cada substância. Tais propriedades como fugacidade, fator de compressibilidade, densidade, volume específico e volume molar, serão mostrados ao usuário juntamente com um gráfico do fator de compressibilidade e podem ser salvos em disco.

## 1.1 Escopo do problema

Na composição do petróleo, além de hidrocarbonetos, são encontradas substâncias como nitrogênio, oxigênio, enxofre, e até metais. Estas substâncias podem estar presentes tanto no estado líquido, quanto no estado gasoso.

Durante a etapa de produção do petróleo, é importante monitorar o comportamento das substâncias que estão sendo produzidas. Isto porque um aumento muito grande de pressão pode fazer com que os hidrocarbonetos fluam de maneira descontrolada e haja derramamento na superfície, o que é indesejável por todos os danos pessoais, ambientais e materiais que isso causa. Além disso, se a produção for parada por um certo tempo, sob certas condições de temperatura e pressão, há o risco de formação de hidratos nos tubos, que os entopem e comprometem a produção.

A necessidade de monitoramento da produção é evidente. Prever como será o comportamento dessas substâncias antes que elas causem algum tipo de problema, é uma

medida que traz certo controle aos engenheiros que gerenciam a produção. Este projeto de engenharia busca indícios que ajudem os engenheiros a prever eventuais problemas que possam ocorrer durante a etapa de produção do petróleo, levando em conta as propriedades termodinâmicas obtidas por meio de simulações das condições de temperatura e pressão.

## 1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:
  - Obter propriedades termodinâmicas de substâncias simples e misturas de substâncias. Estas propriedades incluem fugacidade, densidade, volume específico, volume molar, e o fator de compressibilidade, tanto da fase líquida, quanto da fase vapor (se as duas coexistirem).
- Objetivos específicos:
  - Permitir que o usuário selecione qual equação de estado deseja utilizar para a base de cálculos: Peng-Robinson ou Soave-Redlich-Kwong.
  - Permitir que o usuário selecione as condições de temperatura e pressão que deseja simular, assim como a(s) substância(s) envolvida(s).
  - Gerar resultados (valor das propriedades e gráfico) na tela e salvá-los em disco.

# Capítulo 2

# Especificação

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

## 2.1 Nome do sistema/produto

| | |
|---|---|
| **Nome** | Simulador de propriedades termodinâmicas de substâncias simples e compostas. |
| **Componentes principais** | 1. Sistema de banco de dados que armazena propriedades físicas das principais substâncias envolvidas na produção de petróleo;<br>2. Sistema que calcula propriedades termodinâmicas de uma substância ou uma mistura de duas substâncias. |
| **Missão** | Simular condições de temperatura e pressão que as substâncias podem se encontrar durante a produção de petróleo para obter as propriedades termodinâmicas nessas condições. |

## 2.2 Especificação

O software a ser desenvolvido deverá realizar cálculos para obter a fugacidade, densidade, volume molar, volume específico, e fator de compressibilidade da substância simples ou da mistura de duas substâncias. Cada uma dessas propriedades deve ser mostrada para a fase líquida e para a fase vapor, caso as duas fases coexistam.

O software será desenvolvido em linguagem C++, com orientação a objeto, e poderá ser utilizado nos sistemas operacionais GNU/Linux, Windows, e OS X, sendo operado em modo texto, e contendo apenas uma janela. Sua licença é GPL (*General Public License*).

O usuário deverá informar se quer utilizar a equação de estado de Peng-Robinson (Peng, D. Y.; Robinson, D. B. (1976). "A New Two-Constant Equation of State". *Industrial and Engineering Chemistry: Fundamentals*) ou Soave-Redlich-Kwong (Soave, G. *Equilibrium Constants from a Modified Redlich-Kwong Equation of State*, Chem. Eng. Sci.) para realização dos cálculos das propriedades termodinâmicas. Além disso, deverá informar a substância ou substâncias que deseja, assim como a temperatura e pressão que serão simuladas.

Após a realização dos cálculos, os resultados serão apresentados na tela em forma de texto e gráfico, e o usuário terá a opção de salvá-los em disco.

Os gráficos serão gerados pelo software externo *Gnuplot* (www.gnuplot.info).

### 2.2.1 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

| | |
|---|---|
| **RF-01** | O usuário deve ser capaz de escolher se deseja simular uma substância simples ou uma mistura de duas substâncias. |

| | |
|---|---|
| **RF-02** | O usuário deverá ter liberdade para escolher qual equação de estado deseja utilizar. |

| | |
|---|---|
| **RF-03** | O usuário deve poder escolher a temperatura e pressão deseja para a simulação. |

| | |
|---|---|
| **RF-04** | O usuário deve ser capaz de salvar as propriedades termodinâmicas calculadas em disco, assim como o gráfico do fator de compressibilidade. |

### 2.2.2 Requisitos não funcionais

| | |
|---|---|
| **RNF-01** | A resolução da equação de estado cúbica não deve utilizar um método numérico. Por isso será utilizado o método analítico exato. |

| | |
|---|---|
| **RNF-02** | O programa deverá ser multi-plataforma, podendo ser executado em *Windows*, *GNU/Linux* ou OS X. |

## 2.3 Casos de uso

| | |
|---|---|
| **Nome do caso de uso:** | Obter propriedades termodinâmicas de uma substância simples |
| **Resumo/descrição:** | Processos que o programa realiza desde o momento em que o usuário insere os dados desejados, até a armazenagem dos resultados em disco. |
| **Etapas:** | 1. Reconhecer Tc, Pc e ω da substância;<br>2. Calcular parâmetros da equação de estado escolhida;<br>3. Resolver equação de terceiro grau analiticamente para encontrar fator de compressibilidade;<br>4. Escolher o(s) resultado(s) que fazem sentido físico;<br>5. Calcular propriedades termodinâmicas;<br>6. Mostrar resultados;<br>7. Salvar resultados e gráfico em disco;<br>8. Plotar o gráfico do fator de compressibilidade. |
| **Cenários alternativos:** | 1. Inserir o nome errado de uma substância, ou uma substância que não esteja no banco de dados;<br>2. Inserir pressão negativa;<br>3. Inserir temperatura negativa;<br>3. Inserir número de equação de estado inexistente;<br>4. Inserir fração molar maior ou igual a 1 (um). |

### 2.3.1 Diagrama de caso de uso geral

O diagrama de caso de uso geral da Figura 2.1 mostra o processo para cálculo das propriedades termodinâmicas e saídas para o usuário.

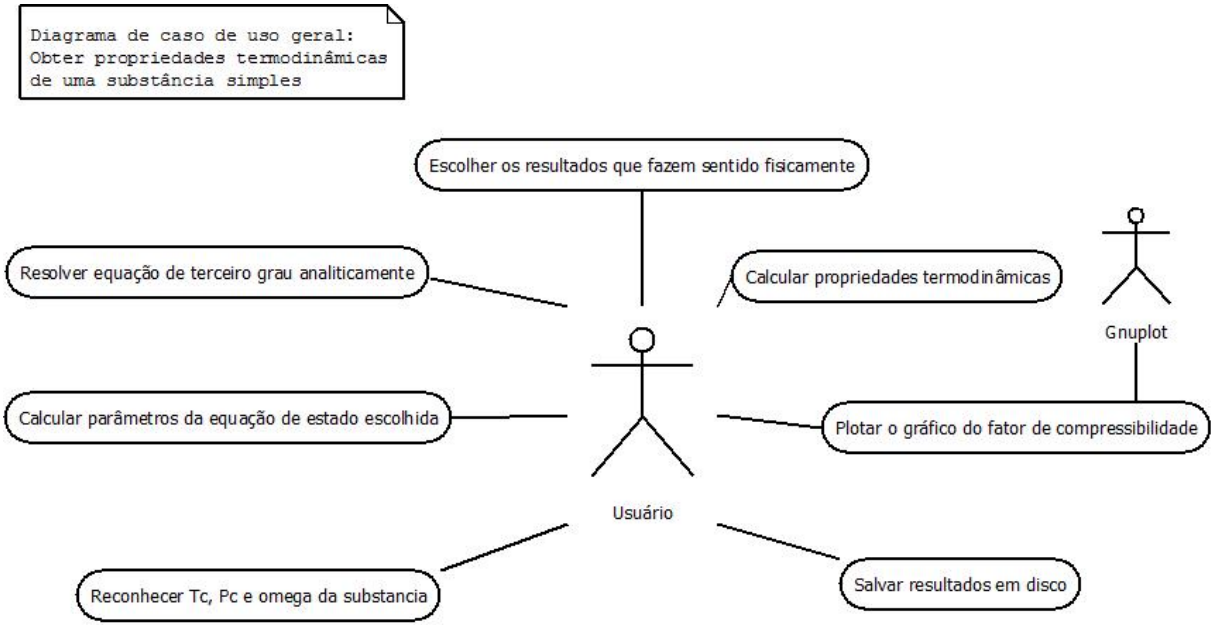



Figura 2.1: Caso de uso geral: Obter propriedades termodinâmicas de uma substância simples

### 2.3.2 Diagrama de caso de uso específico

O caso de uso "Resolver equação de terceiro grau analiticamente com delta menor que zero" descrito na Figura 2.1 é detalhado na Figura 2.2. O software irá calcular os coeficientes $D$ e $E$ de uma equação de terceiro grau, e depois calculará o delta, que neste caso de uso será negativo. Após constatado que o delta é negativo, será calculado o coeficiente theta. Em seguida, será utilizado o método de obtenção das raízes para delta negativo, utilizando os coeficientes calculados.

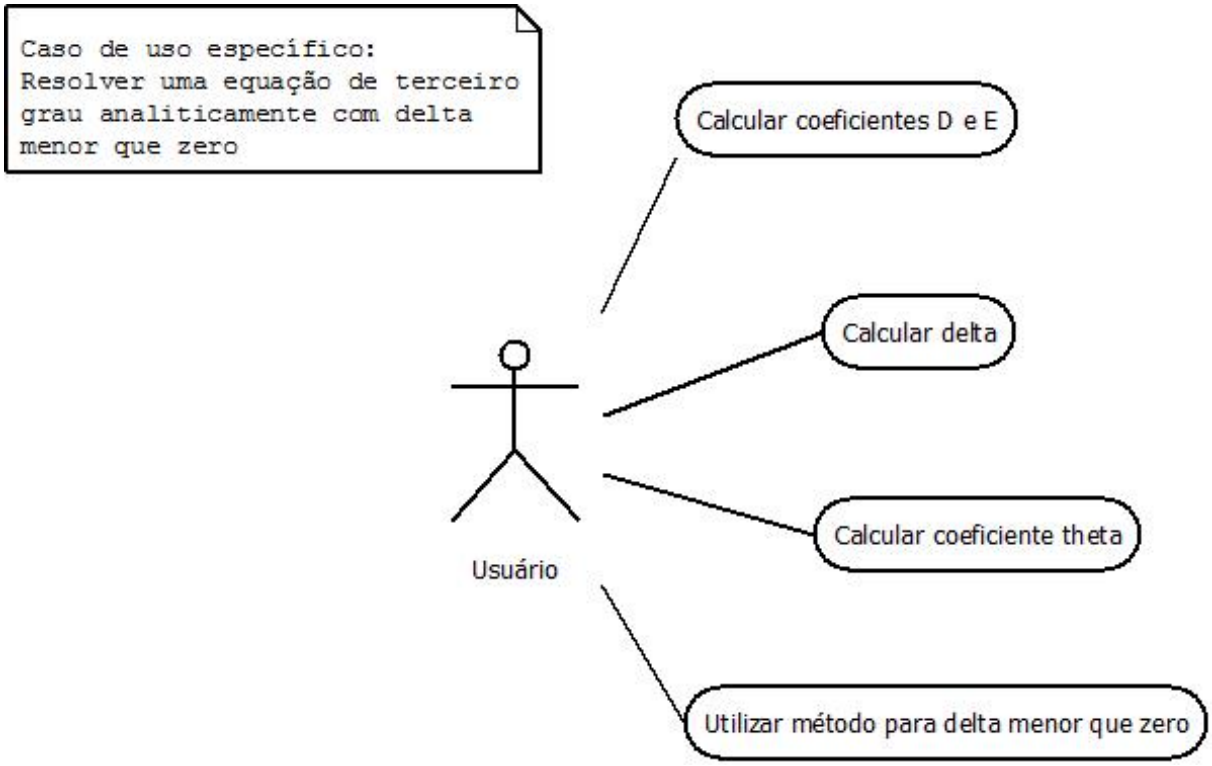



Figura 2.2: Caso de uso específico: Resolver equação de terceiro grau analiticamente com delta menor que zero

# Capítulo 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

Após estudo dos requisitos/especificações do sistema, algumas entrevistas, estudos na biblioteca e disciplinas do curso foi possível identificar nosso domínio de trabalho:

- O software funcionará para uma substância simples ou para uma mistura de duas substâncias;
- O software envolverá o uso de conceitos das disciplinas de Matemática, Física, Termodinâmica e Engenharia de Reservatório, além de Fundamentos da Ciência da Computação;
- Constará no banco de dados as propriedades das principais substâncias encontradas durante a produção de petróleo.

## 3.2 Formulação teórica

Nesta seção consta a formulação física-matemática, as equações envolvidas, as constantes e parâmetros calculados, e as propriedades desejadas pelo usuário.

### 3.2.1 Equações cúbicas

As equações de estado cúbicas, por definição, possuem três soluções. Considerando a equação genérica:

$$x^3 + Ax^2 + Bx + C = 0, \tag{3.1}$$

Podemos definir, para simplificar as contas:

$$D = (A/3.0)^3 - (AB/6.0) + (C/2.0)$$

$$E = (B/3.0) - (A/3.0)^2$$

$$\Delta = D^2 + E^3$$

Se $\Delta = 0$, existem três raízes reais, sendo pelo menos duas iguais:

$$x_1 = 2.0 * \sqrt[3]{(-D)} - (A/3.0)$$

$$x_2 = x_3 = -\sqrt[3]{(-D)} - (A/3.0)$$

Se $\Delta < 0$ , existem três raízes reais e diferentes. Neste caso, definimos:

$$\theta = arccos(-D/\sqrt{-E^3})$$

$$x_1 = 2.0 * \sqrt{-E} * cos(\theta/3.0) - A/3.0$$

$$x_2 = 2.0 * \sqrt{-E} * cos[\theta/3.0 + (2.0/3.0)\pi] - A/3.0$$

$$x_3 = 2.0 * \sqrt{-E} * cos[\theta/3.0 + (4.0/3.0)\pi] - A/3.0$$

Com todos os cálculos trigonométricos feitos em radianos.
Se $\Delta > 0$, existe somente uma raiz real, e podemos definir:

$$F = \sqrt[3]{(-D) + \sqrt{\Delta}}$$

$$G = \sqrt[3]{(-D) - \sqrt{\Delta}}$$

$$x_1 = F + G - (A/3.0)$$

### 3.2.2 Equações de estado cúbicas

É preciso interpretar os resultados obtidos. No caso do software, estas equações serão resolvidas para encontrar as raízes de $Z$, o fator de compressibilidade (adimensional). Porém, só é possível que coexistam as fases vapor e líquida, o que significa que no máximo dois valores de $Z$ terão sentido físico. Deve ser descartado todo valor de $Z$ menor que zero, assim como todos os valores imaginários. Quando forem obtidos dois valores de $Z$ reais, positivos e distintos, o menor será da fase líquida e o maior será da fase vapor. Se houver somente uma raiz real e positiva de $Z$, significa que a substância só terá uma fase. Se o valor de $Z$ for muito pequeno, próximo de zero, significa que a fase única é líquida. Já um valor próximo a 1, significa que a fase única é vapor.

**Peng-Robinson (PR)**

A equação de estado cúbica de Peng-Robinson é descrita por:

$$Z^3 - (1.0 - B)Z^2 + (A - 2.0B - 3.0B^2)Z = 0, \tag{3.2}$$

onde:

$$A = a * \alpha * P/(R^2 * T^2)$$

$$B = b * P/(R * T)$$

$$a = 0,457235 * R^2 * Tc^2/Pc$$

$$b = 0,0777961 * R * Tc/Pc$$

$$\alpha = (1 + \kappa(1 - \sqrt{Tr})^2$$

$$\kappa = 0,37464 + 1,54226\omega - 0,26992\omega^2$$

$$R = 8,314e-5$$

$$Tr = T/Tc$$

$P$ [bar] e $T$ [Kelvin] são a pressão e temperatura que o usuário deseja simular, e insere no início do software;

$Tc$ [Kelvin] é a temperatura crítica, $Pc$ [bar] é a pressão crítica e ω é o fator acêntrico. Estas são propriedades tabeladas de cada substância:

| **Substâncias** | **Temperatura Crítica [Kelvin]** | **Pressão Crítica [bar]** | **Fator acêntrico** |
|---|---|---|---|
| metano | 190,6 | 46 | 0,008 |
| etano | 305,4 | 48,84 | 0,098 |
| propano | 369,8 | 42,46 | 0,152 |
| isobutano | 408,1 | 36,48 | 0,176 |
| n-butano | 425,2 | 38 | 0,193 |
| isopentano | 460,4 | 33,84 | 0,227 |
| n-pentano | 469,6 | 33,74 | 0,51 |
| n-hexano | 507,4 | 29,69 | 0,296 |
| n-heptano | 540,2 | 27,36 | 0,351 |
| n-octano | 568,8 | 24,82 | 0,394 |
| n-decano | 617,6 | 21,08 | 0,49 |
| n-dodecano | 658,3 | 18,24 | 0,562 |
| $CO_2$ | 304,2 | 73,76 | 0,225 |
| nitrogênio | 126,2 | 33,94 | 0,04 |
| $H_2S$ | 373,2 | 89,42 | 0,1 |
| $H_2O$ | 647,3 | 220,48 | 0,344 |

Tabela 3.1: Tabela de propriedades das substâncias
Fonte: *Chemical, Biochemical and Engineering Thermodinamics* (Stanley I. Sandler)

### Soave-Redlich-Kwong (SRK)

A equação de estado cúbica de Soave-Redlich-Kwong é descrita por:

$$Z^3 - Z^2 + (A - B - B^2)Z - (AB) = 0, \tag{3.3}$$

onde:

$$A = a * \alpha * P/(R^2 * T^2)$$

$$B = b * P/(R * T)$$

$$a = 0,42748 * R^2 * Tc^2/Pc$$

$$b = 0,08664 * R * Tc/Pc$$

$$\alpha = (1 + \kappa(1 - \sqrt{Tr})^2$$

$$\kappa = 0,48508 + 1,5517\omega - 0,15613\omega^2$$

$$R = 8,314e - 5$$

$$Tr = T/Tc$$

$P$ [bar] e $T$ [Kelvin] são a pressão e temperatura que o usuário deseja simular, e insere no início do software;

$Tc$ [Kelvin] é a temperatura crítica, $Pc$ [bar] é a pressão crítica e $\omega$ é o fator acêntrico. Estas são propriedades contidas na tabela 3.1.

### 3.2.3 Equações de estado cúbicas para mistura de duas substâncias

Para uma mistura de duas substâncias, a equação 3.2 ainda vale para a PR, mas há uma mudança nos parâmetros. O mesmo ocorre com a equação 3.3, ela ainda vale para SRK, mas há mudanças nos parâmetros.

**Peng-Robinson (PR)**

$$A = a_{mix} * P_{total}/(R^2 * T^2)$$

$$B = b_{mix} * P_{total}/(R * T)$$

$$a_{mix} = a_{subs1} * x_{subs1}^2 + a_{subs2} * x_{subs2}^2 + 2 * x_{subs1} * x_{subs2} * a_{ij}$$

$$b_{mix} = x_{subs1} * b_{subs1} + x_{subs2} * b_{subs2}$$

$$a_{ij} = (1 - Interação) * \sqrt{(a_{subs1} * a_{subs2})}$$

$$a_{subs} = 0,457235 * R^2 * Tc_{subs}^2 / Pc_{subs}$$

$$b_{subs} = 0,0777961 * R * Tc_{subs} / Pc_{subs}$$

$$\alpha_{subs} = (1 + \kappa(1 - \sqrt{Tr_{subs}})^2$$

$$\kappa_{subs} = 0,37464 + 1,54226\omega_{subs} - 0,26992\omega_{subs}^2$$

$$R = 8,314e - 5$$

$$Tr_{subs} = T / Tc_{subs}$$

$P_{total}$ [bar] é a pressão e $T$ [Kelvin] é temperatura que o usuário deseja simular, e insere no início do software;

$Tc$ [Kelvin] é a temperatura crítica, $Pc$ [bar] é a pressão crítica e $\omega$ é o fator acêntrico. Estas são propriedades contidas na tabela 3.1.

$X_{subs}$ é a fração molar de cada substância. A fração molar das duas substâncias somadas deve ser igual a 1;

*Interação* é a interação binária entre os dois compostos, que pode ser encontrada na seguinte tabela:

| **Substâncias** | $C_2H_4$ | $C_2H_6$ | $C_3H_6$ | $C_3H_8$ | $i-C_4H_{10}$ | $n-C_4H_{10}$ | $i-C_4H_{12}$ | $n-C_5H_{12}$ | $n-C_6H_{14}$ | $N_2$ | $CO_2$ | $H_2S$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $CH_4$ | 0,022 | - 0,003 | 0,033 | 0,016 | 0,026 | 0,019 | -0,006 | 0,026 | 0,04 | 0,03 | 0,03 | 0,08 |
| $C_2H_6$ | - | 0,01 | - | - | - | 0,092 | - | - | - | 0,086 | - 0,022 | - |
| $C_3H_6$ | - | - | 0,089 | 0,001 | -0,003 | 0,01 | - | 0,008 | -0,04 | 0,044 | 0,026 | 0,086 |
| $C_3H_8$ | - | - | - | 0,003 | -0,014 | - | - | - | - | 0,09 | 0,026 | 0,08 |
| $i-C_4H_{10}$ | - | - | - | - | -0,003 | 0,003 | 0,011 | 0,027 | 0,001 | 0,078 | 0,03 | 0,08 |
| $n-C_4H_{10}$ | - | - | - | - | - | - | - | - | - | 0,1 | 0,04 | 0,047 |
| $i-C_5H_{12}$ | - | - | - | - | - | - | - | 0,013 | -0,016 | - | 0,04 | 0,03 |
| $n-C_5H_{12}$ | - | - | - | - | - | - | - | 0,06 | - | - | 0,04 | 0,06 |
| $n-C_6H_{14}$ | - | - | - | - | - | - | - | - | - | - | 0,04 | 0,063 |
| $N_2$ | - | - | - | - | - | - | - | - | - | - | 0,012 | 0,17 |
| $CO_2$ | - | - | - | - | - | - | - | - | - | - | - | 0,027 |
| $H_2S$ | - | - | - | - | - | - | - | - | - | - | - | - |

Tabela 3.2: Tabela de interação binária entre substâncias
Fonte: *Chemical, Biochemical and Engineering Thermodinamics* (Stanley I. Sandler)

**Soave-Redlich-Kwong (SRK)**

$$A = a_{mix} * P_{total}/(R^2 * T^2)$$

$$B = b_{mix} * P_{total}/(R * T)$$

$$a_{mix} = a_{subs1} * x_{subs1}^2 + a_{subs2} * x_{subs2}^2 + 2 * x_{subs1} * x_{subs2} * a_{ij}$$

$$b_{mix} = x_{subs1} * b_{subs1} + x_{subs2} * b_{subs2}$$

$$a_{ij} = (1 - interacao) * \sqrt{(a_{subs1} * a_{subs2})}$$

$$a_{subs} = 0,42748 * R^2 * Tc_{subs}^2/Pc_{subs}$$

$$b_{subs} = 0,08664 * R * Tc_{subs}/Pc_{subs}$$

$$\alpha_{subs} = (1 + \kappa(1 - \sqrt{Tr_{subs}})^2$$

$$\kappa_{subs} = 0,48508 + 1,5517\omega_{subs} - 0,15613\omega_{subs}^2$$

$$R = 8,314e - 5$$

$$Tr_{subs} = T/Tc_{subs}$$

$P_{total}$ [bar] é a pressão $T$ [Kelvin] e temperatura que o usuário deseja simular, e insere no início do software;

$Tc$ [Kelvin] é a temperatura crítica, $Pc$ [bar] é a pressão crítica e $\omega$ é o fator acêntrico. Estas são propriedades contidas na tabela 3.1.

$X_{subs}$é a fração molar de cada substância. A fração molar das duas substâncias somadas deve ser igual a 1;

*Interacao* é a interação binária entre os dois compostos, que pode ser encontrada na

tabela 3.2.

### 3.2.4 Cálculo de propriedades termodinâmicas para substâncias simples

Assim que a equação de estado é solucionada e os valores sem significados físicos são descartados, o fator de compressibilidade de cada fase é utilizado para calcular sua densidade [mol/$m^3$], volume específico [$m^3$/mol], volume molar [L/mol], coeficiente de fugacidade e fugacidade [bar] para a pressão da simulação:

**Peng-Robinson (PR)**

$$\rho_{fase} = P/(R * T * Z_{fase})$$

$$v_e = 1.0/\rho_{fase}$$

$$v_m = v_e * 1000.0$$

$$f = exp(Z_{fase} - 1.0 - ln(Z_{fase} - B) - A/(\sqrt{8} * B) * ln((Z_{fase} + (1.0 + \sqrt{2}) * B)/(Z_{fase} + (1.0 - \sqrt{2}) * B)))$$

$$f_P = f * P(bar)$$

onde:
$\rho_{fase}$ [mol/$m^3$] é a densidade de cada fase;
$v_e$ [$m^3$/mol] é o volume específico da substância;
$v_m$ [L/mol] é o volume molar;
$f$ é o coeficiente de fugacidade;
$f_p$ [bar] é a fugacidade na pressão escolhida pelo usuário.

**Soave-Redlich-Kwong (SRK)**

$$\rho_{fase} = P/(R * T * Z_{fase})$$

$$v_e = 1.0/\rho_{fase}$$

$$v_m = v_e * 1000.0$$

$$f = exp(Z_{fase} - 1.0 - ln(Z_{fase} - B) - A/B * ln(1.0 + B/Z_{fase}))$$

$$f_P = f * P(bar)$$

onde:
$\rho$ [mol/$m^3$] é a densidade da substância;
$v_e$ [$m^3$/mol] é o volume específico da substância;
$v_m$ [L/mol] é o volume molar;
$f$ é o coeficiente de fugacidade;
$f_p$[bar] é a fugacidade na pressão escolhida pelo usuário.

### 3.2.5 Cálculo de propriedades termodinâmicas para misturas de duas substâncias

Assim que a equação de estado é solucionada e os valores sem significados físicos são descartados, o fator de compressibilidade de cada fase é utilizado para calcular sua densidade e volume específico. Em seguida é calculado o coeficiente de fugacidade para cada uma das substâncias, além da fugacidade para a pressão da simulação. As correlações a seguir são válidas para ambas as equações de estado trabalhadas no software:

**Peng-Robinson**

$$\rho_{fase} = P_{total}/(R * T * Z_{fase})$$

$$v_e = 1.0/\rho_{fase}$$

$$f_{faseisubs1} = exp(-ln(Z_{fasei} - B) + (Z_{fasei} - 1.0) * b_{subs1}/b_{mix} - A/(\sqrt{8} * B)*$$

$$*(2.0/a_{mix}*(x_{subs1}*a_{subs1}+x_{subs2}*a_{ij})-b_{subs1}/b_{mix})*ln((Z_{fasei}+(1.0+\sqrt{2})*B)/(Z_{fasei}+(1.0-\sqrt{2})*B)))$$

$$f_{faseisubs2} = exp(-ln(Z_{fasei} - B) + (Z_{fasei} - 1.0) * b_{subs2}/b_{mix} - A/(\sqrt{8} * B)*$$

$*(2.0/a_{mix}*(x_{subs2}*a_{subs2}+x_{subs1}*a_{ij})-b_{subs2}/b_{mix})*ln((Z_{fasei}+(1.0+\sqrt{2})*B)/(Z_{fasei}+(1.0-\sqrt{2})*B)))$

$$f_{P_{subs}} = f_{subs} * P_{total}$$

onde:
$\rho_{fase}$ [mol/$m^3$] é a densidade de cada fase;
$v_e$ [$m^3$/mol] é o volume específico de cada fase;
$f_{faseisubs1}$ é o coeficiente de fugacidade de cada fase para substância 1;
$f_{faseisubs2}$ é o coeficiente de fugacidade de cada fase para substância 2;
$f_{psubs}$ [bar] é a fugacidade na pressão escolhida pelo usuário de cada substância.

**Soave-Redlich-Kwong**

$$\rho_{fase} = P_{total}/(R * T * Z_{fase})$$

$$v_e = 1.0/\rho_{fase}$$

$$f_{faseisubs1} = exp(-ln(Z_{fasei} - B) + (Z_{fasei} - 1.0) * b_{subs1}/b_{mix} - A/B*$$

$$(2.0/a_{mix} * (x_{subs1} * a_{subs1} + x_{subs2} * a_{ij}) - b_{subs1}/b_{mix}) * ln(1.0 + B/Z_{fasei}))$$

$$f_{faseisubs2} = exp(-ln(Z_{fasei} - B) + (Z_{fasei} - 1.0) * b_{subs2}/b_{mix} - A/B*$$

$$(2.0/a_{mix} * (x_{subs2} * a_{subs2} + x_{subs1} * a_{ij}) - b_{subs2}/b_{mix}) * ln(1.0 + B/Z_{fasei}))$$

$$f_{P_{faseisubsj}} = f_{faseisubsj} * P_{total}$$

onde:
$\rho_{fase}$ [mol/$m^3$] é a densidade de cada fase;
$v_e$ [$m^3$/mol] é o volume específico de cada fase;
$f_{faseisubs1}$ é o coeficiente de fugacidade de cada fase para substância 1;
$f_{faseisubs2}$ é o coeficiente de fugacidade de cada fase para substância 2;

$f_{pfaseisubsi}$ [ bar] é a fugacidade na pressão escolhida pelo usuário de cada fase e de cada substância.

## 3.3 Identificação de pacotes – assuntos

- CSubstância: Representa as substâncias que serão utilizadas na simulação. Estão contidas nesta classe, o nome e as propriedades de cada substância.
- Matemática: Ciência de raciocínio lógico e abstrato que estuda quantidades, medidas, espaços, estruturas, variações e estatísticas.
  - Resolução de equação: A resolução de equação reúne um conjunto de operações matemáticas necessárias para obter um valor exato da incógnita, que neste caso, é o fator de compressibilidade da(s) substância(s) escolhida(s).
  - Gráfico: O gráfico é a parte não-verbal do resultado retornado pela simulação. Por meio do gráfico cartesiano, pode ser analisada a variação do fator de compressibilidade e a zona de formação de hidratos.
- Termodinâmica: A termodinâmica é uma ciência que estuda os efeitos da variação de temperatura, pressão e volume. Ela está relacionada a todos os outros pacotes, pois cada um deles fornece informações termodinâmicas que serão utilizadas pelo software.
  - PR: A equação de estado Peng-Robinson é uma equação de terceiro grau que relaciona volume, pressão e temperatura de um composto. Quando escrita na forma polinomial, é possível obter de uma a três raízes reais, que poderão ser atribuídas ao(s) fator(es) de compressibilidade.
  - SRK:A equação de estado Soave-Redlich-Kwong é uma equação de terceiro grau que relaciona volume, pressão e temperatura de um composto. Quando escrita na forma polinomial, é possível obter de uma a três raízes reais, que poderão ser atribuídas ao(s) fator(es) de compressibilidade.
- Simulação: É um pacote que reúne todos os dados fornecidos pelos demais pacotes, e simula, matematicamente, as condições de temperatura e pressão. Após fazer os cálculos necessários, retorna ao usuário as propriedades termodinâmicas da(s) substância(s) escolhida(s).

## 3.4 Diagrama de pacotes – assuntos

A Figura 3.1 mostra o diagrama de pacotes do software.

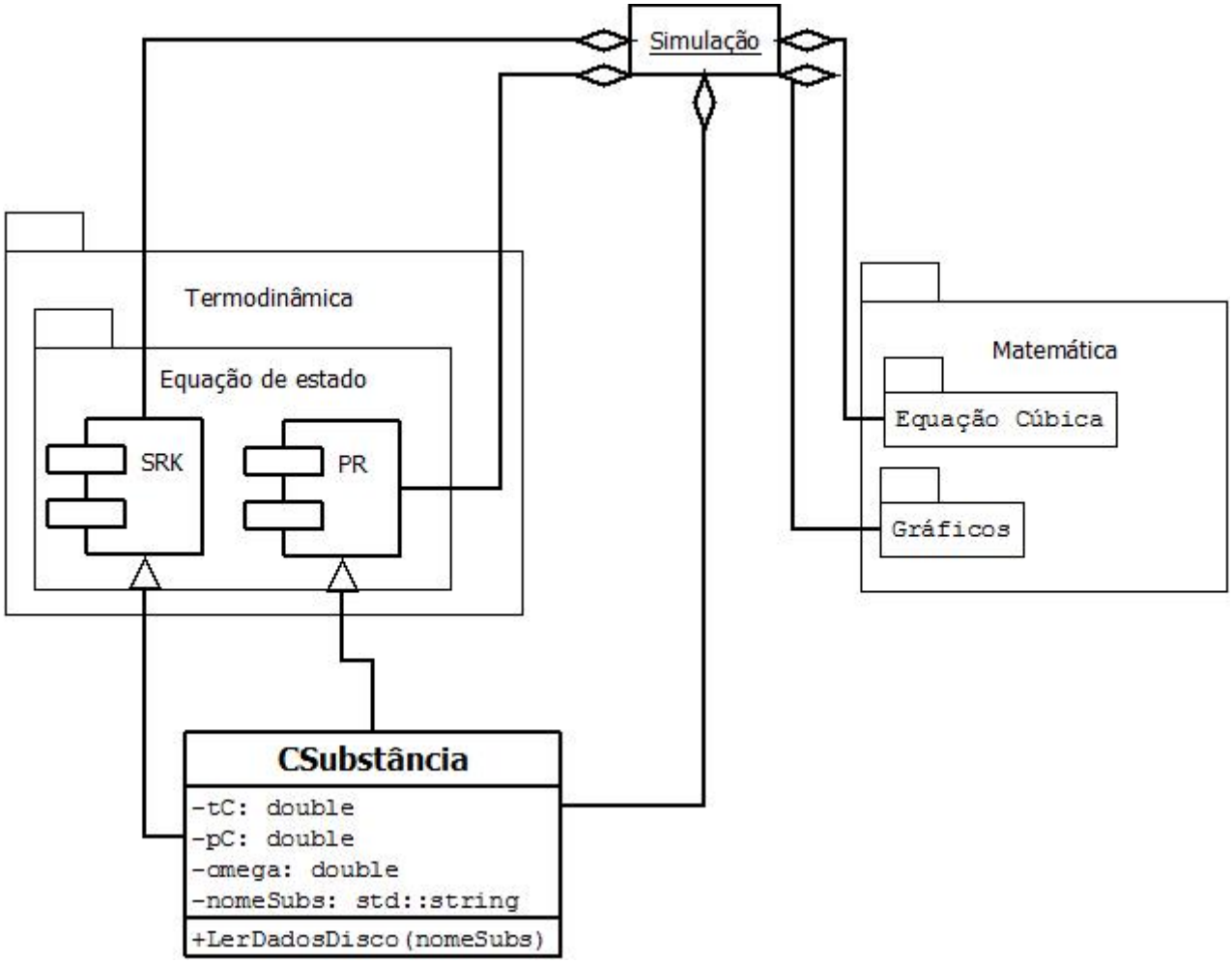



Figura 3.1: Diagrama de Pacotes

# Capítulo 4

# AOO – Análise Orientada a Objeto

## 4.1 Diagramas de classes

O diagrama de classes é apresentado na Figura 4.1.

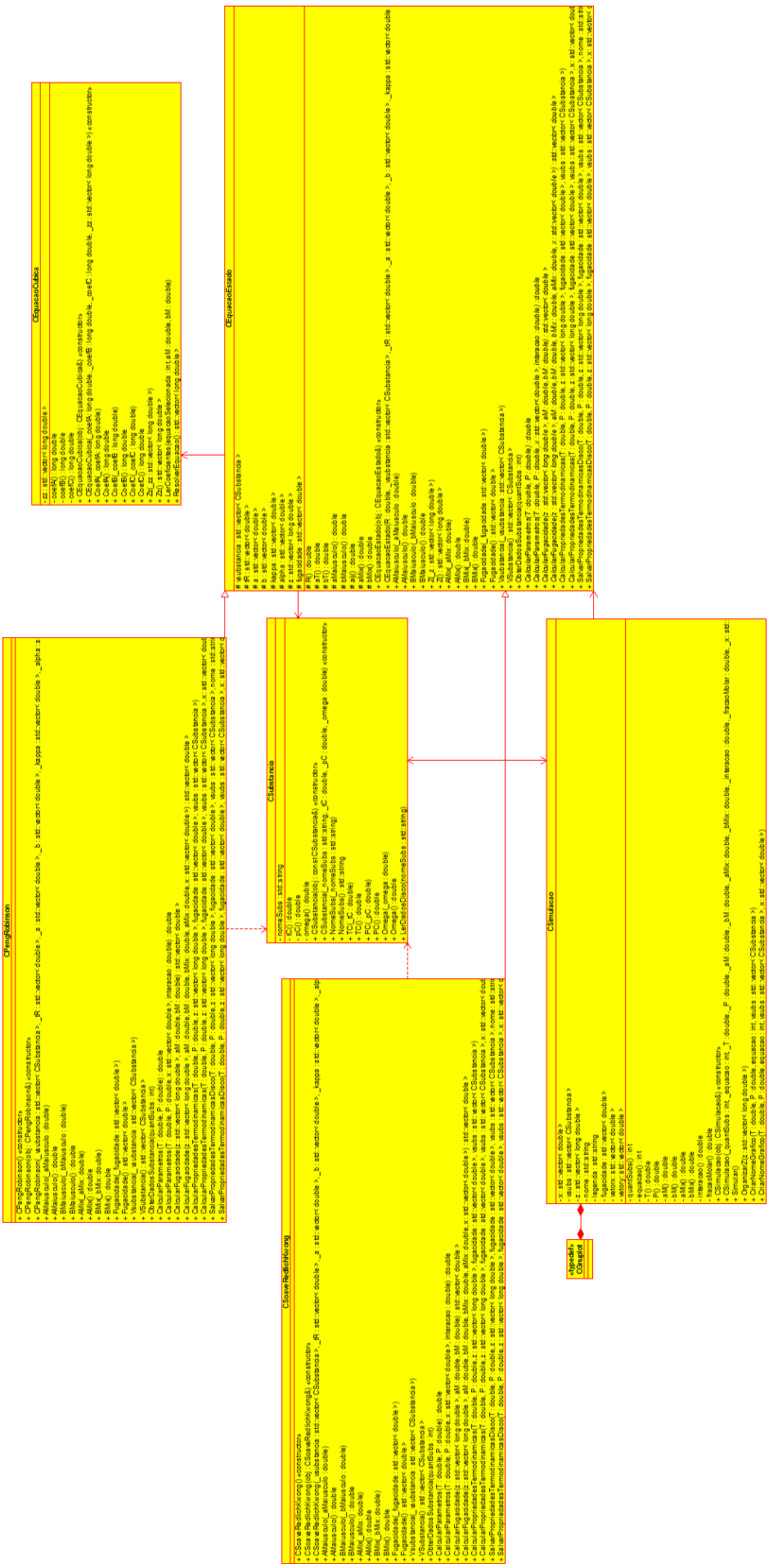


Figura 4.1: Diagrama de classes

### 4.1.1 Dicionário de classes

- Classe CSubstancia: Esta classe é onde ficam armazenadas as propriedades termodinâmicas de cada uma das substâncias presentes na produção de petróleo. Ela deve fornecer as informações necessárias para resolver a equação de estado selecionada (temperatura crítica, pressão crítica e fator acêntrico).
- Classe CEquacaoCubica: Esta classe fornece um método não numérico de resolução de uma equação de terceiro grau.
- Classe CEquacaoEstado: Esta classe reúne os atributos e métodos necessários para obter os dados de uma substância, fazer o cálculo dos parâmetros das equações de estado e das propriedades termodinâmicas, mostrar os resultados na tela do usuário e salvá-los em disco.
- Classe CPengRobinson: Esta classe herda de CEquacaoEstado todos os atributos necessários para montar a equação Peng-Robinson.
- Classe CSoaveRedlichKwong: Esta classe herda de CEquacaoEstado todos os atributos necessários para montar a equação Soave-Redlich-Kwong.
- Classe CGnuplot: Esta classe contém um conjunto de instruções para plotar o gráfico utilizado em CSimulacao.
- Classe CSimulacao: Esta classe utiliza todas as classes anteriores para simular as propriedades termodinâmicas das substâncias nas dadas temperatura e pressão.

## 4.2 Diagrama de seqüência – eventos e mensagens

Mostra a sequência temporal pela qual as informações passam de uma classe para outra.

### 4.2.1 Diagrama de sequência geral

Veja o diagrama de seqüência na Figura 4.2. Ele representa uma ordem temporal pela qual as classes de relacionam entre si e com o usuário.

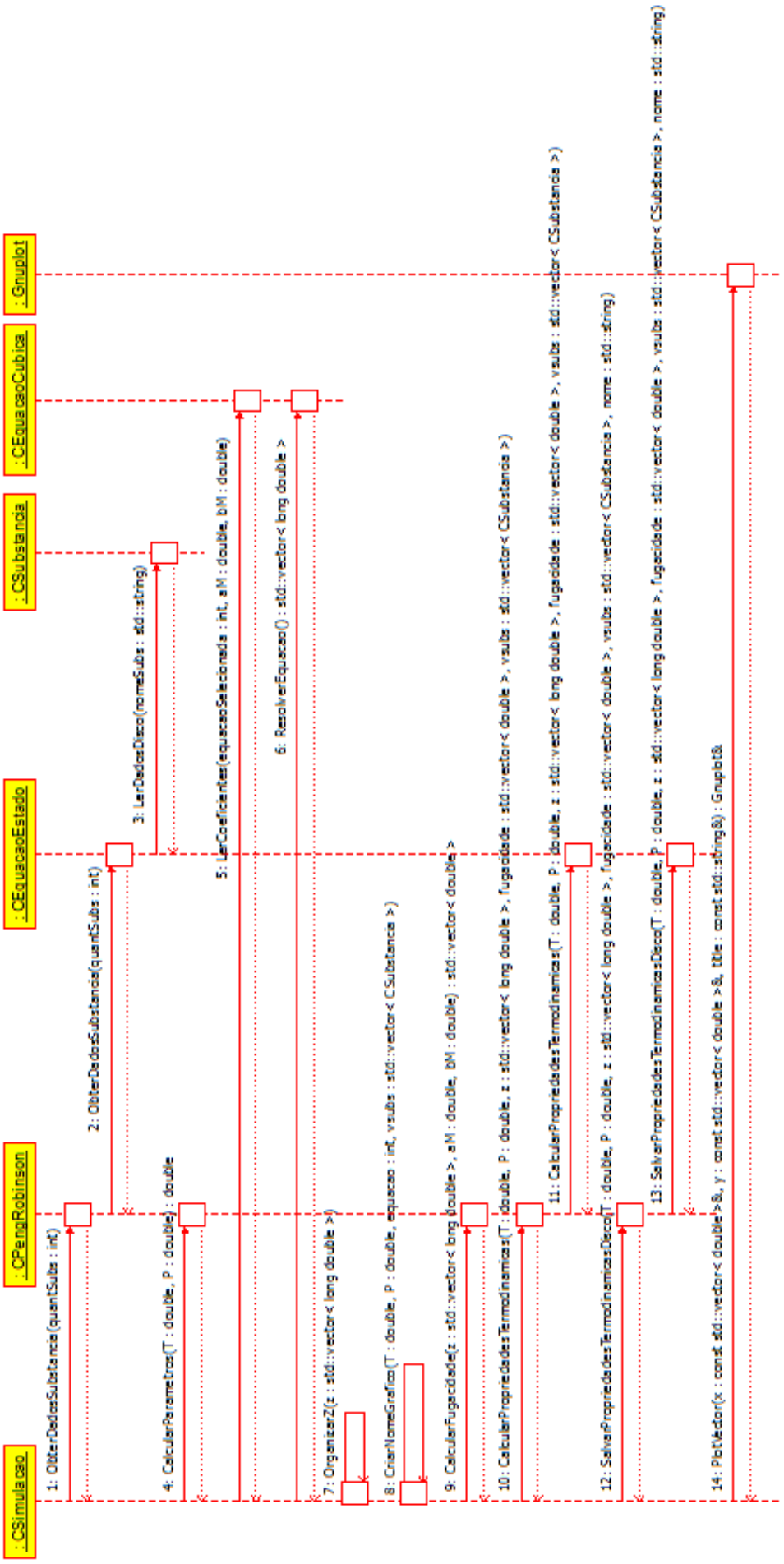



Figura 4.2: Diagrama de seqüência da ordem de chamada das classes por CSimulacao

### 4.2.2 Diagrama de sequência específico

Veja o diagrama de seqüência específico na Figura 4.3. Ele representa a ordem temporal de como CSimulacao utiliza CEquacaoCubica.

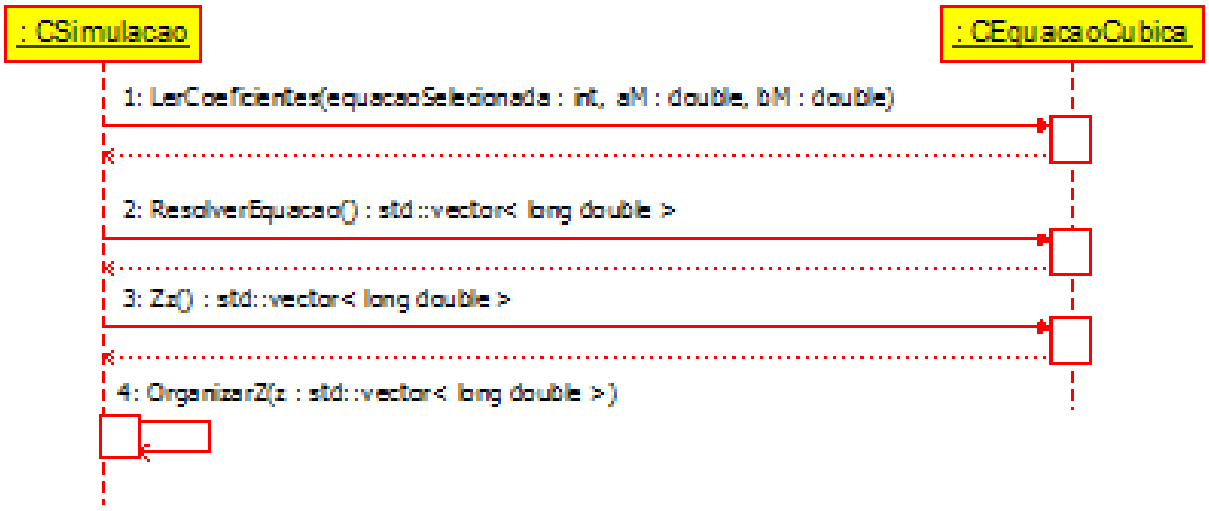



Figura 4.3: Diagrama de seqüência específico da relação entre CSimulacao e CEquacaoCubica

## 4.3 Diagrama de comunicação – colaboração

Veja na Figura 4.3 o diagrama de comunicação mostrando a sequência de colaboração entre as classes para a resolução da esquação de estado cúbica Peng-Robinson e cálculo das propriedades termodinâmicas.

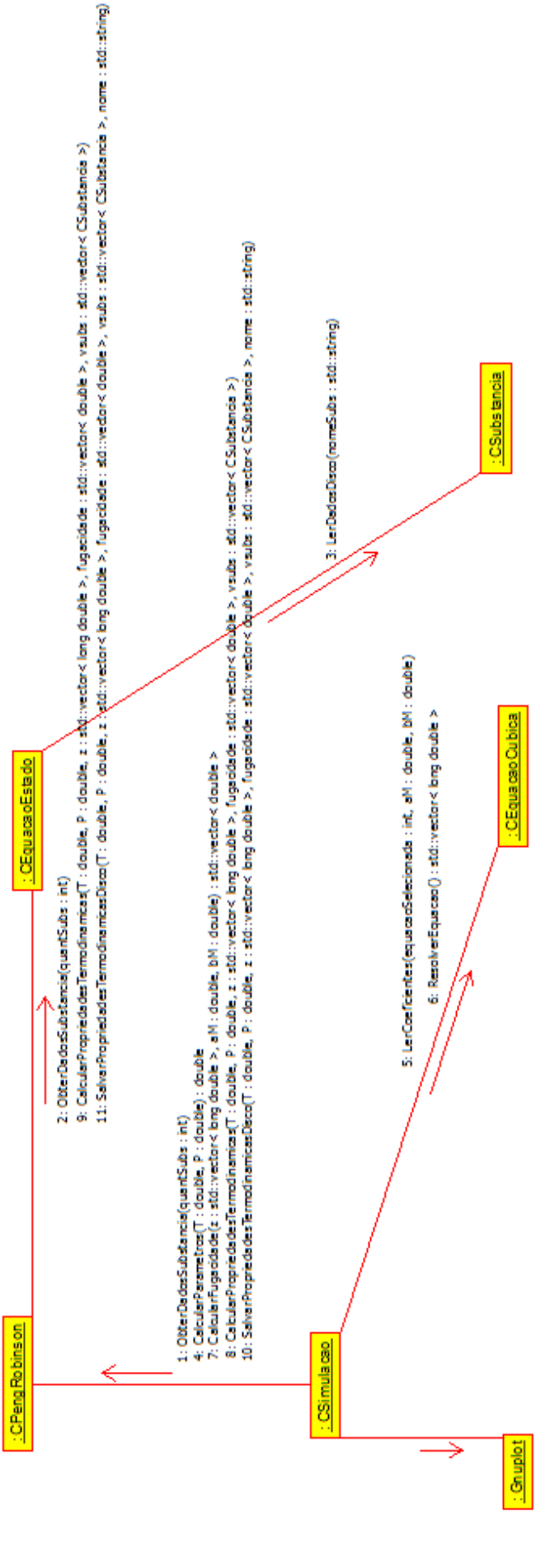



Figura 4.4: Diagrama de comunicação para equação escolha de uma substância e equação Peng-Robinson

## 4.4 Diagrama de máquina de estado

Veja na figura 4.5 o diagrama de máquina de estado. Ele representa os estados pelo qual a classe CPengRobinson passa durante sua execução.

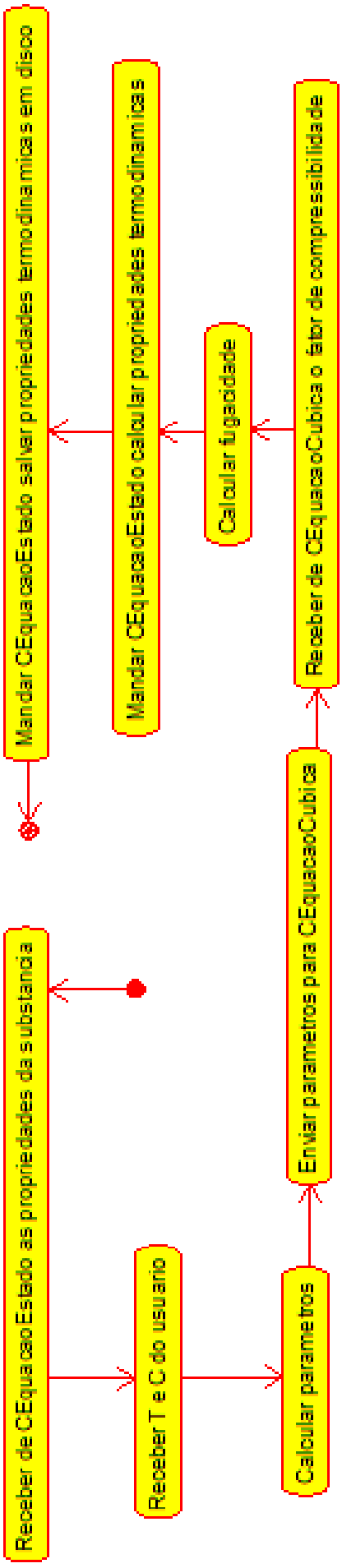



Figura 4.5: Diagrama de máquina de estado para classe CPengRobinson

## 4.5 Diagrama de atividades

Veja na Figura 4.6 o diagrama de atividades. Observe que ele corresponde à atividade específica de calcular parâmetros no diagrama de máquina de estado.

O método CPengRobinson::CalcularParametros(double T, double P) visa o cálculo dos parâmetros A e B, que por sua vez dependem de outros parâmetros. Então este método calcula todos os parâmetros necessários utilizando as informações fornecidas pelo usuário e por CSubstancia, para finalmente chegar no objetivo, A e B. Estes parâmetros A e B serão utilizados para montar uma equação de terceiro grau, que será montada eresolvida pela classe CEquacaoCubica.

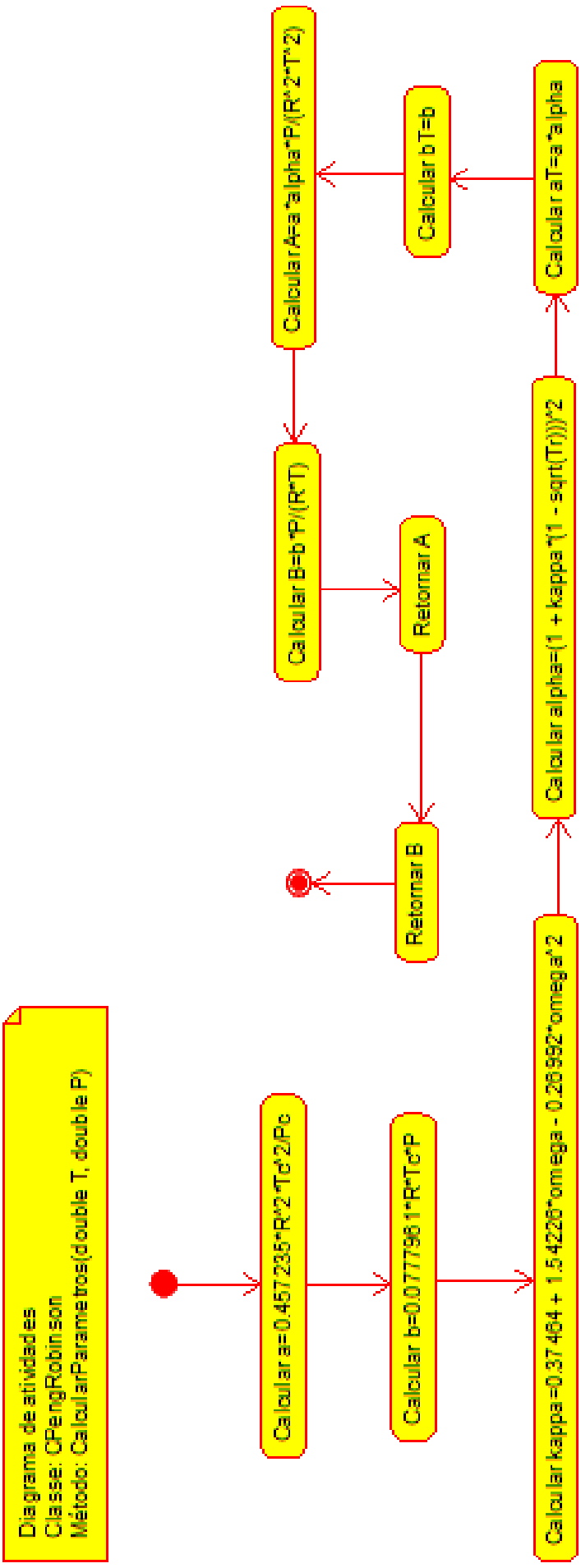



Figura 4.6: Diagrama de atividades para CPengRobinson::CalcularParametros(double T, double P)

# Capítulo 5

# Projeto

Neste capítulo do projeto de engenharia veremos questões associadas ao projeto do sistema, incluindo protocolos, recursos, plataformas suportadas, implicações nos diagramas feitos anteriormente, diagramas de componentes e implantação. Na segunda parte revisamos os diagramas levando em conta as decisões do projeto do sistema.

## 5.1 Projeto do sistema

Depois da análise orientada a objeto desenvolve-se o projeto do sistema, 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 bibliotecas 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.

1. Protocolos

    - O *software* utilizará como componente externo, o Gnuplot. Este tem como finalidade plotar o gráfico das equações resolvidas.
    - O *software* poderá receber os dados somente via teclado.

2. Recursos

    - O *software* requer a utilização de processador, memória *RAM, HD*, teclado e tela.
    - Requer que o *software* externo Gnuplot esteja instalado.

3. Controle

- Não requer muita memória, visto que o programa e seus componentes contém dados que ocupam pouco espaço na memória.
- Não requer processamento paralelo, visto que o programa e seus componentes executam cálculos que requerem pouco poder de processamento.

4. Plataformas

- O *software* irá operar nos sistemas operacionais Windows e GNU/Linux, sendo desenvolvido e testado em ambos os sistemas.
- O ambiente de desenvolvimento será o `Dev C++` (Windows) e `Kate` (Linux).

5. Bibliotecas

- Será utilizada a biblioteca padrão da linguagem C++, incluindo *cmath*, *vector*, *string*, *iostream*, *fstream, locale.h, algorithm, sstream.*
- Iremos utilizar a classe CGnuplot, que fornece acesso ao Gnuplot.

## 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.

**Efeitos do projeto no modelo estrutural**

- A biblioteca gráfica utilizada será CGnuplot. Ela fornece acesso ao *software* externo Gnuplor, utilizado para plotar o gráfico.

**Efeitos do projeto nos métodos**

- Neste projeto, os métodos de classe exercem a função de armazenar valores nos atributos, calcular ou executar atividades que suas respectivas classes propõem.

**Efeitos do projeto nas associações**

- As classes CPengRobinson e CSoaveRedlichKwong reunirão as informações contidas em CSubstancia, e utilizarão os métodos contidos em CEquacaoEstadoCubica para calcular parâmetros de uma equação de estado cúbica. As raízes desta equação cúbica serão encontradas por CEquacaoCubica, que retornará os valores para CPengRobinson ou CSoaveRedlichKwong para que possam calcular as propriedades termodinâmicas desejadas.

## 5.3 Diagrama de componentes

O diagrama de componentes mostra a forma como os componentes do software se relacionam, suas dependências. Inclui itens como: componentes, subsistemas, executáveis, nós, associações, dependências, generalizações, restrições e notas. Exemplos de componentes são bibliotecas estáticas, bibliotecas dinâmicas, dlls, componentes Java, executáveis, arquivos de disco, código-fonte.

Veja na Figura 5.1 o diagrama de componentes para nosso *software*. Observe que este inclui muitas dependências, ilustrando as relações entre os arquivos.

Algumas observações úteis para o diagrama de componentes:

- De posse do diagrama de componentes, temos a lista de todos os arquivos necessários para compilar e rodar o software.
- Observe que um assunto/pacote pode se transformar em uma biblioteca e será incluído no diagrama de componentes.

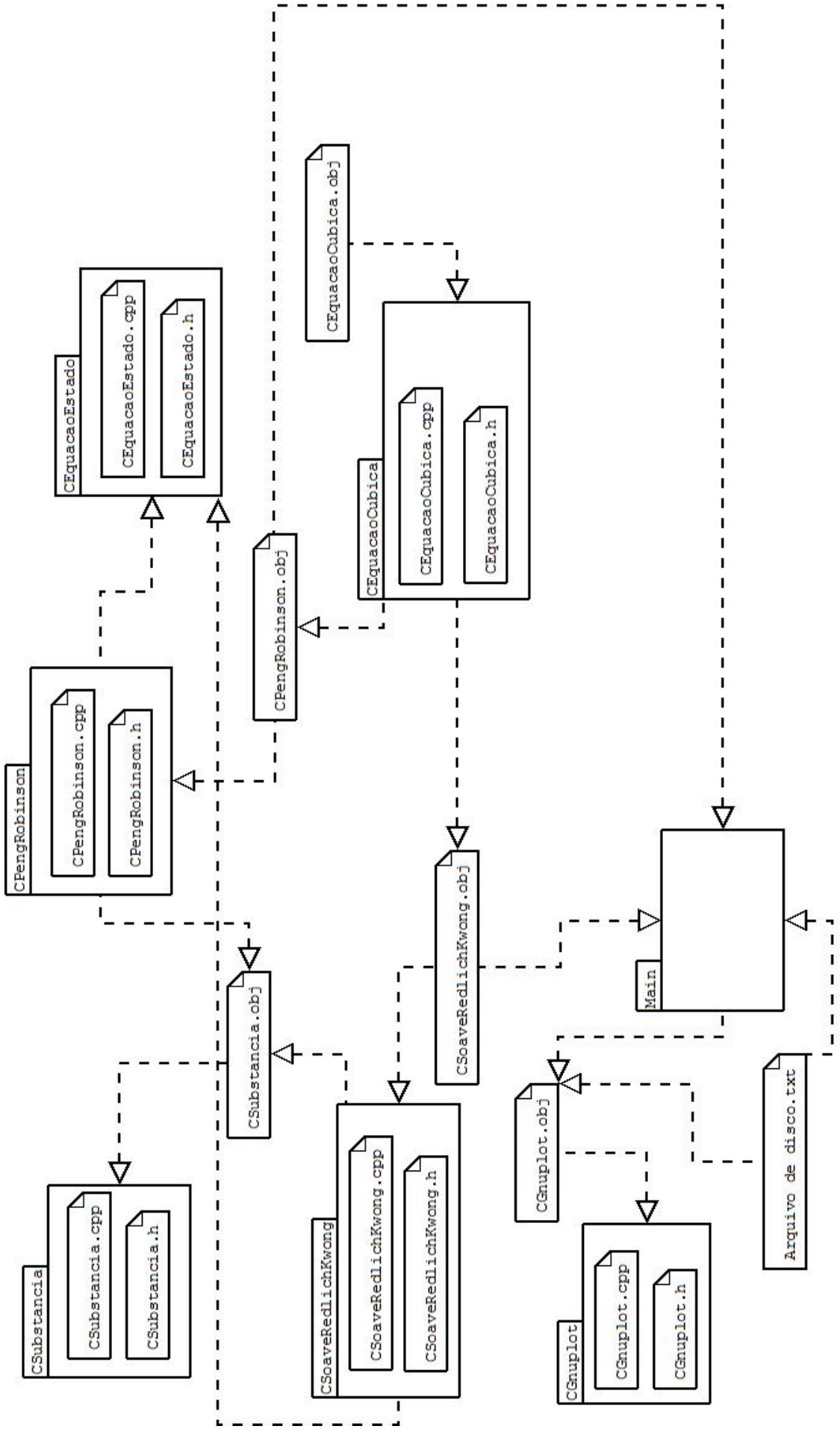



Figura 5.1: Diagrama de componentes do software Simulação de Propriedades Termodinâmicas de Substâncias Simples ou Compostas

## 5.4 Diagrama de implantação

O diagrama de implantação é um diagrama de alto nível que inclui relações entre o sistema e o hardware e que se preocupa com os aspectos da arquitetura computacional escolhida. Seu enfoque é o hardware, a configuração dos nós em tempo de execução.

O diagrama de implantação deve incluir os elementos necessários para que o sistema seja colocado em funcionamento: computador, periféricos, processadores, dispositivos, nós, relacionamentos de dependência, associação, componentes, subsistemas, restrições e notas.

Veja na Figura 5.2 o diagrama de implantação do programa.

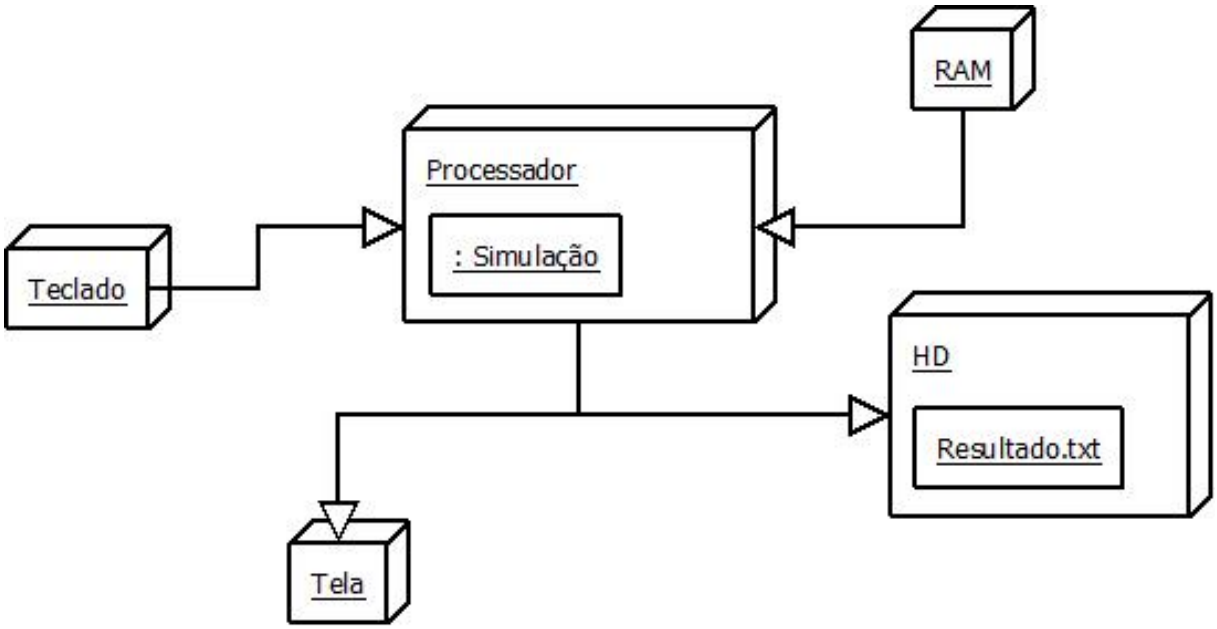



Figura 5.2: Diagrama de implantação.

# 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 `CSubstancia`.

Listing 6.1: Arquivo de cabeçalho da classe CSubstancia.

```
#ifndef CSubstancia_h
#define CSubstancia_h
#include <iostream>
#include <cmath>
#include <vector>
#include <string>

class CSubstancia{

private:

//Atributos
std::string nomeSubs; ///Nome da substancia
double tC{0}; ///Temperatura critica
double pC{0}; ///Pressao critica
double omega{0}; ///Fator acentrico

public:

//Construtores
CSubstancia()=default;
CSubstancia(const CSubstancia& obj);
```

```
23 CSubstancia(std::string _nomeSubs, double _tC, double _pC, double _omega
       );
24 ~CSubstancia()=default;
25
26 //Metodos Get e Set
27 void NomeSubs(std::string _nomeSubs){
28         nomeSubs=_nomeSubs;}
29
30 std::string NomeSubs(){
31         return nomeSubs;}
32
33 void TC(double _tC){
34         tC=_tC;}
35
36 double TC(){
37         return tC;}
38
39 void PC(double _pC){
40         pC=_pC;}
41
42 double PC(){
43         return pC;}
44
45 void Omega(double _omega){
46         omega=_omega;}
47
48 double Omega(){
49         return omega;}
50
51 //Metodos
52 void LerDadosDisco(std::string nomeSubs);
53
54 friend class CPengRobinson;
55 friend class CSoaveRedlichKwong;
56
57 };
58 #endif
```

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

Listing 6.2: Arquivo de implementação da classe CSubstancia.

```
59 #include <iostream>
60 #include <sstream>
61 #include <locale.h>
62 #include <cmath>
63 #include <vector>
64 #include <string>
65 #include <fstream>
66 #include "CSubstancia.h"
```

```
67
68 using namespace std;
69
70 //Construtores
71 CSubstancia::CSubstancia(const CSubstancia& obj){
72         nomeSubs=obj.nomeSubs;
73         tC=obj.tC;
74         pC=obj.pC;
75         omega=obj.omega;
76 }
77
78 CSubstancia::CSubstancia(string _nomeSubs, double _tC, double _pC,
       double _omega){
79         nomeSubs=_nomeSubs;
80         tC=_tC;
81         pC=_pC;
82         omega=_omega;
83 }
84
85 //Le o arquivo em disco e salva as propriedades da substancia
86 void CSubstancia::LerDadosDisco(string nomeSubs){
87         NomeSubs(nomeSubs);
88         ifstream fin;
89         string lixo="␣";
90         fin.open("Substancias.txt");
91         if(!fin) exit(0);
92         while( lixo != nomeSubs and !fin.eof()){
93                 fin >> lixo;
94         }
95         fin >> tC;
96         fin >> pC;
97         fin >> omega;
98         fin.close();
99 }
```

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

Listing 6.3: Arquivo de cabeçalho da classe CEquacaoCubica.

```
100 #ifndef CEquacaoCubica_h
101 #define CEquacaoCubica_h
102 #include "CEquacaoEstado.h"
103 #include "CPengRobinson.h"
104 #include "CSoaveRedlichKwong.h"
105 #include <iostream>
106 #include <cmath>
107 #include <vector>
108
109 class CEquacaoCubica {
110
```

```
111 private:
112
113 long double coefA{0}; ///Coeficiente A
114 long double coefB{0}; ///Coeficiente B
115 long double coefC{0}; ///Coeficiente C
116 std::vector<long double> zz; /// Fator de compressibilidade
117
118 public:
119
120 //Construtores e Destrutor
121 CEquacaoCubica() = default;
122 CEquacaoCubica(CEquacaoCubica& obj);
123 CEquacaoCubica(long double _coefA, long double _coefB, long double
      _coefC, std::vector<long double> _zz);
124 ~CEquacaoCubica() = default;
125
126 //Métodos Get e Set
127 void CoefA(long double _coefA){
128         coefA=_coefA;}
129
130 long double CoefA(){
131         return coefA;}
132
133 void CoefB(long double _coefB){
134         coefB = _coefB;}
135
136 long double CoefB(){
137         return coefB;}
138
139 void CoefC(long double _coefC){
140         coefC = _coefC;}
141
142 long double CoefC(){
143         return coefC;}
144
145 void Zz(std::vector<long double> _zz){
146         zz = _zz;}
147
148 std::vector<long double> Zz(){
149         return zz;}
150
151 //Métodos
152 void LerCoeficientes(int equacaoSelecionada, double aM, double bM);
153 std::vector<long double> ResolverEquacao();
154
155 };
156
157 #endif
```

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

Listing 6.4: Arquivo de implementação da classe CEquacaoCubica.

```
158 #include <iostream>
159 #include <cmath>
160 #include <vector>
161 #include <algorithm>
162 #include "CEquacaoCubica.h"
163 #include "CEquacaoEstado.h"
164
165 using namespace std;
166
167 //Construtores
168 CEquacaoCubica::CEquacaoCubica(CEquacaoCubica& obj){
169         coefA=obj.coefA;
170         coefB=obj.coefB;
171         coefC=obj.coefC;
172         zz=obj.zz;
173 }
174
175 CEquacaoCubica::CEquacaoCubica(long double _coefA, long double _coefB,
      long double _coefC, std::vector<long double> _zz){
176         coefA=_coefA;
177         coefB=_coefB;
178         coefC=_coefC;
179         zz=_zz;
180 }
181
182 //Organizar aM e bM em uma equacao cubica no formato PR ou SRK
183 void CEquacaoCubica::LerCoeficientes(int equacaoSelecionada, double aM,
      double bM){
184         if (equacaoSelecionada==1){          //Peng-Robinson: z^3 - (1 -
              B) * z^2 + (A - 2*B - 3*B^2) * z - (A * B - B^2 - B^3)
185                 coefA=bM - 1.;
186                 coefB=aM - 2.*bM - 3.*pow(bM,2.);
187                 coefC=-(aM*bM - pow(bM,2.) - pow(bM,3.));
188         }
189 else{                                       //Soave-Redlich-Kwong: z^3 - z^2
      + (A - B - B^2) * z - (A*B)
190                 coefA=-1;
191                 coefB=aM - bM - pow(bM,2.);
192                 coefC=-(aM*bM);
193         }
194 }
195 //Resolve equacao cubica montada pelo metodo anterior
196 std::vector<long double> CEquacaoCubica::ResolverEquacao(){
197         long double coefD, coefE, delta;
198         coefD=pow(coefA/3.,3.) - (coefA*coefB/6.) + (coefC/2.);
```

```
199         coefE=(coefB/3) - pow(coefA/3,2);
200         delta=pow(coefD,2.) + pow(coefE,3.);
201         if (delta==0){
202                 zz.push_back(2.*pow(-coefD,(1./3.)) - (coefA/3.));
203                 zz.push_back(-pow(-coefD,(1./3.)) - (coefA/3.));
204                 zz.push_back(zz[1]);
205         }
206         else
207                 if (delta>0){
208                         long double coefF,coefG;
209                         long double cte=(-coefD)+sqrt(delta);
210                         if (cte<0)
211                                 coefF=-pow(abs(cte),1./3.);
212                         else
213                                 coefF=pow(cte,1./3.);
214                         cte=(-coefD)-sqrt(delta);
215                         if (cte<0)
216                                 coefG=-pow(abs(cte),1./3.);
217                         else
218                                 coefG=pow(cte,1./3.);
219                         for(int i=0;i<3;i++)
220                         zz.push_back(coefF + coefG - (coefA/3.));
221                         }
222                 else{
223                         long double theta;
224                         theta=acos(-coefD/sqrt(pow(-coefE,3.)));
225                         zz.push_back(2.*sqrt(-coefE)*cos(theta/3.) -
                                coefA/3.);
226                         zz.push_back(2.*sqrt(-coefE)*cos(theta/3. + 2.*
                                M_PI/3.) - coefA/3.);
227                         zz.push_back(2.*sqrt(-coefE)*cos(theta/3. + 4.*
                                M_PI/3.) - coefA/3.);
228                 }
229 return zz;}
```

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

Listing 6.5: Arquivo de cabeçalho da classe CEquacaoEstado.

```
230 #ifndef CEquacaoEstado_h
231 #define CEquacaoEstado_h
232 #include "CSubstancia.h"
233 #include <iostream>
234 #include <fstream>
235 #include <cmath>
236 #include <vector>
237 #include <string>
238
239 class CEquacaoEstado{
240
```

```
241 protected:
242 //Atributos
243 double R{8.314e-5}; ///Constante dos gases
244 std::vector<CSubstancia> vsubstancia; ///Vetor de substancia
245 std::vector<double> tR; ///Temperatura Critica
246 std::vector<double> a; ///a
247 std::vector<double> b; ///b
248 std::vector<double> kappa; ///kappa
249 std::vector<double> alpha; ///alpha
250 double aT{0}; ///aT
251 double bT{0}; ///bT
252 double aMaiusculo{0}; ///aMaiusculo
253 double bMaiusculo{0}; ///bMaiusculo
254 double aij{0}; ///aij
255 double aMix{0}; ///aMix
256 double bMix{0}; ///bMix
257 std::vector<long double> z; ///Fator de compressibilidade
258 std::vector<double> fugacidade; ///Fugacidade
259
260 public:
261
262 //Construtores e Destrutor
263 CEquacaoEstado()=default;
264 CEquacaoEstado(CEquacaoEstado& obj);
265 CEquacaoEstado(double R, std::vector<CSubstancia> _vsubstancia, std::
      vector<double> _tR, std::vector<double> _a, std::vector<double> _b,
      std::vector<double> _kappa, std::vector<double> _alpha, double _aT,
      double _bT, double _aMaiusculo, double _bMaiusculo, double _aij,
      double _interecao, double _aMix, double _bMix, std::vector<long
      double> _z, std::vector<double> _fugacidade);
266 ~CEquacaoEstado()=default;
267
268 //Métodos Get e Set
269 void AMaiusculo(double _aMaiusculo){
270         aMaiusculo=_aMaiusculo;}
271
272 double AMaiusculo(){
273         return aMaiusculo;}
274
275 void BMaiusculo(double _bMaiusculo){
276         bMaiusculo=_bMaiusculo;}
277
278 double BMaiusculo(){
279         return bMaiusculo;}
280
281 void Z(std::vector<long double> _z){
282         z = _z;}
283
```

```
284 std::vector<long double> Z(){
285         return z;}
286
287 void AMix(double _aMix){
288         aMix=_aMix;}
289
290 double AMix(){
291         return aMix;}
292
293 void BMix(double _bMix){
294         bMix=_bMix;}
295
296 double BMix(){
297         return bMix;}
298
299 void Fugacidade(std::vector<double> _fugacidade){
300         fugacidade = _fugacidade;}
301
302 std::vector<double> Fugacidade(){
303         return fugacidade;}
304
305 void Vsubstancia(std::vector<CSubstancia> _vsubstancia){
306         vsubstancia = _vsubstancia;}
307
308 std::vector<CSubstancia> VSubstancia(){
309         return vsubstancia;}
310
311 //Métodos
312 void ObterDadosSubstancia(int quantSubs);
313 virtual double CalcularParametros(double T, double P)=0;
314 virtual double CalcularParametros(double T, double P, std::vector<double
      > x, double interacao)=0;
315 virtual std::vector<double> CalcularFugacidade(std::vector<long double>
      z, double aM, double bM)=0;
316 virtual std::vector<double> CalcularFugacidade(std::vector<long double>
      z, double aM, double bM, double bMix, double aMix, std::vector<double
      > x)=0;
317 void CalcularPropriedadesTermodinamicas(double T, double P, std::vector<
      long double> z, std::vector<double> fugacidade, std::vector<
      CSubstancia> vsubs);
318 void CalcularPropriedadesTermodinamicas(double T, double P, std::vector<
      long double> z, std::vector<double> fugacidade, std::vector<
      CSubstancia> vsubs, std::vector<double> x);
319 void SalvarPropriedadesTermodinamicasDisco(double T, double P, std::
      vector<long double> z, std::vector<double> fugacidade, std::vector<
      CSubstancia> vsubs, std::string nome);
320 void SalvarPropriedadesTermodinamicasDisco(double T, double P, std::
      vector<long double> z, std::vector<double> fugacidade, std::vector<
```

```
    CSubstancia> vsubs, std::vector<double> x, std::string nome);
};
#endif
```

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

Listing 6.6: Arquivo de implementação da classe CEquacaoEstado.

```
#include <iostream>
#include <cmath>
#include <vector>
#include <string>
#include <iomanip>
#include <locale>
#include <sstream>
#include "CEquacaoEstado.h"
#include <algorithm>

using namespace std;

//Construtores
CEquacaoEstado::CEquacaoEstado(CEquacaoEstado& obj){
        vsubstancia=obj.vsubstancia;
        tR=obj.tR;
        a=obj.a;
        b=obj.b;
        kappa=obj.kappa;
        alpha=obj.alpha;
        aT=obj.aT;
        bT=obj.bT;
        aMaiusculo=obj.aMaiusculo;
        bMaiusculo=obj.bMaiusculo;
        aij=obj.aij;
        aMix=obj.aMix;
        bMix=obj.bMix;
        z=obj.z;
        fugacidade=obj.fugacidade;
        R=8.314e-5;
}

CEquacaoEstado::CEquacaoEstado(double R, vector<CSubstancia>
    _vsubstancia, vector<double> _tR, vector<double> _a, vector<double>
    _b, vector<double> _kappa, vector<double> _alpha, double _aT, double
    _bT, double _aMaiusculo, double _bMaiusculo, double _aij, double
    _interecao, double _aMix, double _bMix, vector<long double> _z,
    vector<double> _fugacidade){
        vsubstancia=_vsubstancia;
        tR=_tR;
        a=_a;
        b=_b;
```

```
362         kappa=_kappa;
363         alpha=_alpha;
364         aT=_aT;
365         bT=_bT;
366         aMaiusculo=_aMaiusculo;
367         bMaiusculo=_bMaiusculo;
368         aij=_aij;
369         aMix=_aMix;
370         bMix=_bMix;
371         z=_z;
372         fugacidade=_fugacidade;
373         R=8.314e-5;
374 }
375
376 //Obter os dados da substancia selecionada pelo usuario e armazenar no
      vetor vsubstancia
377 void CEquacaoEstado::ObterDadosSubstancia(int quantSubs){
378         CSubstancia subs;
379         string nomeSubs;
380         cout << "\nAs␣substancias␣possiveis␣sao:␣\nmetano␣\netano␣\
               npropano␣\nisobutano␣\nnbutano␣\nisopentano␣\nnpentano␣\
               nnhexano␣\nnheptano␣\nnoctano␣\nndecano␣\ndodecano␣\nco2␣\
               nnitrogenio␣\nh2s␣\nh2o␣\n\n";
381         for (int i=0;i<quantSubs;i++){
382                 cout << "\nQual␣a␣substancia␣" << i+1 << "␣?" << endl;
383                 getline (cin, nomeSubs);
384                 subs.LerDadosDisco(nomeSubs);
385                 vsubstancia.push_back(subs);
386                 }
387 }
388
389 //Calcula Propriedades termodinamicas para 1 substancia
390 void CEquacaoEstado::CalcularPropriedadesTermodinamicas(double T, double
       P, vector<long double> z, std::vector<double> fugacidade, std::
      vector<CSubstancia> vsubs){
391         vector<double> densidade;
392         for (int i=0;i<2;i++)
393                 densidade.push_back(P/(R*T*z[i]));
394         cout << "Molecula:␣"<< vsubs[0].NomeSubs()<<endl;
395         cout << "Temperatura␣Critica:␣"<< vsubs[0].TC()<< "␣K"<<endl;
396         cout << "Pressao␣Critica:␣"<<vsubs[0].PC()<< "␣bar"<<endl;
397         cout << "Fator␣acentrico:␣"<<vsubs[0].Omega()<<endl;
398         if (z[0]==z[1]){
399                 cout << "So␣existe␣uma␣fase."<<endl;
400                 cout << "Fator␣de␣compressibilidade:␣"<<z[0]<<endl;
401                 cout << "Coeficiente␣de␣fugacidade:␣"<< fugacidade[0]<<
                       endl;
402                 cout << "Fugacidade␣em␣"<< P << "␣bar:␣" << fugacidade
```

```
                    [0]*P << " bar"<<endl;
403                 cout << "Densidade: " << densidade[0]<< " mol/m^3"<<endl
                    ;
404                 cout << "Volume especifico: " << 1./densidade[0]<< " m
                    ^3/mol"<<endl;
405                 cout << "Volume molar: " << 1000./densidade[0]<< " L/mol
                    "<<endl;
406         }
407         else{
408                 cout << "As fases liquido e vapor coexistem."<<endl;
409                 cout << "Fator de compressibilidade (liquido): "<<z[0]<<
                    endl;
410                 cout << "Fator de compressibilidade (vapor): "<<z[1]<<
                    endl;
411                 cout << "Coeficiente de fugacidade (liquido): "<<
                    fugacidade[0]<<endl;
412                 cout << "Coeficiente de fugacidade (vapor): "<<
                    fugacidade[1]<<endl;
413                 cout << "Fugacidade em "<< P << " bar (liquido): " <<
                    fugacidade[0]*P<< " bar"<<endl;
414                 cout << "Fugacidade em "<< P << " bar (vapor): " <<
                    fugacidade[1]*P<< " bar"<<endl;
415                 cout << "Densidade (liquido): " << densidade[0]<< " mol/
                    m^3"<<endl;
416                 cout << "Densidade (vapor): " << densidade[1]<< " mol/m
                    ^3"<<endl;
417                 cout << "Volume especifico (liquido): " << 1./densidade
                    [0]<< " m^3/mol"<<endl;
418                 cout << "Volume especifico (vapor): " << 1./densidade
                    [1]<< " m^3/mol"<<endl;
419                 cout << "Volume molar (liquido): " << 1000./densidade
                    [0]<< " L/mol"<<endl;
420                 cout << "Volume molar (vapor): " << 1000./densidade[1]<<
                     " L/mol"<<endl;
421         }
422 }
423
424 //Calcula Propriedades termodinamicas para 2 substancias
425 void CEquacaoEstado::CalcularPropriedadesTermodinamicas(double T, double
       P, vector<long double> z, std::vector<double> fugacidade, std::
      vector<CSubstancia> vsubs, std::vector<double> x){
426         vector<double> densidade;
427         for (int i=0;i<2;i++)
428                 densidade.push_back(P/(R*T*z[i]));
429         if (z[0]==z[1]){
430                 cout << "So existe uma fase."<<endl;
431                 cout << "Densidade: " << densidade[0]<< " mol/m^3"<<endl
                    ;
```

```
432                 cout << "Volume␣especifico:␣" << 1./densidade[0]<< "␣m
                        ^3/mol"<<endl;
433                 cout << "Volume␣molar:␣" << 1000./densidade[0]<< "␣L/mol
                        "<<endl;
434                 cout << "Fator␣de␣compressibilidade:␣"<<z[0]<<endl;
435                 cout << "Componente␣A:␣"<< vsubs[0].NomeSubs()<<endl;
436                 cout << "Coeficiente␣de␣fugacidade:␣"<< fugacidade[0]<<
                        endl;
437                 cout << "Fugacidade␣em␣"<< P << "␣bar:␣" << fugacidade
                        [0]*P*x[0]<< "␣bar"<<endl;
438                 cout << "Componente␣B:␣"<< vsubs[1].NomeSubs()<<endl;
439                 cout << "Coeficiente␣de␣fugacidade:␣"<< fugacidade[1]<<
                        endl;
440                 cout << "Fugacidade␣em␣"<< P << "␣bar:␣" << fugacidade
                        [1]*P*x[1]<< "␣bar"<<endl;
441         }
442         else{
443                 cout << "As␣fases␣liquido␣e␣vapor␣coexistem."<<endl;
444                 cout << "Densidade␣(liquido):␣" << densidade[0]<< "␣mol/
                        m^3"<<endl;
445                 cout << "Densidade␣(vapor):␣" << densidade[1]<< "␣mol/m
                        ^3"<<endl;
446                 cout << "Volume␣especifico␣(liquido):␣" << 1./densidade
                        [0]<< "␣m^3/mol"<<endl;
447                 cout << "Volume␣especifico␣(vapor):␣" << 1./densidade
                        [1]<< "␣m^3/mol"<<endl;
448                 cout << "Volume␣molar␣(liquido):␣" << 1000./densidade
                        [0]<< "␣L/mol"<<endl;
449                 cout << "Volume␣molar␣(vapor):␣" << 1000./densidade[1]<<
                         "␣L/mol"<<endl;
450                 cout << "Fator␣de␣compressibilidade␣(liquido):␣"<<z[0]<<
                        endl;
451                 cout << "Fator␣de␣compressibilidade␣(vapor):␣"<<z[1]<<
                        endl;
452                 cout << "Componente␣A:␣"<< vsubs[0].NomeSubs()<<endl;
453                 cout << "Coeficiente␣de␣fugacidade␣(liquido):␣"<<
                        fugacidade[0]<<endl;
454                 cout << "Fugacidade␣em␣"<< P << "␣bar␣(liquido):␣" <<
                        fugacidade[0]*P*x[0]<< "␣bar"<<endl;
455                 cout << "Coeficiente␣de␣fugacidade␣(vapor):␣"<<
                        fugacidade[1]<<endl;
456                 cout << "Fugacidade␣em␣"<< P << "␣bar␣(vapor):␣" <<
                        fugacidade[1]*P*x[0]<< "␣bar"<<endl;
457                 cout << "Componente␣B:␣"<< vsubs[1].NomeSubs()<<endl;
458                 cout << "Coeficiente␣de␣fugacidade␣(liquido):␣"<<
                        fugacidade[2]<<endl;
459                 cout << "Fugacidade␣em␣"<< P << "␣bar␣(liquido):␣" <<
                        fugacidade[2]*P*x[1]<< "␣bar"<<endl;
```

```
460                 cout << "Coeficiente␣de␣fugacidade␣(vapor):␣"<<
                        fugacidade[3]<<endl;
461                 cout << "Fugacidade␣em␣"<< P << "␣bar␣(vapor):␣" <<
                        fugacidade[3]*P*x[1]<< "␣bar"<<endl;
462         }
463 }
464
465 //Salvar em disco propriedades termodinamicas para 1 substancia
466 void CEquacaoEstado::SalvarPropriedadesTermodinamicasDisco(double T,
      double P, vector<long double> z, std::vector<double> fugacidade, std
      ::vector<CSubstancia> vsubs, std::string nome){
467         vector<double> densidade;
468         for (int i=0;i<2;i++)
469                 densidade.push_back(P/(R*T*z[i]));
470         nome=nome+".txt";
471         ofstream fout;
472         fout.open(nome.c_str());
473         fout << "Simulacao␣de␣" << nome << endl;
474         fout << "Molecula:␣"<< vsubs[0].NomeSubs()<<endl;
475         fout << "Temperatura␣Critica:␣"<< vsubs[0].TC()<< "␣K"<<endl;
476         fout << "Pressao␣Critica:␣"<<vsubs[0].PC()<< "␣bar"<<endl;
477         fout << "Fator␣acentrico:␣"<<vsubs[0].Omega()<<endl;
478         if (z[0]==z[1]){
479                 fout << "So␣existe␣uma␣fase."<<endl;
480                 fout << "Fator␣de␣compressibilidade:␣"<<z[0]<<endl;
481                 fout << "Coeficiente␣de␣fugacidade:␣"<< fugacidade[0]<<
                        endl;
482                 fout << "Fugacidade␣em␣"<< P << "␣bar:␣" << fugacidade
                        [0]*P << "␣bar"<<endl;
483                 fout << "Densidade:␣" << densidade[0]<< "␣mol/m^3"<<endl
                        ;
484                 fout << "Volume␣especifico:␣" << 1./densidade[0]<< "␣m
                        ^3/mol"<<endl;
485                 fout << "Volume␣molar:␣" << 1000./densidade[0]<< "␣L/mol
                        "<<endl;
486         }
487         else{
488                 fout << "As␣fases␣liquido␣e␣vapor␣coexistem."<<endl;
489                 fout << "Fator␣de␣compressibilidade␣(liquido):␣"<<z[0]<<
                        endl;
490                 fout << "Fator␣de␣compressibilidade␣(vapor):␣"<<z[1]<<
                        endl;
491                 fout << "Coeficiente␣de␣fugacidade␣(liquido):␣"<<
                        fugacidade[0]<<endl;
492                 fout << "Coeficiente␣de␣fugacidade␣(vapor):␣"<<
                        fugacidade[1]<<endl;
493                 fout << "Fugacidade␣em␣"<< P << "␣bar␣(liquido):␣" <<
                        fugacidade[0]*P<< "␣bar"<<endl;
```

```
494                 fout << "Fugacidade␣em␣"<< P << "␣bar␣(vapor):␣" <<
                        fugacidade[1]*P<< "␣bar"<<endl;
495                 fout << "Densidade␣(liquido):␣" << densidade[0]<< "␣mol/
                        m^3"<<endl;
496                 fout << "Densidade␣(vapor):␣" << densidade[1]<< "␣mol/m
                        ^3"<<endl;
497                 fout << "Volume␣especifico␣(liquido):␣" << 1./densidade
                        [0]<< "␣m^3/mol"<<endl;
498                 fout << "Volume␣especifico␣(vapor):␣" << 1./densidade
                        [1]<< "␣m^3/mol"<<endl;
499                 fout << "Volume␣molar␣(liquido):␣" << 1000./densidade
                        [0]<< "␣L/mol"<<endl;
500                 fout << "Volume␣molar␣(vapor):␣" << 1000./densidade[1]<<
                         "␣L/mol"<<endl;
501         }
502         fout.close();
503 }
504
505 //Salvar em disco Propriedades termodinamicas para 2 substancias
506 void CEquacaoEstado::SalvarPropriedadesTermodinamicasDisco(double T,
      double P, vector<long double> z, std::vector<double> fugacidade, std
      ::vector<CSubstancia> vsubs, std::vector<double> x, std::string nome)
      {
507         vector<double> densidade;
508         for (int i=0;i<2;i++)
509                 densidade.push_back(P/(R*T*z[i]));
510         nome=nome+".txt";
511         ofstream fout;
512         fout.open(nome.c_str());
513         fout << "Simulacao␣de␣" << nome<< endl;
514         if (z[0]==z[1]){
515                 fout << "So␣existe␣uma␣fase."<<endl;
516                 fout << "Densidade:␣" << densidade[0]<< "␣mol/m^3"<<endl
                        ;
517                 fout << "Volume␣especifico:␣" << 1./densidade[0]<< "␣m
                        ^3/mol"<<endl;
518                 fout << "Volume␣molar:␣" << 1000./densidade[0]<< "␣L/mol
                        "<<endl;
519                 fout << "Fator␣de␣compressibilidade:␣"<<z[0]<<endl;
520                 fout << "Componente␣A:␣"<< vsubs[0].NomeSubs()<<endl;
521                 fout << "Coeficiente␣de␣fugacidade:␣"<< fugacidade[0]<<
                        endl;
522                 fout << "Fugacidde␣em␣"<< P << "␣bar:␣" << fugacidade
                        [0]*P*x[0]<< "␣bar"<<endl;
523                 fout << "Componente␣B:␣"<< vsubs[1].NomeSubs()<<endl;
524                 fout << "Coeficiente␣de␣fugacidade:␣"<< fugacidade[1]<<
                        endl;
525                 fout << "Fugacidade␣em␣"<< P << "␣bar:␣" << fugacidade
```

```
                    [1]*P*x[1]<< "␣bar"<<endl;
526         }
527         else{
528                 fout << "As␣fases␣liquido␣e␣vapor␣coexistem."<<endl;
529                 fout << "Densidade␣(liquido):␣" << densidade[0]<< "␣mol/
                    m^3"<<endl;
530                 fout << "Densidade␣(vapor):␣" << densidade[1]<< "␣mol/m
                    ^3"<<endl;
531                 fout << "Volume␣especifico␣(liquido):␣" << 1./densidade
                    [0]<< "␣m^3/mol"<<endl;
532                 fout << "Volume␣especifico␣(vapor):␣" << 1./densidade
                    [1]<< "␣m^3/mol"<<endl;
533                 fout << "Volume␣molar␣(liquido):␣" << 1000./densidade
                    [0]<< "␣L/mol"<<endl;
534                 fout << "Volume␣molar␣(vapor):␣" << 1000./densidade[1]<<
                     "␣L/mol"<<endl;
535                 fout << "Fator␣de␣compressibilidade␣(liquido):␣"<<z[0]<<
                    endl;
536                 fout << "Fator␣de␣compressibilidade␣(vapor):␣"<<z[1]<<
                    endl;
537                 fout << "Componente␣A:␣"<< vsubs[0].NomeSubs()<<endl;
538                 fout << "Coeficiente␣de␣fugacidade␣(liquido):␣"<<
                    fugacidade[0]<<endl;
539                 fout << "Fugacidade␣em␣"<< P << "␣bar␣(liquido):␣" <<
                    fugacidade[0]*P*x[0]<< "␣bar"<<endl;
540                 fout << "Coeficiente␣de␣fugacidade␣(vapor):␣"<<
                    fugacidade[1]<<endl;
541                 fout << "Fugacidade␣em␣"<< P << "␣bar␣(vapor):␣" <<
                    fugacidade[1]*P*x[0]<< "␣bar"<<endl;
542                 fout << "Componente␣B:␣"<< vsubs[1].NomeSubs()<<endl;
543                 fout << "Coeficiente␣de␣fugacidade␣(liquido):␣"<<
                    fugacidade[2]<<endl;
544                 fout << "Fugacidde␣em␣"<< P << "␣bar␣(liquido):␣" <<
                    fugacidade[2]*P*x[1]<< "␣bar"<<endl;
545                 fout << "Coeficiente␣de␣fugacidade␣(vapor):␣"<<
                    fugacidade[3]<<endl;
546                 fout << "Fugacidade␣em␣"<< P << "␣bar␣(vapor):␣" <<
                    fugacidade[3]*P*x[1]<< "␣bar"<<endl;
547         }
548         fout.close();
549 }
```

Apresenta-se na listagem 6.7 o arquivo com código da classe `CPengRobinson`.

Listing 6.7: Arquivo de cabeçalho da classe CPengRobinson.

```
550 #ifndef CPengRobinson_h
551 #define CPengRobinson_h
552 #include "CEquacaoEstado.h"
553 #include <iostream>
```

```
554 #include <fstream>
555 #include <cmath>
556 #include <vector>
557 #include <string>
558
559 class CPengRobinson: public CEquacaoEstado{
560
561 public:
562
563 //Construtor
564 CPengRobinson(): CEquacaoEstado() {};
565 CPengRobinson(CPengRobinson& obj);
566 CPengRobinson(std::vector<CSubstancia> _vsubstancia, std::vector<double>
       _tR, std::vector<double> _a, std::vector<double> _b, std::vector<
      double> _kappa, std::vector<double> _alpha, double _aT, double
      _aMaiusculo, double _bMaiusculo, double _aij, double _interecao,
      double _aMix, double _bMix, std::vector<double> _fugacidade);
567 ~CPengRobinson()=default;
568
569 //Metodos Get e Set
570 void AMaiusculo(double _aMaiusculo){
571         aMaiusculo=_aMaiusculo;}
572
573 double AMaiusculo(){
574         return aMaiusculo;}
575
576 void BMaiusculo(double _bMaiusculo){
577         bMaiusculo=_bMaiusculo;}
578
579 double BMaiusculo(){
580         return bMaiusculo;}
581
582 void AMix(double _aMix){
583         aMix=_aMix;}
584
585 double AMix(){
586         return aMix;}
587
588 void BMix(double _bMix){
589         bMix=_bMix;}
590
591 double BMix(){
592         return bMix;}
593
594 void Fugacidade(std::vector<double> _fugacidade){
595         fugacidade = _fugacidade;}
596
597 std::vector<double> Fugacidade(){
```

```
598         return fugacidade;}
599
600 void Vsubstancia(std::vector<CSubstancia> _vsubstancia){
601         vsubstancia = _vsubstancia;}
602
603 std::vector<CSubstancia> VSubstancia(){
604         return vsubstancia;}
605
606 //Metodos
607 void ObterDadosSubstancia(int quantSubs);
608 double CalcularParametros(double T, double P);
609 double CalcularParametros(double T, double P, std::vector<double> x,
      double interacao);
610 std::vector<double> CalcularFugacidade(std::vector<long double> z,
      double aM, double bM);
611 std::vector<double> CalcularFugacidade(std::vector<long double> z,
      double aM, double bM, double bMix, double aMix, std::vector<double> x
      );
612 void CalcularPropriedadesTermodinamicas(double T, double P, std::vector<
      long double> z, std::vector<double> fugacidade, std::vector<
      CSubstancia> vsubs);
613 void CalcularPropriedadesTermodinamicas(double T, double P, std::vector<
      long double> z, std::vector<double> fugacidade, std::vector<
      CSubstancia> vsubs, std::vector<double> x);
614 void SalvarPropriedadesTermodinamicasDisco(double T, double P, std::
      vector<long double> z, std::vector<double> fugacidade, std::vector<
      CSubstancia> vsubs, std::string nome);
615 void SalvarPropriedadesTermodinamicasDisco(double T, double P, std::
      vector<long double> z, std::vector<double> fugacidade, std::vector<
      CSubstancia> vsubs, std::vector<double> x, std::string nome);
616 };
617 #endif
```

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

Listing 6.8: Arquivo de implementação da classe CPengRobinson.

```
618 #include "CPengRobinson.h"
619 #include <iostream>
620 #include <cmath>
621 #include <vector>
622 #include <string>
623
624 using namespace std;
625
626 //Construtores
627 CPengRobinson::CPengRobinson(CPengRobinson& obj){
628         vsubstancia=obj.vsubstancia;
629         tR=obj.tR;
630         a=obj.a;
```

```
631         b=obj.b;
632         kappa=obj.kappa;
633         alpha=obj.alpha;
634         aT=obj.aT;
635         aMaiusculo=obj.aMaiusculo;
636         bMaiusculo=obj.bMaiusculo;
637         aij=obj.aij;
638         aMix=obj.aMix;
639         bMix=obj.bMix;
640         fugacidade=obj.fugacidade;
641 }
642 CPengRobinson::CPengRobinson(vector<CSubstancia> _vsubstancia, vector<
      double> _tR, vector<double> _a, vector<double> _b, vector<double>
      _kappa, vector<double> _alpha, double _aT, double _aMaiusculo, double
       _bMaiusculo, double _aij, double _interecao, double _aMix, double
      _bMix, vector<double> _fugacidade){
643         vsubstancia=_vsubstancia;
644         tR=_tR;
645         a=_a;
646         b=_b;
647         kappa=_kappa;
648         alpha=_alpha;
649         aT=_aT;
650         aMaiusculo=_aMaiusculo;
651         bMaiusculo=_bMaiusculo;
652         aij=_aij;
653         aMix=_aMix;
654         bMix=_bMix;
655         fugacidade=_fugacidade;
656 }
657
658 //Calcula parametros PR para 1 substancia
659 double CPengRobinson::CalcularParametros(double T, double P){
660                 a.push_back(0.457235 * pow(R,2) * pow(vsubstancia[0].TC
                        (),2) / vsubstancia[0].PC());
661                 b.push_back(0.0777961 * R * vsubstancia[0].TC()/
                        vsubstancia[0].PC());
662                 kappa.push_back(0.37464 + 1.54226 * vsubstancia[0].Omega
                        () - 0.26992 * pow(vsubstancia[0].Omega(),2));
663                 tR.push_back(T/vsubstancia[0].TC());
664                 alpha.push_back(pow(1 + kappa[0] * (1 - sqrt(tR[0])),2))
                        ;
665                 aT=a[0]*alpha[0];
666                 bT=b[0];
667                 aMaiusculo=aT*P/pow(R,2)/pow(T,2);
668                 bMaiusculo=bT*P/R/T;
669                 return aMaiusculo;
670                 return bMaiusculo;
```

```
671 }
672 //Calcula parametros PR para 2 substancias
673 double CPengRobinson::CalcularParametros(double T, double P, vector<
      double> x, double interacao){
674         for(int i=0;i<2;i++){
675                 a.push_back(0.457235 * pow(R,2) * pow(vsubstancia[i].TC
                        (),2) / vsubstancia[i].PC());
676                 b.push_back(0.0777961 * R * vsubstancia[i].TC()/
                        vsubstancia[i].PC());
677                 kappa.push_back(0.37464 + 1.54226 * vsubstancia[i].Omega
                        () - 0.26992 * pow(vsubstancia[i].Omega(),2));
678                 tR.push_back(T/vsubstancia[i].TC());
679                 alpha.push_back(a[i]*pow(1 + kappa[i] * (1 - sqrt(tR[i])
                        ),2));
680         }
681         aij=(1.0 - interacao) * sqrt(a[0] * a[1]);
682         aMix=alpha[0] * pow(x[0],2) + alpha[1] * pow(x[1],2) + 2.0 * x
              [0] * x[1] * aij;
683         bMix= x[0] * b[0] + x[1] * b[1];
684         aT=aMix;
685         bT=bMix;
686         aMaiusculo=aT*P/pow(R,2)/pow(T,2);
687         bMaiusculo=bT*P/R/T;
688         return aMaiusculo;
689         return bMaiusculo;
690         return aMix;
691         return bMix;
692 }
693
694 void CPengRobinson::ObterDadosSubstancia(int quantSubs){
695         CEquacaoEstado::ObterDadosSubstancia(quantSubs);
696 }
697
698 //Calcula fugacidade PR para 1 substancia
699 vector<double> CPengRobinson::CalcularFugacidade(vector<long double> z,
      double aM, double bM){
700         for(int i=0;i<z.size();i++)
701                 fugacidade.push_back(exp(z[i] - 1 - log(z[i] - bM) - aM
                        / sqrt(8.) / bM * log((z[i] + (1 + sqrt(2)) * bM) / (
                        z[i] + (1. - sqrt(2.)) * bM))));
702         return fugacidade;
703 }
704
705 //Calcula fugacidade PR para 2 substancias
706 vector<double> CPengRobinson::CalcularFugacidade(vector<long double> z,
      double aM, double bM, double bMix, double aMix, vector<double> x){
707         for (int i=0;i<2;i++)
708                 fugacidade.push_back(exp(-log(z[i] - bM) + (z[i] - 1.0)
```

```
                    * b[0] / bMix - aM / sqrt(8.) / bM * (2.0 / aMix * (x
                    [0] * a[0] + x[1] * aij) - b[0] / bMix) * log((z[i] +
                     (1.0 + sqrt(2)) * bM) / (z[i] + (1.0 - sqrt(2.)) *
                    bM))));
709         for (int i=0;i<2;i++)
710                 fugacidade.push_back(exp(-log(z[i] - bM) + (z[i] - 1.0)
                    * b[1] / bMix - aM / sqrt(8.) / bM * (2.0 / aMix * (x
                    [1] * a[1] + x[0] * aij) - b[1] / bMix) * log((z[i] +
                     (1.0 + sqrt(2)) * bM) / (z[i] + (1.0 - sqrt(2.)) *
                    bM))));
711         return fugacidade;
712 }
713
714 void CPengRobinson::CalcularPropriedadesTermodinamicas(double T, double
    P, vector<long double> z, vector<double> fugacidade, vector<
    CSubstancia> vsubs){
715         CEquacaoEstado::CalcularPropriedadesTermodinamicas(T,P, z,
               fugacidade, vsubs);
716 }
717
718 void CPengRobinson::CalcularPropriedadesTermodinamicas(double T, double
    P, vector<long double> z, vector<double> fugacidade, vector<
    CSubstancia> vsubs, vector<double> x){
719         CEquacaoEstado::CalcularPropriedadesTermodinamicas(T, P, z,
               fugacidade, vsubs, x);
720 }
721
722 void CPengRobinson::SalvarPropriedadesTermodinamicasDisco(double T,
    double P, vector<long double> z, vector<double> fugacidade, vector<
    CSubstancia> vsubs, string nome){
723         CEquacaoEstado::SalvarPropriedadesTermodinamicasDisco(T,P, z,
               fugacidade, vsubs, nome);
724 }
725
726 void CPengRobinson::SalvarPropriedadesTermodinamicasDisco(double T,
    double P, vector<long double> z, vector<double> fugacidade, vector<
    CSubstancia> vsubs, vector<double> x, string nome){
727         CEquacaoEstado::SalvarPropriedadesTermodinamicasDisco(T, P, z,
               fugacidade, vsubs, x, nome);
728 }
```

Apresenta-se na listagem 6.9 o arquivo com código da classe `CSoaveRedlichKwong`.

Listing 6.9: Arquivo de cabeçalho da classe CSoaveRedlichKwong.

```
729 #ifndef CSoaveRedlichKwong_h
730 #define CSoaveRedlichKwong_h
731 #include "CEquacaoEstado.h"
732 #include <iostream>
733 #include <fstream>
```

```
734 #include <cmath>
735 #include <vector>
736 #include <string>
737
738 class CSoaveRedlichKwong: public CEquacaoEstado{
739
740 public:
741
742 //Atributos
743 CSoaveRedlichKwong(): CEquacaoEstado() {};
744 CSoaveRedlichKwong(CSoaveRedlichKwong& obj);
745 CSoaveRedlichKwong(std::vector<CSubstancia> _vsubstancia, std::vector<
      double> _tR, std::vector<double> _a, std::vector<double> _b, std::
      vector<double> _kappa, std::vector<double> _alpha, double _aT, double
       _aMaiusculo, double _bMaiusculo, double _aij, double _interecao,
      double _aMix, double _bMix, std::vector<double> _fugacidade);
746 ~CSoaveRedlichKwong()=default;
747
748 //Metodos Get e Set
749 void AMaiusculo(double _aMaiusculo){
750         aMaiusculo=_aMaiusculo;}
751
752 double AMaiusculo(){
753         return aMaiusculo;}
754
755 void BMaiusculo(double _bMaiusculo){
756         bMaiusculo=_bMaiusculo;}
757
758 double BMaiusculo(){
759         return bMaiusculo;}
760
761 void AMix(double _aMix){
762         aMix=_aMix;}
763
764 double AMix(){
765         return aMix;}
766
767 void BMix(double _bMix){
768         bMix=_bMix;}
769
770 double BMix(){
771         return bMix;}
772
773 void Fugacidade(std::vector<double> _fugacidade){
774         fugacidade = _fugacidade;}
775
776 std::vector<double> Fugacidade(){
777         return fugacidade;}
```

```
778
779 void Vsubstancia(std::vector<CSubstancia> _vsubstancia){
780         vsubstancia = _vsubstancia;}
781
782 std::vector<CSubstancia> VSubstancia(){
783         return vsubstancia;}
784
785 //Metodos
786 void ObterDadosSubstancia(int quantSubs);
787 double CalcularParametros(double T, double P);
788 double CalcularParametros(double T, double P, std::vector<double> x,
      double interacao);
789 std::vector<double> CalcularFugacidade(std::vector<long double> z,
      double aM, double bM);
790 std::vector<double> CalcularFugacidade(std::vector<long double> z,
      double aM, double bM, double bMix, double aMix, std::vector<double> x
      );
791 void CalcularPropriedadesTermodinamicas(double T, double P, std::vector<
      long double> z, std::vector<double> fugacidade, std::vector<
      CSubstancia> vsubs);
792 void CalcularPropriedadesTermodinamicas(double T, double P, std::vector<
      long double> z, std::vector<double> fugacidade, std::vector<
      CSubstancia> vsubs, std::vector<double> x);
793 void SalvarPropriedadesTermodinamicasDisco(double T, double P, std::
      vector<long double> z, std::vector<double> fugacidade, std::vector<
      CSubstancia> vsubs, std::string nome);
794 void SalvarPropriedadesTermodinamicasDisco(double T, double P, std::
      vector<long double> z, std::vector<double> fugacidade, std::vector<
      CSubstancia> vsubs, std::vector<double> x, std::string nome);
795 };
796 #endif
```

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

Listing 6.10: Arquivo de implementação da classe CSoaveRedlichKwong.

```
797 #include "CSoaveRedlichKwong.h"
798 #include <iostream>
799 #include <cmath>
800 #include <vector>
801 #include <string>
802
803 using namespace std;
804
805 //Construtores
806 CSoaveRedlichKwong::CSoaveRedlichKwong(CSoaveRedlichKwong& obj){
807         vsubstancia=obj.vsubstancia;
808         tR=obj.tR;
809         a=obj.a;
810         b=obj.b;
```

```
811         kappa=obj.kappa;
812         alpha=obj.alpha;
813         aT=obj.aT;
814         aMaiusculo=obj.aMaiusculo;
815         bMaiusculo=obj.bMaiusculo;
816         aij=obj.aij;
817         aMix=obj.aMix;
818         bMix=obj.bMix;
819         fugacidade=obj.fugacidade;
820 }
821 CSoaveRedlichKwong::CSoaveRedlichKwong(vector<CSubstancia> _vsubstancia,
      vector<double> _tR, vector<double> _a, vector<double> _b, vector<
     double> _kappa, vector<double> _alpha, double _aT, double _aMaiusculo
     , double _bMaiusculo, double _aij, double _interecao, double _aMix,
     double _bMix, vector<double> _fugacidade){
822         vsubstancia=_vsubstancia;
823         tR=_tR;
824         a=_a;
825         b=_b;
826         kappa=_kappa;
827         alpha=_alpha;
828         aT=_aT;
829         aMaiusculo=_aMaiusculo;
830         bMaiusculo=_bMaiusculo;
831         aij=_aij;
832         aMix=_aMix;
833         bMix=_bMix;
834         fugacidade=_fugacidade;
835 }
836
837 //Calcula parametros SRK para 1 substancia
838 double CSoaveRedlichKwong::CalcularParametros(double T, double P){
839                 a.push_back(0.42748 * pow(R,2) * pow(vsubstancia[0].TC()
                    ,2) / vsubstancia[0].PC());
840                 b.push_back(0.08664 * R * vsubstancia[0].TC()/
                    vsubstancia[0].PC());
841                 kappa.push_back(0.48508 + 1.5517 * vsubstancia[0].Omega
                    () - 0.15613 * pow(vsubstancia[0].Omega(),2));
842                 tR.push_back(T/vsubstancia[0].TC());
843                 alpha.push_back(pow(1 + kappa[0] * (1 - sqrt(tR[0])),2))
                    ;
844                 aT=a[0]*alpha[0];
845                 bT=b[0];
846                 aMaiusculo=aT*P/pow(R,2)/pow(T,2);
847                 bMaiusculo=bT*P/R/T;
848                 return aMaiusculo;
849                 return bMaiusculo;
850 }
```

```
851
852 //Calcula parametros SRK para 2 substancias
853 double CSoaveRedlichKwong::CalcularParametros(double T, double P, vector
      <double> x, double interacao){
854         for(int i=0;i<2;i++){
855                 a.push_back(0.42748 * pow(R,2) * pow(vsubstancia[i].TC()
                        ,2) / vsubstancia[i].PC());
856                 b.push_back(0.08664 * R * vsubstancia[i].TC()/
                        vsubstancia[i].PC());
857                 kappa.push_back(0.48508 + 1.5517 * vsubstancia[i].Omega
                        () - 0.15613 * pow(vsubstancia[i].Omega(),2));
858                 tR.push_back(T/vsubstancia[i].TC());
859                 alpha.push_back(a[i]*pow(1 + kappa[i] * (1 - sqrt(tR[i])
                        ),2));
860         }
861         aij=(1.0 - interacao) * sqrt(a[0] * a[1]);
862         aMix=alpha[0] * pow(x[0],2) + alpha[1] * pow(x[1],2) + 2.0 * x
               [0] * x[1] * aij;
863         bMix= x[0] * b[0] + x[1] * b[1];
864         aT=aMix;
865         bT=bMix;
866         aMaiusculo=aT*P/pow(R,2)/pow(T,2);
867         bMaiusculo=bT*P/R/T;
868         return aMaiusculo;
869         return bMaiusculo;
870         return aMix;
871         return bMix;
872 }
873
874 void CSoaveRedlichKwong::ObterDadosSubstancia(int quantSubs){
875         CEquacaoEstado::ObterDadosSubstancia(quantSubs);
876 }
877
878 //Calcula fugacidade SRK para 1 substancia
879 vector<double> CSoaveRedlichKwong::CalcularFugacidade(vector<long double
      > z, double aM, double bM){
880         for(int i=0;i<z.size();i++)
881                 fugacidade.push_back(exp(z[i] - 1 - log(z[i] - bM) - aM
                        / sqrt(8.) / bM * log((z[i] + (1 + sqrt(2.)) * bM) /
                        (z[i] + (1 - sqrt(2.)) * bM))));
882         return fugacidade;
883 }
884
885 //Calcula fugacidade SRK para 2 substancias
886 vector<double> CSoaveRedlichKwong::CalcularFugacidade(vector<long double
      > z, double aM, double bM, double bMix, double aMix, vector<double> x
      ){
887         for (int i=0;i<2;i++)
```

```
888                 fugacidade.push_back(exp(-log(z[i] - bM) + (z[i] - 1.0)
                        * b[0] / bMix - aM  / bM * (2.0 / aMix * (x[0] * a[0]
                         + x[1] * aij) - b[0] / bMix) * log(1+bM/z[i])));
889         for (int i=0;i<2;i++)
890                 fugacidade.push_back(exp(-log(z[i] - bM) + (z[i] - 1.0)
                        * b[1] / bMix - aM  / bM * (2.0 / aMix * (x[1] * a[1]
                         + x[0] * aij) - b[1] / bMix) * log(1+bM/z[i])));
891         return fugacidade;
892 }
893
894 void CSoaveRedlichKwong::CalcularPropriedadesTermodinamicas(double T,
      double P, vector<long double> z, vector<double> fugacidade, vector<
      CSubstancia> vsubs){
895         CEquacaoEstado::CalcularPropriedadesTermodinamicas(T,P, z,
               fugacidade, vsubs);
896 }
897
898 void CSoaveRedlichKwong::CalcularPropriedadesTermodinamicas(double T,
      double P, vector<long double> z, vector<double> fugacidade, vector<
      CSubstancia> vsubs, vector<double> x){
899         CEquacaoEstado::CalcularPropriedadesTermodinamicas(T, P, z,
               fugacidade, vsubs, x);
900 }
901
902 void CSoaveRedlichKwong::SalvarPropriedadesTermodinamicasDisco(double T,
       double P, vector<long double> z, vector<double> fugacidade, vector<
      CSubstancia> vsubs, string nome){
903         CEquacaoEstado::SalvarPropriedadesTermodinamicasDisco(T,P, z,
               fugacidade, vsubs, nome);
904 }
905
906 void CSoaveRedlichKwong::SalvarPropriedadesTermodinamicasDisco(double T,
       double P, vector<long double> z, vector<double> fugacidade, vector<
      CSubstancia> vsubs, vector<double> x, string nome){
907         CEquacaoEstado::SalvarPropriedadesTermodinamicasDisco(T, P, z,
               fugacidade, vsubs, x, nome);
908 }
```

Apresenta-se na listagem 6.11 o programa que usa as classes anteriores.

Listing 6.11: Arquivo de implementação da função `main()`.

```
909 #include "CSubstancia.h"
910 #include "CEquacaoCubica.h"
911 #include "CEquacaoEstado.h"
912 #include "CPengRobinson.h"
913 #include "CSoaveRedlichKwong.h"
914 #include "CGnuplot.h"
915 #include "CSimulacao.h"
916 #include <cmath>
```

```
917 #include <locale.h>
918 #include <vector>
919 #include <string>
920 #include <iostream>
921 #include <algorithm>
922 #include <fstream>
923 #include <sstream>
924
925 using namespace std;
926
927 int main(){
928 CSimulacao sim;
929 sim.Simular();
930 system("pause");
931 return 0;
932 }
```

# 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 Descrição

No início apresente texto explicativo do teste:

Foi testada a capacidade do programa de gerar resultados semelhantes para os dois métodos (Peng-Robinson e Soave-Redlich-Kwong). Para isso, foram simuladas condições físicas idênticas para cada um, e a seguir os resultados foram comparados. Essas condições físicas foram testadas para todos os casos possíveis: substância simples com uma fase, substância simples com duas fases, mistura de duas substâncias com uma fase e mistura de duas substâncias com duas fases. Então há 8 (oito) casos possíveis que serão mostrados a seguir, já que para cada método há a opção de uma ou duas substâncias, e para cada uma dessas, a opção de uma ou duas fases.

Em todos os testes, o $Z$ (fator de compressibilidade) obtido foi maior que 0 (zero), podendo chegar até valores ligeiramente acima de 1 (um), o que é exatamente o esperado para substâncias reais. Como as demais propriedades dependem direta ou indiretamente dele, também tiveram valores condizentes com o esperado.

A Figura 7.1 mostra um exemplo dos resultados que o programa mostra após o usuário inserir as condições da simulação.

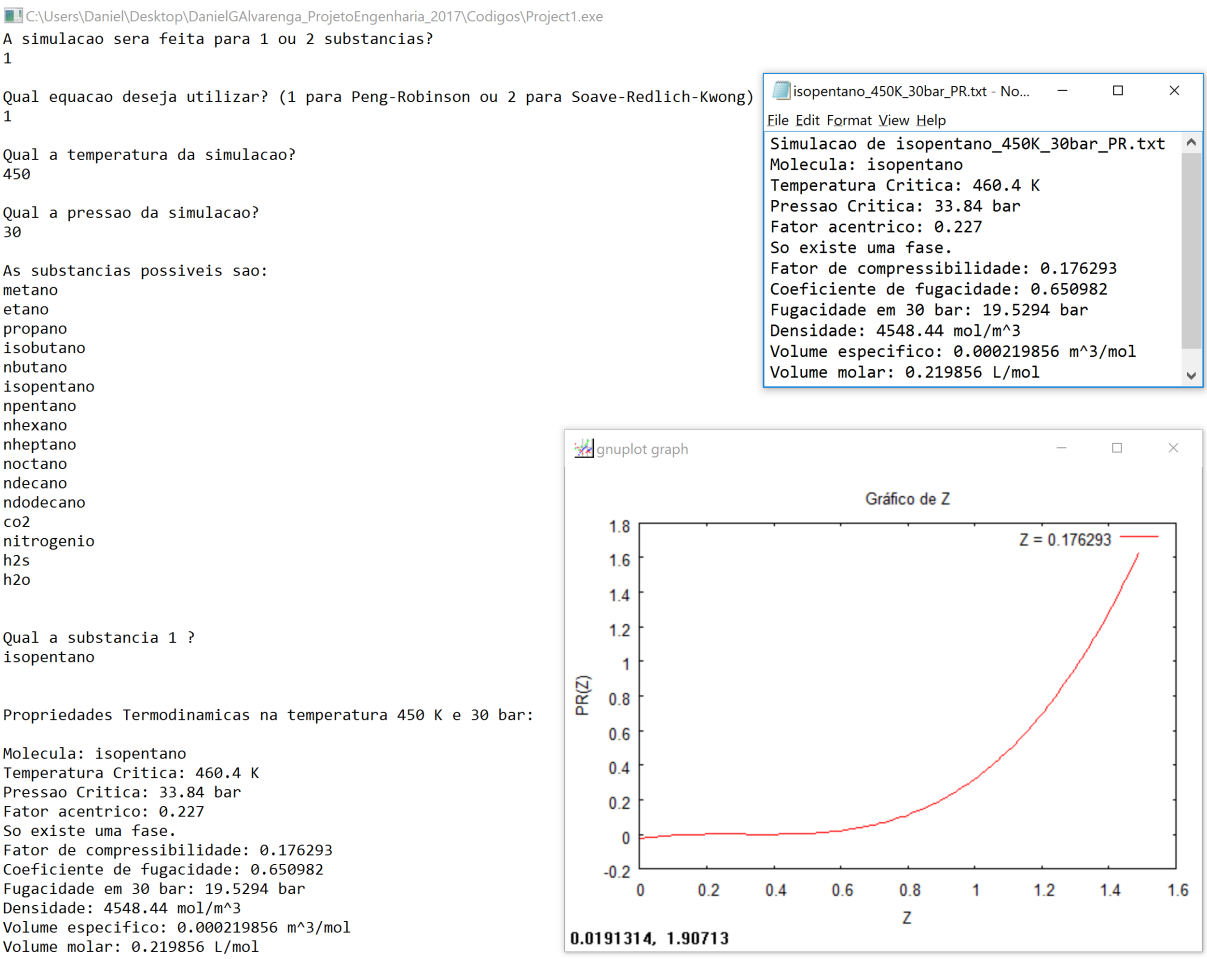
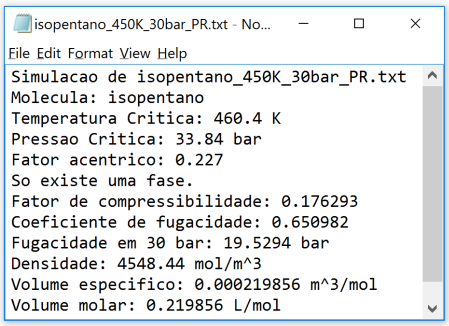



Figura 7.1: Captura de tela do programa e os resultados

## 7.2 Teste 2: Descrição

A Figura 7.2 mostra o programa rodando a simulação para método Peng-Robinson de uma substância e obtendo uma fase como resposta.

```
C:\Users\Daniel\Desktop\DanielGAlvarenga_ProjetoEngenharia_2017\Codigos\Project1.exe
A simulacao sera feita para 1 ou 2 substancias?
1

Qual equacao deseja utilizar? (1 para Peng-Robinson ou 2 para Soave-Redlich-Kwong)
1

Qual a temperatura da simulacao?
450

Qual a pressao da simulacao?
30

As substancias possiveis sao:
metano
etano
propano
isobutano
nbutano
isopentano
npentano
nhexano
nheptano
noctano
ndecano
ndodecano
co2
nitrogenio
h2s
h2o


Qual a substancia 1 ?
isopentano


Propriedades Termodinamicas na temperatura 450 K e 30 bar:

Molecula: isopentano
Temperatura Critica: 460.4 K
Pressao Critica: 33.84 bar
Fator acentrico: 0.227
So existe uma fase.
Fator de compressibilidade: 0.176293
Coeficiente de fugacidade: 0.650982
Fugacidade em 30 bar: 19.5294 bar
Densidade: 4548.44 mol/m^3
Volume especifico: 0.000219856 m^3/mol
Volume molar: 0.219856 L/mol
```



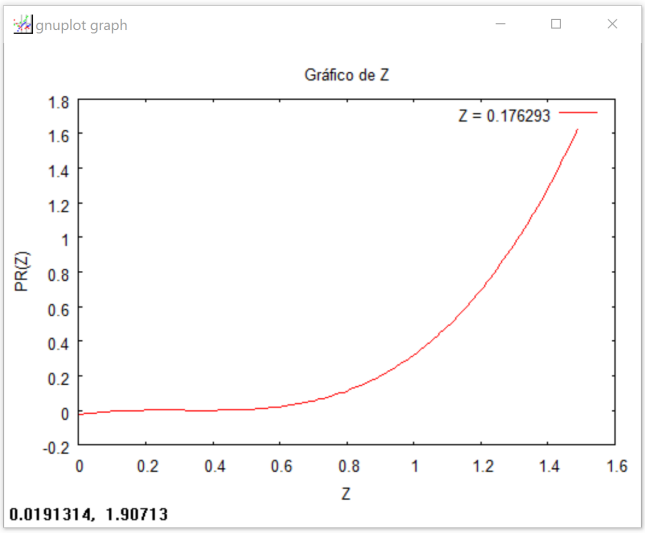
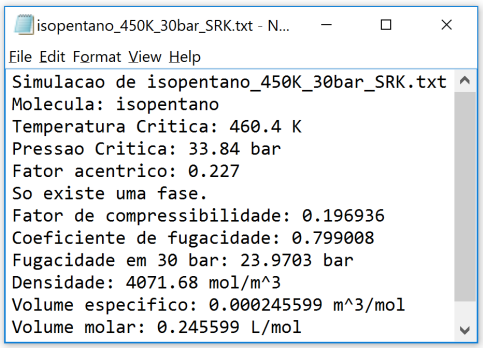




Figura 7.2: Captura de tela com simulação de isopentano a 450 K e 30 bar pela equação PR

A Figura 7.3 mostra o programa rodando a simulação para método Soave-Redlich-Kwong de uma substância e obtendo uma fase como resposta.

```
C:\Users\Daniel\Desktop\DanielGAlvarenga_ProjetoEngenharia_2017\Codigos\Project1.exe
A simulacao sera feita para 1 ou 2 substancias?
1

Qual equacao deseja utilizar? (1 para Peng-Robinson ou 2 para Soave-Redlich-Kwong)
2

Qual a temperatura da simulacao?
450

Qual a pressao da simulacao?
30

As substancias possiveis sao:
metano
etano
propano
isobutano
nbutano
isopentano
npentano
nhexano
nheptano
noctano
ndecano
ndodecano
co2
nitrogenio
h2s
h2o


Qual a substancia 1 ?
isopentano


Propriedades Termodinamicas na temperatura 450 K e 30 bar:

Molecula: isopentano
Temperatura Critica: 460.4 K
Pressao Critica: 33.84 bar
Fator acentrico: 0.227
So existe uma fase.
Fator de compressibilidade: 0.196936
Coeficiente de fugacidade: 0.799008
Fugacidade em 30 bar: 23.9703 bar
Densidade: 4071.68 mol/m^3
Volume especifico: 0.000245599 m^3/mol
Volume molar: 0.245599 L/mol
```



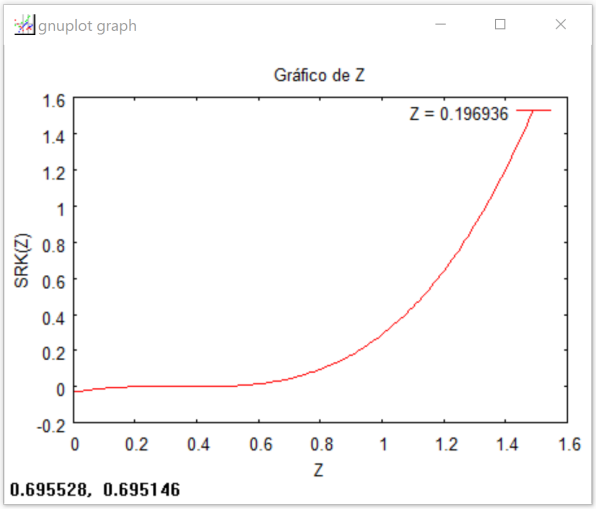




Figura 7.3: Captura de tela com simulação de isopentano a 450 K e 30 bar pela equação SRK

A Figura 7.4 mostra o programa rodando a simulação para método Peng-Robinson de uma substância e obtendo duas fases como resposta.

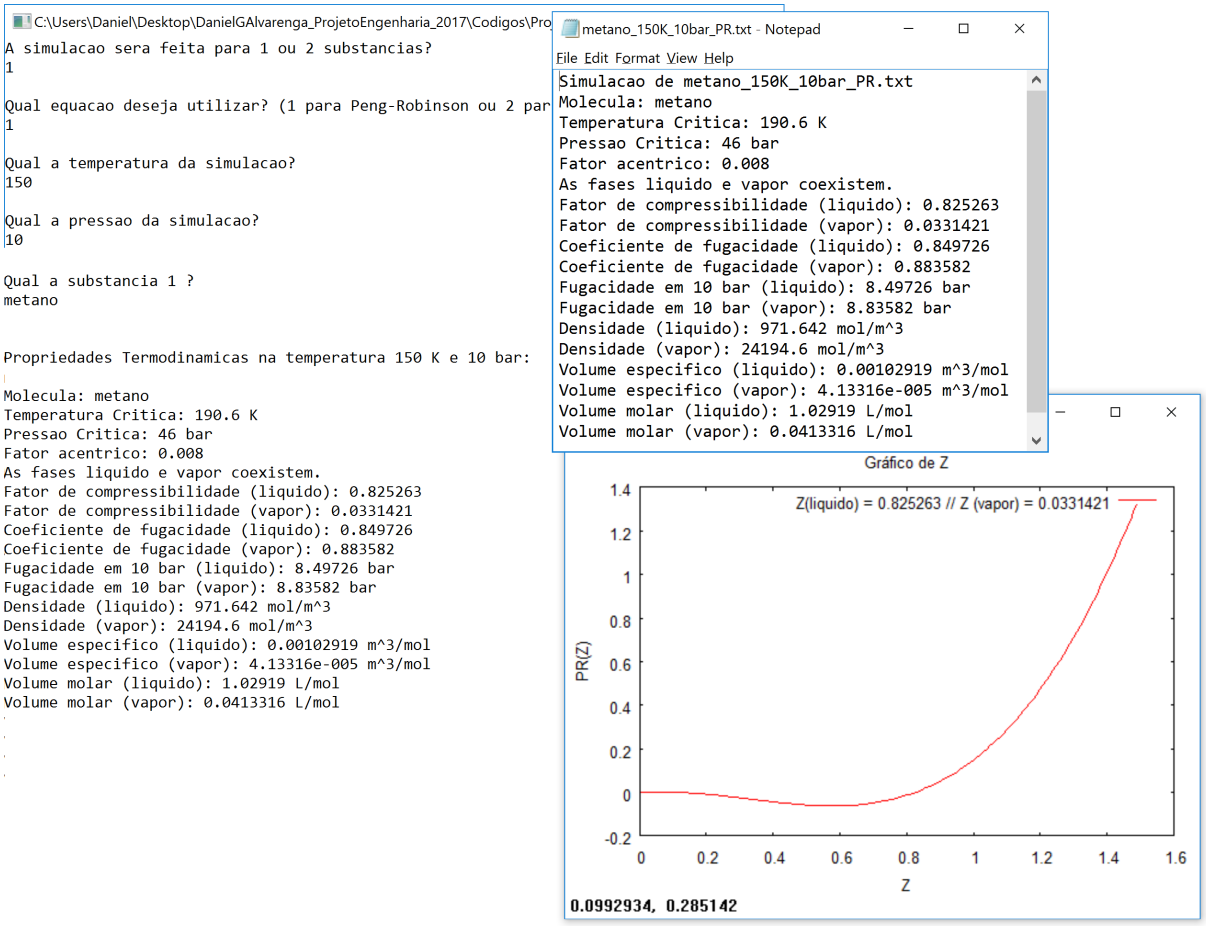



Figura 7.4: Captura de tela com simulação de metano a 150 K e 10 bar pela equação PR

A Figura 7.5 mostra o programa rodando a simulação para método Soave-Redlich-Kwong de uma substância e obtendo duas fases como resposta.

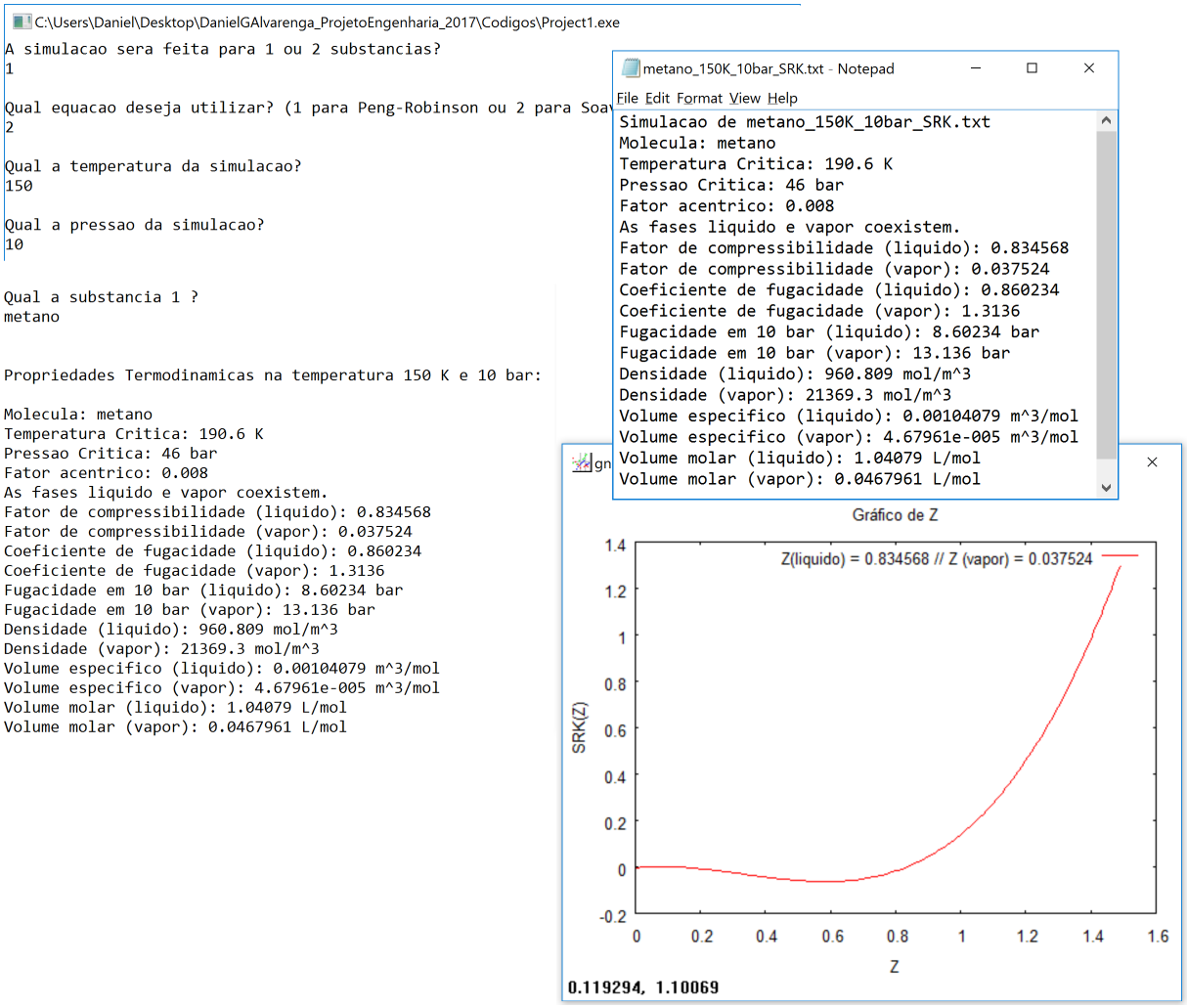



Figura 7.5: Captura de tela com simulação de metano a 150 K e 10 bar pela equação SRK

A Figura 7.6 mostra o programa rodando a simulação para método Peng-Robinson de duas substâncias e obtendo uma fase como resposta.

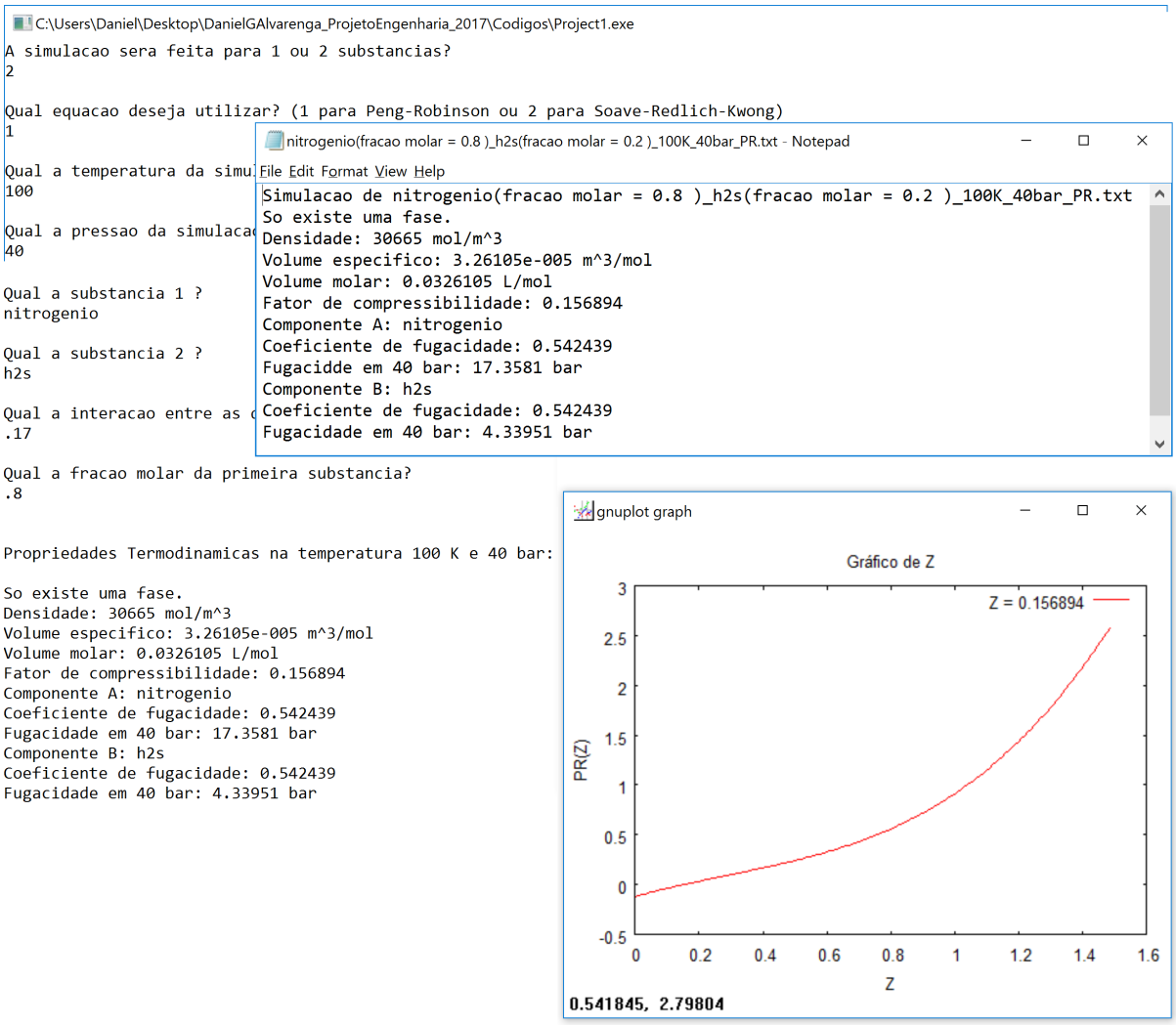



Figura 7.6: Captura de tela com simulação de nitrogênio e $H_2S$ a 100 K e 40 bar pela equação PR

A Figura 7.7 mostra o programa rodando a simulação para método Soave-Redlich-Kwong de duas substâncias e obtendo uma fase como resposta.

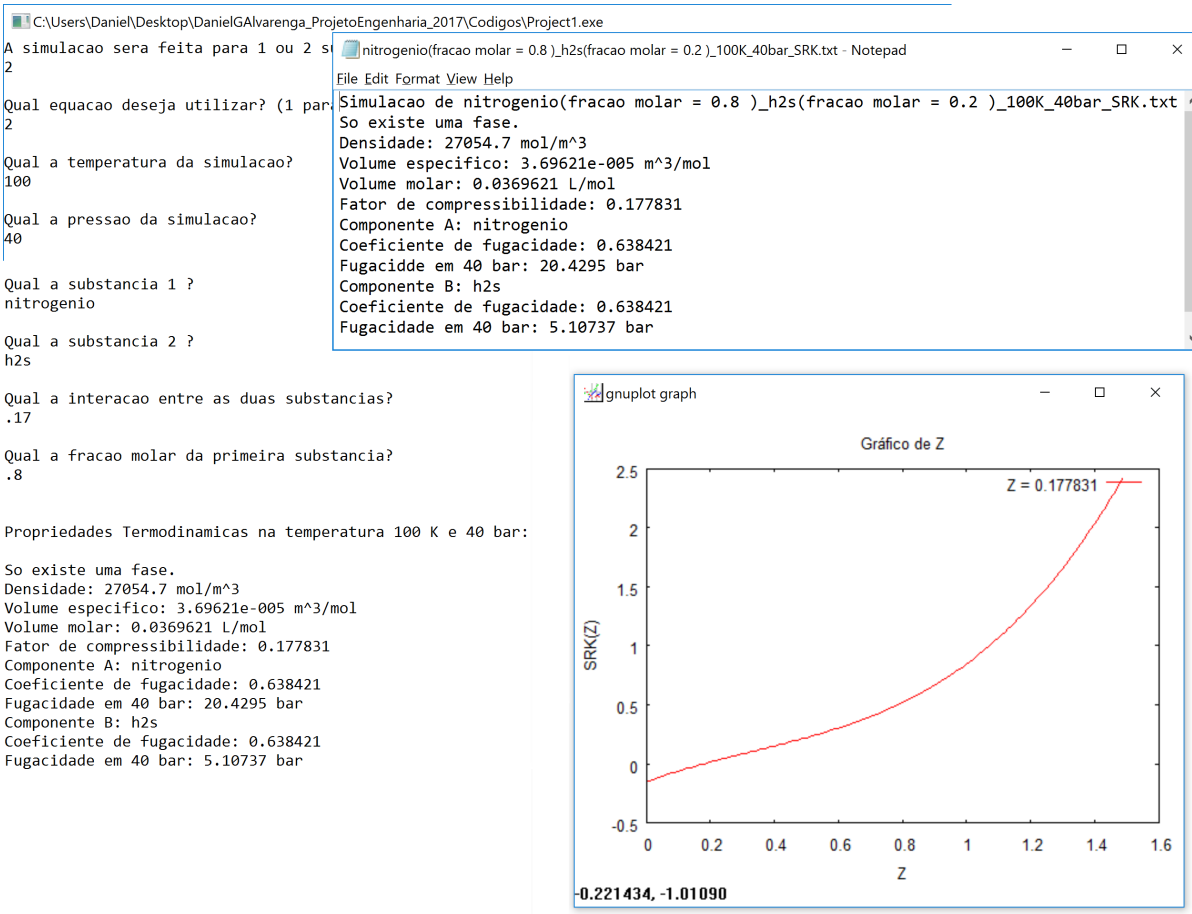



Figura 7.7: Captura de tela com simulação de nitrogênio e $H_2S$ a 100 K e 40 bar pela equação SRK

A Figura 7.8 mostra o programa rodando a simulação para método Peng-Robinson de duas substâncias e obtendo duas fases como resposta.

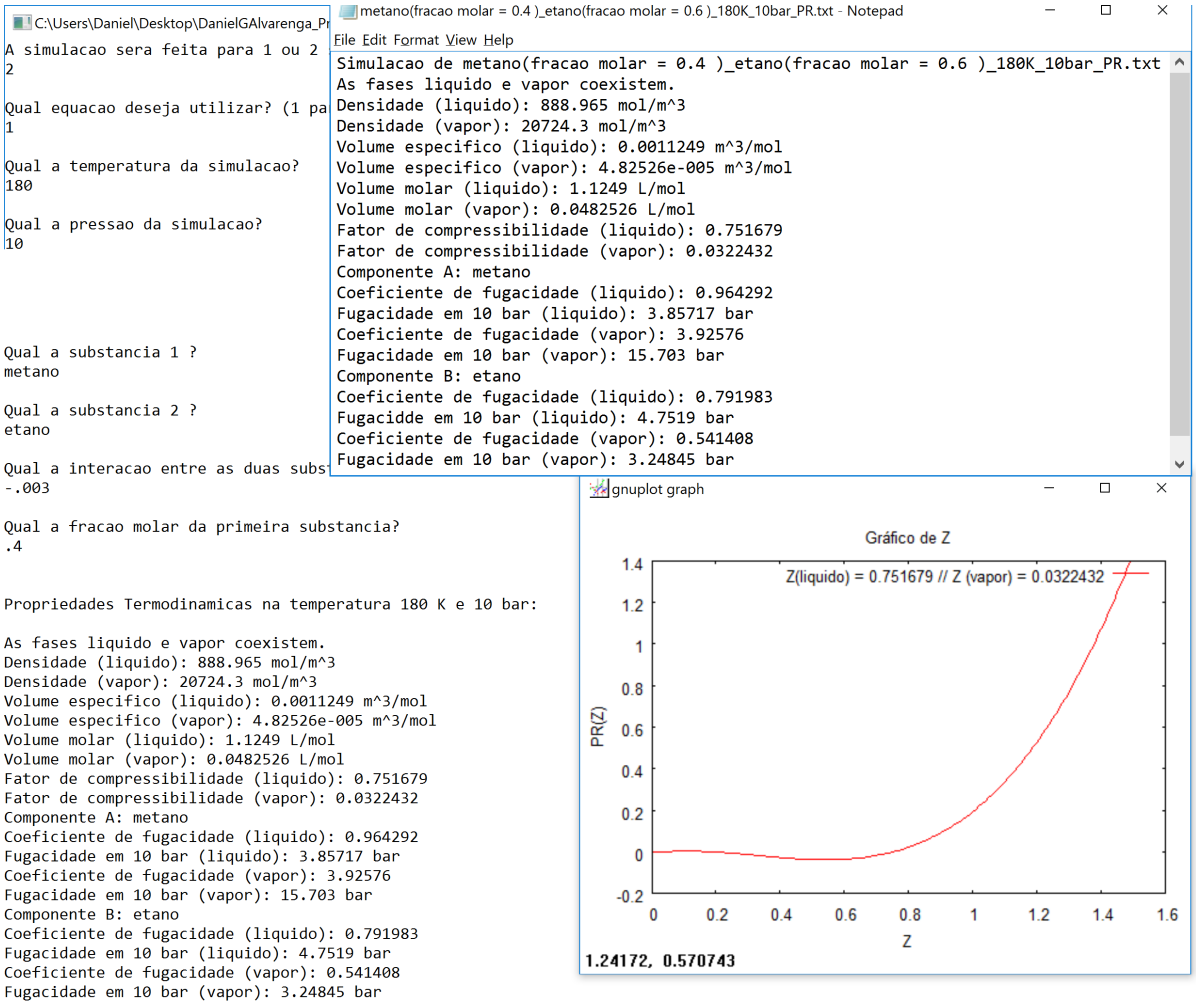



Figura 7.8: Captura de tela com simulação de metano e etano a 180 K e 10 bar pela equação PR

A Figura 7.9 mostra o programa rodando a simulação para método Soave-Redlich-Kwong de duas substâncias e obtendo duas fases como resposta.

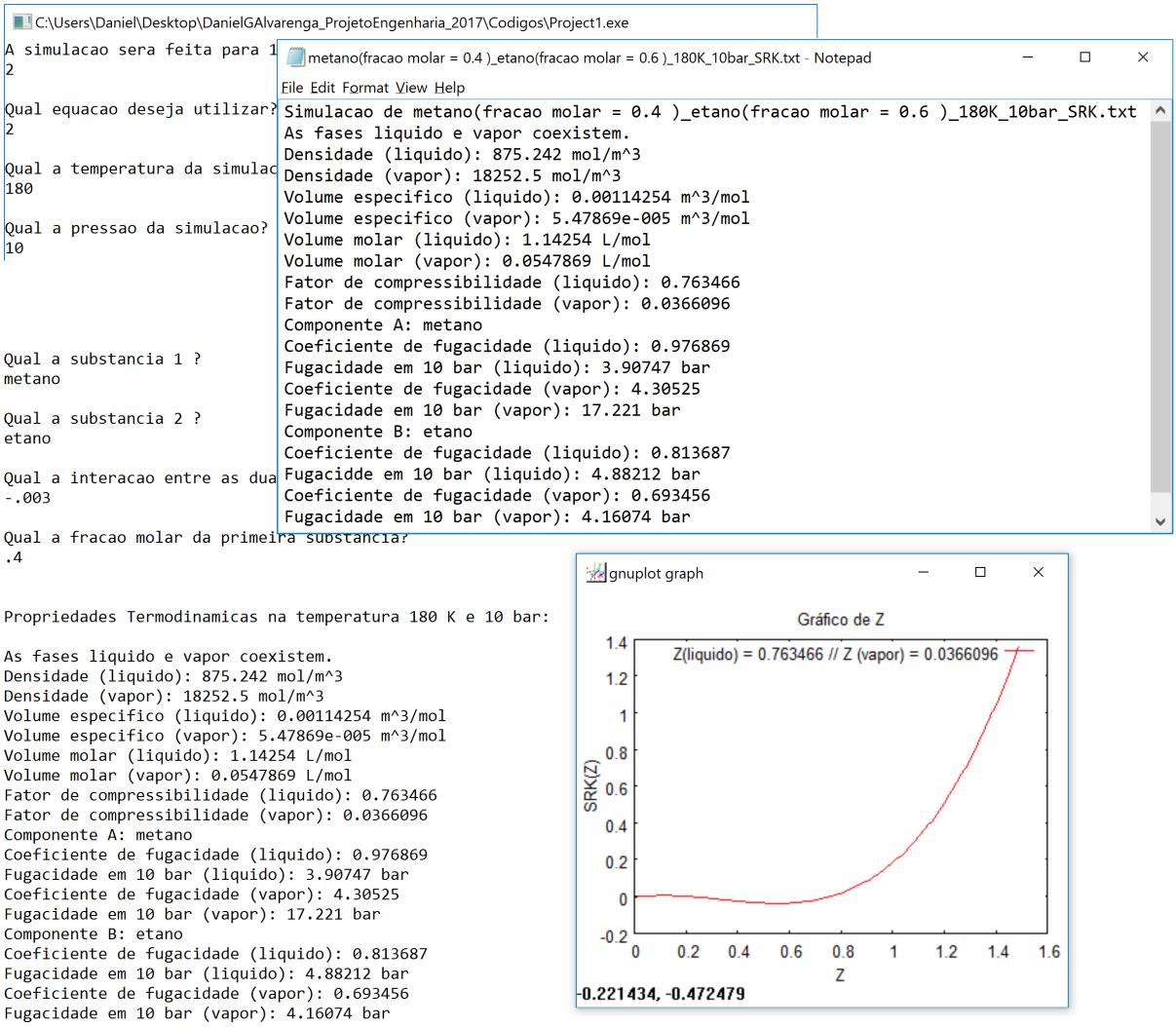



Figura 7.9: Captura de tela com simulação de metano e etano a 180 K e 10 bar pela equação SRK

# Capítulo 8

# Documentação

Apresenta-se neste capítulo a documentação de uso do "Simulador de propriedades termodinâmicas de substâncias simples e compostas". Esta documentação tem o formato de uma apostila que explica passo a passo como instalar e 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 Dependências

Para instalar o *software*, basta salvar os arquivos com extensão .h e .cpp de cada classe, o arquivo Main.cpp e o arquivo Substancias.txt na pasta. A seguir, o programa precisa ser compilado usando um compilador compatível com C++ 11/14.

### 8.1.2 Como rodar o software

Para rodar o *software* no windows é preciso executar o arquivo com extensão .exe. Para rodá-lo no GNU/Linux é preciso executar o arquivo a.out, gerado após a compilação.

Veja no Capítulo 7 - Teste, exemplos de uso do software.

## 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 Compilação

Para compilar o software é necessário atender as seguintes dependências:

- Instalar o compilador g++ da GNU disponível em http://gcc.gnu.org. Para instalar no GNU/Linux/Fedora use o comando `yum install gcc`.
- Biblioteca CGnuplot; os arquivos para acesso a biblioteca CGnuplot (CGnuplot.h e CGnuplot.cpp) devem estar no diretório com os códigos do software;
- O software `gnuplot`, disponível no endereço http://www.gnuplot.info/, deve estar instalado. É possível que haja necessidade de setar o caminho para execução do `gnuplot`. .

### 8.2.2 Como gerar a documentação usando doxygen

A documentação do código do software deve ser feita usando o padrão JAVADOC, conforme apresentada no Capítulo - Documentação, do livro texto da disciplina. Depois de documentar o código, use o software `doxygen` para gerar a documentação do desenvolvedor no formato html. O software `doxygen` lê os arquivos com os códigos (*.h e *.cpp) e gera uma documentação muito útil e de fácil navegação no formato html.

- Veja informações sobre uso do formato JAVADOC em:
  - http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html
- Veja informações sobre o software `doxygen` em
  - http://www.stack.nl/~dimitri/doxygen/

Passos para gerar a documentação usando o `doxygen`.

- Documente o código usando o formato JAVADOC. Um bom exemplo de código documentado é apresentado nos arquivos da biblioteca CGnuplot, abra os arquivos `CGnuplot.h` e `CGnuplot.cpp` no editor de texto e veja como o código foi documentado.
- Abra um terminal.
- Vá para o diretório onde esta o código.

```
cd /caminho/para/seu/codigo
```

- Peça para o `doxygen` gerar o arquivo de definições (arquivo que diz para o doxygem como deve ser a documentação).

```
dogygen -g
```

- Peça para o `doxygen` gerar a documentação.

```
doxygen
```

- Verifique a documentação gerada abrindo o arquivo html/index.html.

```
firefox html/index.html
```

ou

```
chrome html/index.html
```

# Índice Remissivo