

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

PROJETO ENGENHARIA  
DESENVOLVIMENTO DO SOFTWARE  
MODELOS DE AQUÍFEROS ANALÍTICOS  
TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA

Versão 1:  
RODRIGO SANTOS  
THALIA RODRIGUES  
Prof. André Duarte Bueno

MACAÉ - RJ  
Dezembro - 2021

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Escopo do problema . . . . .	1
1.2	Objetivos . . . . .	1
<b>2</b>	<b>Especificação</b>	<b>3</b>
2.1	O que é a especificação? . . . . .	3
2.2	Nome do sistema/produto . . . . .	3
2.3	Especificação . . . . .	4
2.3.1	Requisitos funcionais . . . . .	4
2.3.2	Requisitos não funcionais . . . . .	4
2.4	Casos de uso . . . . .	5
2.4.1	O que são os diagramas de casos de uso? . . . . .	5
2.4.2	Diagrama de caso de uso específico . . . . .	6
<b>3</b>	<b>Elaboração</b>	<b>8</b>
3.1	Análise de domínio . . . . .	8
3.2	Formulação teórica . . . . .	10
3.2.1	Equação da compressibilidade . . . . .	10
3.2.2	Modelo de Van Everdingen & Hurst (1949) . . . . .	11
3.2.3	Modelo de Carter-Tracy (1960) . . . . .	24
3.2.4	Modelo Aproximado de Fetkovich (1971) . . . . .	27
3.3	Identificação de pacotes – assuntos . . . . .	33
3.4	Diagrama de pacotes – assuntos . . . . .	33
<b>4</b>	<b>AOO – Análise Orientada a Objeto</b>	<b>35</b>
4.1	Diagramas de classes . . . . .	35
4.1.1	Dicionário de classes . . . . .	38
4.2	Diagrama de sequência – eventos e mensagens . . . . .	39
4.2.1	Diagrama de sequência geral . . . . .	39
4.3	Diagrama de comunicação – colaboração . . . . .	40
4.4	Diagrama de máquina de estado . . . . .	42
4.5	Diagrama de atividades . . . . .	42

<b>5</b>	<b>Projeto</b>	<b>45</b>
5.1	Projeto do sistema . . . . .	45
5.2	Projeto orientado a objeto – POO . . . . .	46
5.3	Diagrama de componentes . . . . .	47
5.4	Diagrama de implantação . . . . .	48
<b>6</b>	<b>Implementação</b>	<b>50</b>
6.1	Código fonte . . . . .	50
<b>7</b>	<b>Teste</b>	<b>242</b>
7.1	Teste 1: Escolha do Software . . . . .	242
7.2	Teste 2: <i>Software</i> - Van Everdingen & Hurst . . . . .	243
7.2.1	Passo inicial . . . . .	243
7.2.2	Cálculo do influxo de água adimensional para o aquífero radial in- finito x aquífero radial finito selado com seus respectivos <i>plots</i> . . .	243
7.2.3	Cálculo do influxo de água adimensional para o aquífero radial in- finito x aquífero radial finito com manutenção de pressão com seus respectivos <i>plots</i> . . . . .	244
7.2.4	Cálculo o influxo acumulado de água adimensional para o aquífero linear infinito, selado e com manutenção de pressão . . . . .	245
7.2.5	Salvando <i>plots</i> da simulação, destruindo arquivos temporários e saída do programa . . . . .	246
7.3	Teste 3: <i>Software</i> - Carter-Tracy e Fetkovich . . . . .	247
7.3.1	Passo inicial . . . . .	247
7.3.2	Carregando os dados do reservatório (dados iniciais) e dados da tabela (pressão e tempo) . . . . .	248
7.3.3	Cálculo do influxo acumulado de água do modelo de Carter-Tracy .	249
7.3.4	Cálculo do influxo acumulado de água do modelo de Fetkovich e o encerramento do <i>software</i> . . . . .	249
<b>8</b>	<b>Documentação</b>	<b>251</b>
8.1	Documentação do usuário . . . . .	251
8.1.1	Como instalar o software . . . . .	251
8.1.2	Como rodar o software . . . . .	251
8.2	Documentação para desenvolvedor . . . . .	251
8.2.1	Dependências . . . . .	252
8.2.2	Como gerar a documentação usando doxygen . . . . .	252

# Capítulo 1

## Introdução

No presente projeto de engenharia desenvolve-se o software Modelos de Aquíferos Analíticos, um software aplicado à engenharia de petróleo e que utiliza o paradigma da orientação a objetos.

O software é da área de Engenharia de Reservatórios e permite a análise de três modelos de aquíferos analíticos, como seus comportamentos, diferenças, semelhanças e influência sobre reservatórios de petróleo.

### 1.1 Escopo do problema

No ambiente da Engenharia de Reservatórios, o objeto de estudo é o próprio reservatório de óleo e gás. No entanto, para que esse estudo ocorra de forma eficiente, é necessário que se entenda as características (porosidade, permeabilidade, volume de reservatório, presença e propriedades de aquíferos) e o comportamento sob produção. O objetivo de estudo deste software é analisar aquíferos analíticos com modelos distintos e como sua presença provoca efeitos em reservatórios de óleo e gás.

Os aquíferos são facilmente encontrados em locais próximos aos reservatórios de hidrocarbonetos. Esses podem ser pequenos no tamanho e, por isso, terem seu efeito sobre o reservatório considerado desprezível; ou podem ser grandes, às vezes até maior do que os reservatórios de óleo e gás, e provocarem efeitos sobre eles. A queda de pressão devida à produção de fluidos causa a expansão do aquífero contíguo a zona de óleo, ocorre, então uma invasão da zona de óleo pelo volume de água excedente. Esse influxo de água, além de manter a pressão elevada na zona de óleo, desloca o fluido, facilitando a produção.

### 1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:

- Desenvolver um projeto de engenharia de software para analisar os diferentes modelos de aquíferos analíticos, com enfoque em aquíferos de grandes dimensões onde torna-se necessário um modelo matemático que inclua a dependência do tempo, tendo em vista que demanda um certo tempo para o aquífero responder integralmente a uma mudança de pressão no reservatório. Três modelos serão estudados neste projeto: Modelo de Van Everdingen & Hurst (1949), Modelo de Carter-Tracy (1960) e Modelo aproximado de Fetkovich (1971).
- Objetivos específicos:
  - Modelar física e matematicamente o problema.
  - Modelagem estática do software (diagramas de caso de uso, de pacotes, de classes).
  - Modelagem dinâmica do software (desenvolver algoritmos e diagramas exemplificando os fluxos de processamento).
  - Estimar o influxo de água com base em suposições que descrevem as características dos aquíferos.
  - Estimar o influxo de água através de modelos matemáticos que incluam a dependência do tempo.
  - Cálculo da vazão que o aquífero fornece (ou do influxo acumulado) em relação a queda de pressão.
  - Prever o desempenho de reservatórios de água.
  - Representar a compressibilidade do sistema.
  - Realizar simulações para teste do software desenvolvido.
  - Implementar manual simplificado de uso do software.

# Capítulo 2

## Especificação

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

### 2.1 O que é a especificação?

Nesta seção são descritas as principais características, além dos requisitos para a utilização do software desenvolvido.

### 2.2 Nome do sistema/produto

Nome	MODELOS DE AQUÍFEROS ANALÍTICOS
Componentes principais	Sistema para análise de modelos de aquíferos analíticos em reservatórios de óleo e gás.
Missão	Simular diferentes cenários do sistema com a presença de modelos distintos de aquíferos analíticos e sua influência na produtividade dos poços. Estimar o influxo de água. Cálculo da vazão que o aquífero fornece. Prever o desempenho dos reservatórios de óleo e gás. Gerar gráficos que permitam comparar poços com modelos de aquíferos de grandes dimensões.

## 2.3 Especificação

Será desenvolvido um software em modo texto para a análise do comportamento de reservatórios que produzem com a presença de modelos distintos de aquíferos analíticos. Os cálculos serão feitos a partir das propriedades do reservatório e dos aquíferos e as simulações serão feitas usando modelos como o de Van Everdingen & Hurst (1949), modelo de Carter-Tracy (1960) e modelo aproximado de Fetkovich (1971). Portanto, já que estamos tratando de dezenas de dados de propriedades de fluidos assim como dados de produção, o usuário poderá escolher realizar a entrada de dados por meio de um arquivo de entrada salvo em disco ou entrada direta via teclado. Ao final de cada simulação, o usuário definirá a saída dos dados resultantes da simulação, podendo ser: na tela, no disco e/ou em modo gráfico. Para as saídas gráficas serão apresentados gráficos de cada modelo de aquífero. Os gráficos serão gerados pelo programa gnuplot. Será fornecido um manual de ajuda para guiar aqueles em caso de dúvidas, explicando o funcionamento do software. O seguinte software será um software multi-plataforma desenvolvido para GNU/LINUX e Windows com licença GPL.

### 2.3.1 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais:

<b>RF-01</b>	O programa carrega dados de entrada.
<b>RF-02</b>	Deve permitir a geração de gráficos com o resultado da simulação.
<b>RF-03</b>	Deve mostrar os resultados na tela.
<b>RF-04</b>	Deve salvar os resultados em um arquivo de disco e em um diretório de escolha do usuário
<b>RF-05</b>	O usuário poderá visualizar seus resultados em um gráfico. O gráfico poderá ser salvo como imagem ou ter seus dados exportados como texto.

### 2.3.2 Requisitos não funcionais

<b>RNF-01</b>	Os cálculos devem ser feitos utilizando-se modelos matemáticos apresentados na literatura ou modelos idealizados que tenham os parâmetros carregados no software.
---------------	---

<b>RNF-02</b>	O programa deverá ser multi-plataforma, podendo ser executado em <i>Windows</i> , <i>GNU/Linux</i> ou <i>Mac</i> .
<b>RNF-03</b>	O software será escrito usando a linguagem C++ utilizando o paradigma de orientação à objeto.

## 2.4 Casos de uso

Esta seção contém uma tabela que descreve um caso de uso do sistema, assim como os diagramas de caso de uso.

Tabela 2.1: Caso de uso do sistema

Nome do caso de uso:	Influxo de água.
Resumo/descrição:	Cálculo do influxo para diferentes modelos de aquíferos.
Etapas:	<ol style="list-style-type: none"> <li>1. Entrada de dados.</li> <li>2. Calcular o influxo de água.</li> <li>3. Gerar gráficos.</li> <li>4. Salvar a simulação.</li> <li>5. Analisar resultados.</li> </ol>
Cenários alternativos:	Um cenário alternativo envolve uma entrada "errada" do usuário, onde o mesmo aplique dados de reservatório que não sejam coerentes, sendo assim, o programa plotará um gráfico discrepante com o que foi proposto.

### 2.4.1 O que são os diagramas de casos de uso?

O diagrama de caso de uso geral da Figura 2.1 mostra o usuário acessando o sistema do software para a realização do cálculo do influxo de água para diferentes modelos de aquíferos. Neste caso de uso geral, o usuário insere os dados de reservatórios reais ou ideais (via tela ou através de um arquivo .txt), o software realizará os cálculos, irá gerar os gráficos, para então analisar os resultados obtidos.





Figura 2.1: Diagrama de caso de uso – cálculo do influxo de água

### 2.4.2 Diagrama de caso de uso específico

O caso de uso descrito na Figura 2.1 e a Tabela 2.1 são detalhados nas Figuras 2.2 e 2.3. O usuário poderá escolher os diferentes modelos de aquíferos para realizar a simulação, plotar os gráficos do modelo escolhido, realizar a análise, fazer comparações e definir qual melhor modelo na produção de um poço de petróleo.

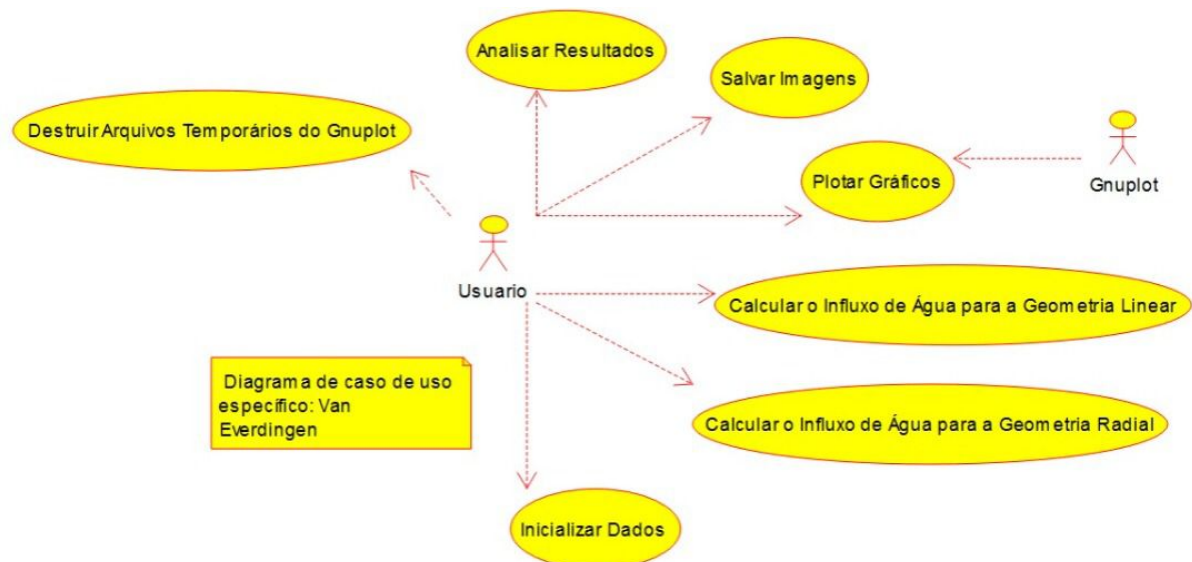


Figura 2.2: Diagrama de caso de uso específico do modelo Van Everdingen & Hurst



Figura 2.3: Diagrama de caso de uso específico dos modelos de Carter-Tracy e Fetkovich

# 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 as 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

Na Engenharia de Reservatórios, o influxo de água também pode ser referido como invasão de água ou influxo de aquífero. Este mecanismo de produção é composto por uma camada subterrânea de rocha porosa contendo um ou mais fluidos que podem permear para qualquer espaço disponível nessa rocha reservatório. Neste contexto, um aquífero é referido como um grande volume de água subjacente ao acúmulo de hidrocarbonetos na estrutura do reservatório, atuando como uma fonte de energia significativa para a produção de óleo e/ou gás, uma vez que o poço é perfurado [Okotie and Ikporo, 2019].

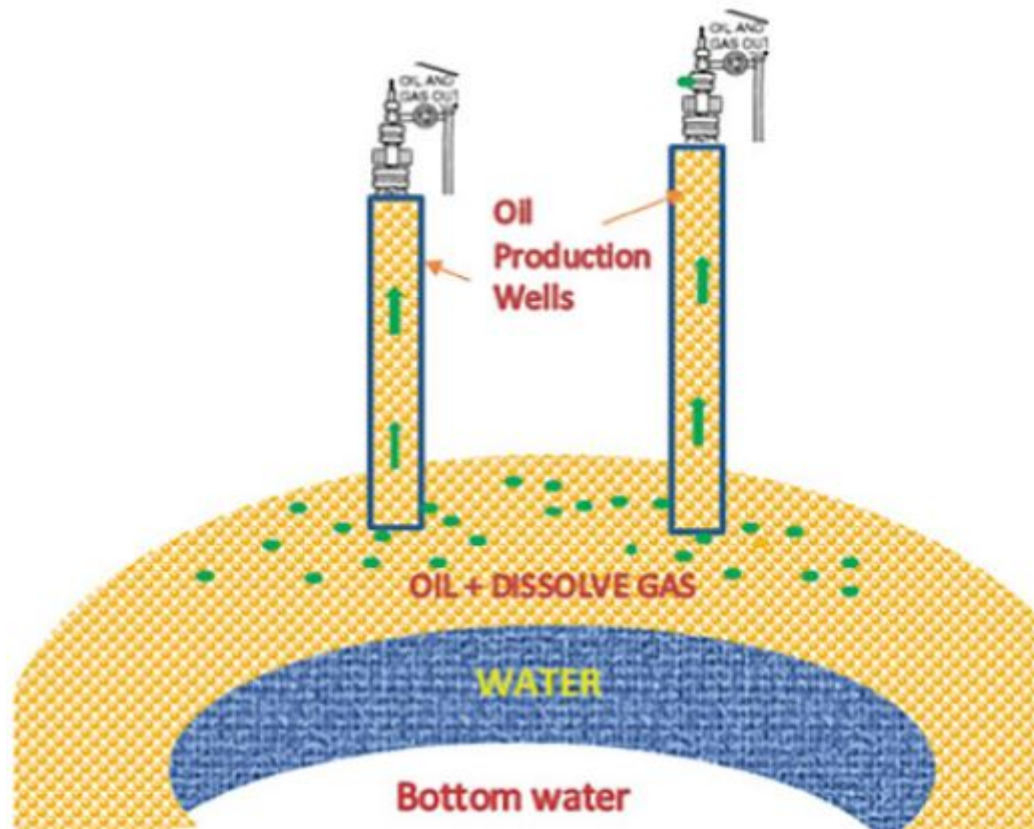


Figura 3.1: Esquema típico de um reservatório em produção associado a um aquífero. Fonte: [Okotie and Ikporo, 2019]

Segundo Ahmed (2006), é importante destacar que no gerenciamento desses reservatórios existem muitas incertezas associadas, principalmente quanto ao aquífero. Isso ocorre porque raramente se perfuram poços exploratórios em aquíferos para a caracterização petrofísica do mesmo, ou seja, conhecer os dados de porosidade, permeabilidade, espessura e propriedades do fluido. Ainda mais incerto, entretanto, é a geometria e a continuidade do próprio aquífero. Com isso, vários modelos foram desenvolvidos para estimar o influxo de água com base em suposições que descrevem essas características do aquífero, levando em consideração principalmente os dados históricos de desempenho do reservatório. A equação de balanço de materiais também pode ser usada para determinar o influxo de água, desde que o volume de óleo original seja conhecido a partir de estimativas de volume de poro da rocha. Isso permite a avaliação das constantes nas equações dos modelos para que a taxa de influxo de água futura possa ser prevista através do contato reservatório-aquífero [Ahmed, 2006].

Este projeto tem como objetivo mostrar o cálculo do influxo de água para diferentes modelos de aquíferos analíticos, se valendo do embasamento teórico para definir os parâmetros de reservatório, como por exemplo, sua geometria, ou seja, tudo que possa influenciar no influxo de água.

Depois de muito estudo e reflexão acerca do tema referido, do material acadêmico (tanto em livros quanto artigos científicos), foi possível identificar as seguintes áreas abor-

dadas pelo programa:

- A Engenharia de Reservatórios é o pilar deste projeto. O software deste trabalho traz conceitos dos modelos de influxo de água (Van Everdingen & Hurst, Carter-Tracy e Fetkovich).
- No que tange a Modelagem Numérica Computacional, serão utilizados neste software a equação de Bessel Modificada, o Algoritmo de Stehfest e a Transformada de Laplace, que são conceitos da área de Cálculo Numérico.
- Pacote gráfico: O software plotará os gráficos em cima dos respectivos resultados dos diferentes modelos de aquíferos analíticos.
- Software: Serão utilizadas bibliotecas já existentes para a utilização dos modelos de influxo de água.

## 3.2 Formulação teórica

Nesta seção será discutida a fundamentação teórica do software desenvolvido.

### 3.2.1 Equação da compressibilidade

O modelo mais simples para se estimar o influxo de água,  $w_e$ , baseia-se na **equação da compressibilidade** [Rosa, 2011]:

$$w_e = c_t w_i (p_i - p) \quad (3.1)$$

onde:

$w_e$ : influxo de água acumulado;

$c_t$ : compressibilidade total do aquífero;

$w_i$ : volume inicial de água do aquífero;

$p_i$ : pressão inicial;

$p$ : pressão no contato óleo/água;

$\Delta p = p_i - p_o$ : queda de pressão constante no contato o/a.

Esta equação, porém, só é aplicável a **aquíferos muito pequenos**, pois admite a equalização imediata de pressões entre o reservatório e o aquífero [Dake, 1983].

Segundo Dake (1978), para **aquíferos maiores**, torna-se necessário um modelo matemático que inclua a dependência do tempo, tendo em vista que demanda um certo tempo para o aquífero responder integralmente a uma mudança de pressão no reservatório. Entre os vários modelos existentes na literatura, serão apresentados neste trabalho os seguintes modelos:

- Modelo de Van Everdingen & Hurst (1949)
- Modelo de Carter-Tracy (1960)
- Modelo aproximado de Fetkovich (1971)

### 3.2.2 Modelo de Van Everdingen & Hurst (1949)

#### Aquífero Radial

O sistema reservatório-aquífero para este caso está representado na Figura 3.2.

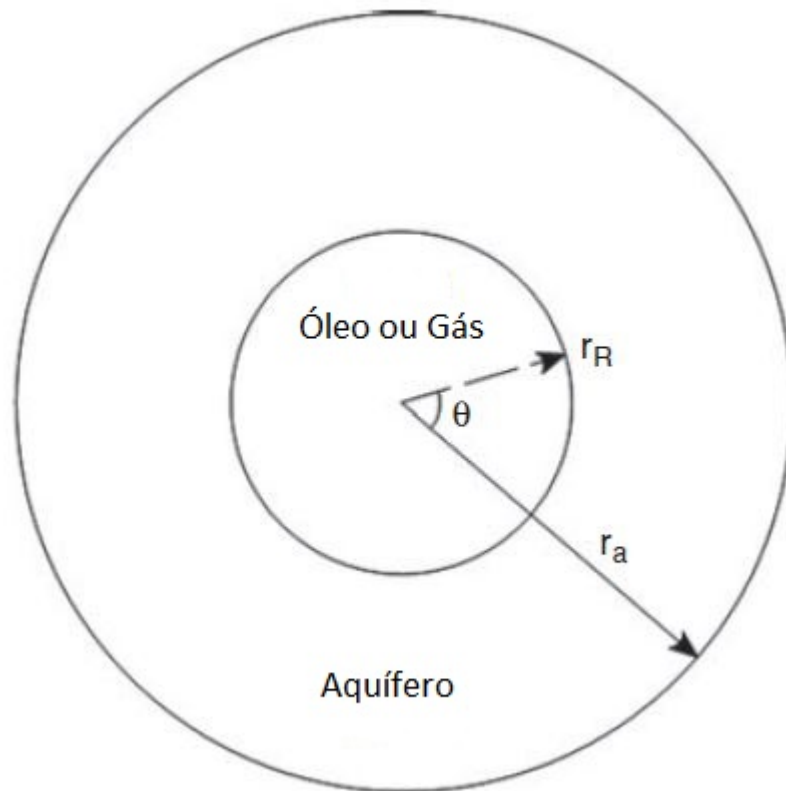


Figura 3.2: Esboço de um reservatório com aquífero radial. Fonte: [Ezekwe, 2011]

Inicialmente definem-se as variáveis adimensionais do modelo:

raio adimensional:

$$r_D = \frac{r}{r_0} \quad (3.2)$$

tempo adimensional:

$$t_D = \frac{kt}{\phi\mu c_t r_0^2} \quad (3.3)$$

pressão adimensional:

$$p_D = \frac{p_i - p}{p_i - p_0} = \frac{p_i - p}{\Delta p_0} \quad (3.4)$$

onde:

$\Delta p_0 = p_i - p_0$ : queda de pressão constante no contato, tomada como referência.

A vazão que o aquífero fornece, isto é, a vazão no ponto  $r = r_0$ , é dada pela lei de Darcy e pode ser escrita como:

$$q = \frac{2\pi fkh}{\mu} \left( r \frac{\partial p}{\partial r} \right)_{r_0} \quad (3.5)$$

onde:

$f = \theta/2\pi$ ,  $\theta$  (em radianos).

Usando as definições de variáveis adimensionais dadas pelas equações 3.2 e 3.4, a 3.5 pode também ser escrita em termos de variáveis adimensionais:

$$- \left( r_D \frac{\partial p_D}{\partial r_D} \right)_{r_D=1} = \frac{q\mu}{2\pi fkh\Delta p_0} \equiv q_D(t_D) \quad (3.6)$$

onde:

$q_D(t_D)$ : vazão adimensional fornecida pelo aquífero, ou seja, a vazão adimensional calculada no contato reservatório-aquífero (ponto  $r_D = 1$ ).

É mais conveniente expressar a solução em termos do influxo acumulado (integral da vazão) do que em termos da vazão. Resolvendo a equação 3.6 para a vazão e integrando em relação ao tempo obtém-se:

$$W_e \equiv \int_0^t q dt = \frac{2\pi fkh\Delta p_0}{\mu} \int_0^{t_D} q_D \frac{dt}{dt_D} dt_D \quad (3.7)$$

Da definição de tempo adimensional, dada pela equação 3.3, tem-se que:

$$\frac{dt}{dt_D} = \frac{\phi\mu c_t r_0^2}{k} \quad (3.8)$$

Substituindo a equação 3.8 na equação 3.7:

$$W_e = 2\pi f\phi c_t h r_0^2 \Delta p_0 \int_0^{t_D} q_D dt_D \quad (3.9)$$

Finalmente, denominando a integral de  $q_D$  em relação a  $t_D$  por  $W_D(t_D)$ , a equação 3.9 simplifica-se para:

$$W_e = U \Delta p_0 W_D(t_D) \quad (3.10)$$

onde:

$$U = 2\pi f\phi c_t h r_0^2 \quad (3.11)$$

sendo que:

$U$ : geralmente denominado constante de influxo de água do aquífero;

$W_D$ : influxo adimensional acumulado para uma queda de pressão constante no contato;

$W_D(t_D)$ : pode ser obtido resolvendo-se o problema do fluxo no aquífero.

Serão considerados três modelos de aquífero radial:

- a) Aquífero **infinito**;
- b) Aquífero finito **selado** no limite externo;
- c) Aquífero finito com manutenção de pressão no limite externo (**realimentado**).

A diferença entre estes modelos está apenas na condição de contorno externa (C.C.E.), isto é, na condição imposta no limite externo do aquífero.

O problema do fluxo no aquífero pode ser escrito matematicamente como:

E.D.P.:

$$\frac{\partial^2 p_D}{\partial r_D^2} + \frac{1}{r_D} \frac{\partial p_D}{\partial r_D} = \frac{\partial p_D}{\partial t_D} \quad (3.12)$$

C.I.:

$$p_D(r_D; t_D = 0) = 0 \quad (3.13)$$

C.C.I.:

$$p_D(r_D = 1; t_D) = 1 \quad (3.14)$$

A eq. 3.12, denominada equação da difusividade hidráulica, é:

(E.D.P.): a equação diferencial parcial que rege o fluxo no meio poroso, e pode ser obtida usando-se as definições das variáveis adimensionais na equação da difusividade hidráulica deduzida em termos de variáveis reais.

(C.I.): a condição inicial representa a condição de que inicialmente as pressões em qualquer ponto do aquífero estão em equilíbrio e iguais a  $p_i$ .

(C.C.I.): a condição de contorno interna impõe a condição de queda de pressão  $\Delta p_0 = p_i - p_0$  constante no contato aquífero-reservatório.

Consideram-se três opções para a condição de contorno externa (C.C.E.):



a) Aquífero **infinito**:

$$p(r_D \rightarrow \infty; t_D) = 0 \quad (3.15)$$

b) Aquífero finito **selado**:

Neste caso não há fluxo no limite externo e a C.C.E. pode ser escrita como:

$$\left( \frac{\partial p_D}{\partial r_D} \right)_{r_D=r_e/r_o} = 0 \quad (3.16)$$

onde:

$r_e$ : raio externo do aquífero;

$r_o$ : raio interno do aquífero.

c) Aquífero finito com pressão constante no limite externo (**realimentado**):

$$p_D(r_D = r_e/r_o; t_D) = 0 \quad (3.17)$$

Em qualquer dos casos o influxo pode ser calculado com a seguinte equação:

$$\mathbf{W}_D \equiv \int_0^{t_D} \mathbf{q}_D(t_D) dt_D = - \int_0^{t_D} \left( r_D \frac{\partial p_D}{\partial r_D} \right)_{r_D=1} dt_D \quad (3.18)$$

Os problemas matemáticos formados pelas equações 3.12 a 3.18 são normalmente resolvidos aplicando-se o conceito de Transformada de Laplace e suas soluções serão apresentadas no tópico de 3.2.2. Como as soluções, neste caso, são obtidas analiticamente apenas no campo de Laplace, algum tipo de inversão numérica tem que ser utilizado para se obter o comportamento de  $W_D$  em função de  $t_D$ . Para isto utiliza-se o algoritmo de Stehfest [Stehfest, 1970], que normalmente é utilizado para a inversão numérica da Transformada de Laplace.

É comum a apresentação dos valores de  $W_D$  (ou  $W_{eD}$ ) em função de  $t_D$  na forma de tabelas. Essas tabelas apresentam o resultado para o caso de aquífero radial infinito, apresentam o influxo acumulado adimensional  $W_D$  em função de  $t_D$  para o caso de aquífero finito para diferentes valores de raio externo adimensional  $r_{eD}$  (ou  $r_D$ ). Uma vez conhecido o influxo adimensional  $W_D$ , o influxo dimensional  $W_e$  é obtido com a eq. 3.10.

As Figuras 3.3 e 3.4 mostram o comportamento do influxo adimensional  $W_D$  para o aquífero radial em função do tempo adimensional  $t_D$  e do tamanho do aquífero dado pelo parâmetro ( $r_{eD}$ ). Inicialmente, independentemente da condição no seu limite externo, o aquífero se comporta como se fosse infinito. Quanto maior é o tamanho do aquífero, maior é o período em que o mesmo se comporta como aquífero infinito [Allard and Chen, 1988].

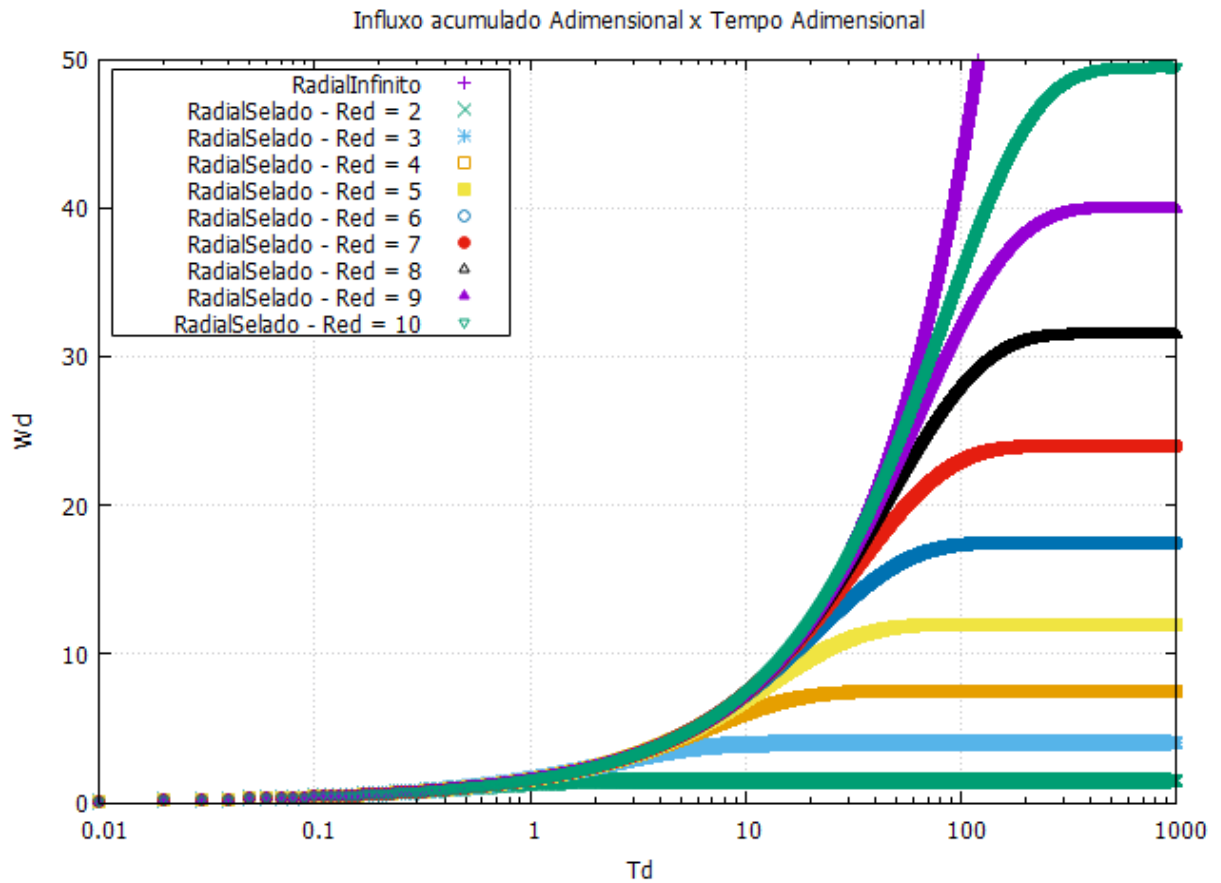


Figura 3.3: Gráfico de  $W_D \times t_D$  para vários tamanhos de aquíferos radiais selados. Fonte: Elaborado pelo autor com dados de Van Everdingen e Hurst (1949)

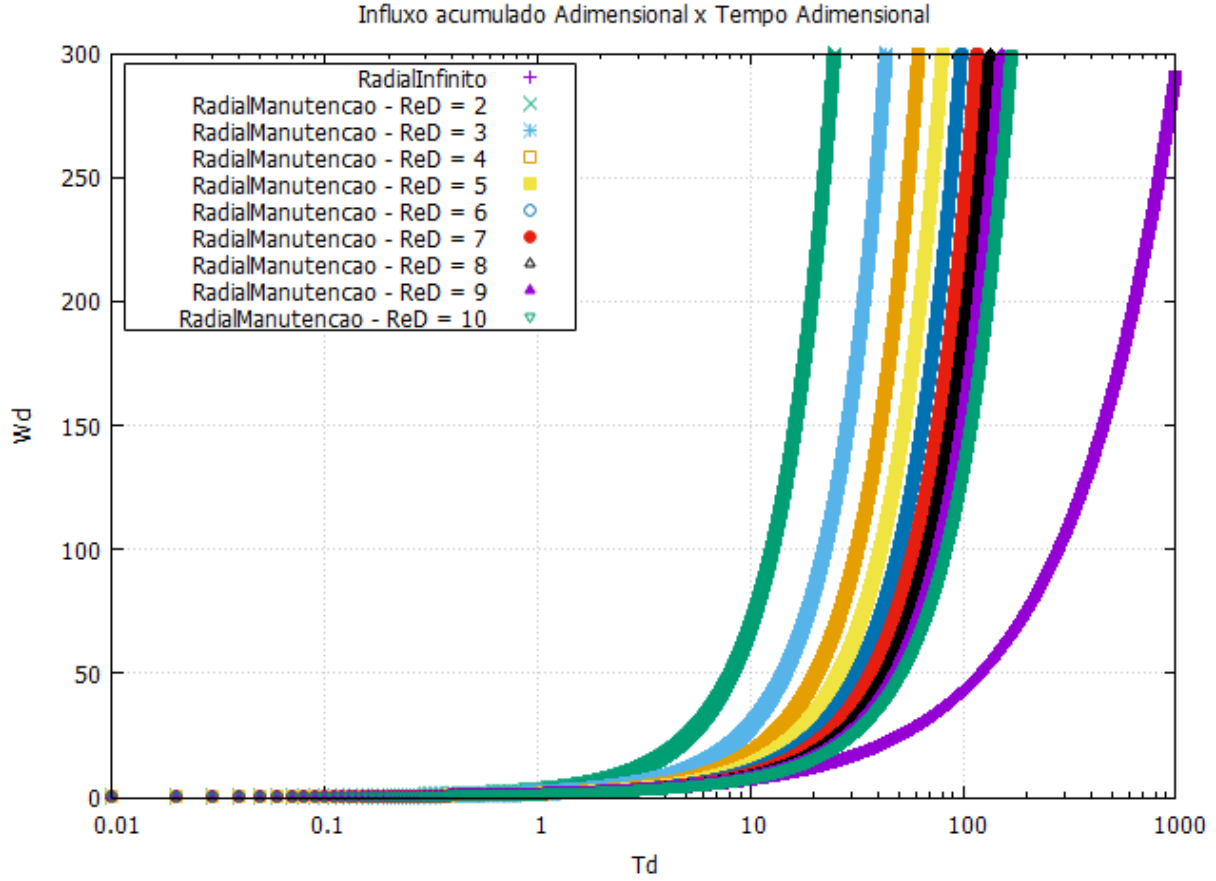


Figura 3.4: Gráfico de  $W_D$  x  $t_D$  para vários tamanhos de aquíferos radiais realimentados. Fonte: Elaborado pelo autor com dados de Van Everdingen e Hurst (1949)

Para os aquíferos selados existe um valor máximo para o influxo acumulado. Esse valor máximo é alcançado após a equalização das pressões do reservatório (no contato) e do aquífero. A partir da equação da compressibilidade (3.1), pode-se mostrar que o influxo máximo é dado pela equação:

$$W_{Dmáx} = \frac{r_{eD}^2 - 1}{2} \quad (3.19)$$

### Soluções clássicas para o aquífero radial

As soluções dos problemas apresentados neste tópico foram obtidas utilizando o método de Transformada de Laplace, onde:

$\bar{W}_D(u)$ : denota a Transformada de Laplace de  $W_D(t_D)$ ;

$u$ : variável de Laplace.

As soluções são escritas em termos das funções de Bessel modificadas de primeira espécie ( $I_0$  e  $I_1$ ) e de segunda espécie ( $K_0$  e  $K_1$ ) de ordens zero e um, respectivamente.

**a) Aquífero radial infinito****Problema:**

E.D.P:

$$\frac{\partial^2 p_D}{\partial r_D^2} + \frac{1}{r_D} \frac{\partial p_D}{\partial r_D} = \frac{\partial p_D}{\partial t_D} \quad (3.20)$$

C.I:

$$p_D(r_D; t_D = 0) = 0 \quad (3.21)$$

C.C.I:

$$p_D(r_D = 1; t_D) = 1 \quad (3.22)$$

C.C.E:

$$p_D(r_D \rightarrow \infty; t_D) = 0 \quad (3.23)$$

**Eq. geral do influxo acumulado:**

$$W_D \equiv \int_0^{t_D} q_D(t_D) dt_D = - \int_0^{t_D} \left( r_D \frac{\partial p_D}{\partial r_D} \right)_{r_D=1} dt_D \quad (3.24)$$

**Solução:**

$$\overline{W}_D(u) = \frac{k_1(\sqrt{u})}{u^{3/2} k_0(\sqrt{u})} \quad (3.25)$$

**b) Aquífero radial finito selado no limite externo****Problema:**

E.D.P.:

$$\frac{\partial^2 p_D}{\partial r_D^2} + \frac{1}{r_D} \frac{\partial p_D}{\partial r_D} = \frac{\partial p_D}{\partial t_D} \quad (3.26)$$

C.I.:

$$p_D(r_D; t_D = 0) = 0 \quad (3.27)$$

C.C.I.:

$$p_D(r_D = 1; t_D) = 1 \quad (3.28)$$

C.C.E.:

$$\left( \frac{\partial p_D}{\partial r_D} \right)_{r_D=r_e/r_o} = 0 \quad (3.29)$$

**Eq. geral do influxo acumulado:**

$$W_D \equiv \int_0^{t_D} q_D(t_D) dt_D = - \int_0^{t_D} \left( r_D \frac{\partial p_D}{\partial r_D} \right)_{r_D=1} dt_D \quad (3.30)$$

**Solução:**

$$\overline{W}_D(u) = \frac{I_1(r_{eD}\sqrt{u})k_1(\sqrt{u}) - I_1(\sqrt{u})k_1(r_{eD}\sqrt{u})}{u^{3/2} [I_0(\sqrt{u})k_1(r_{eD}\sqrt{u}) + I_1(r_{eD}\sqrt{u})k_0(\sqrt{u})]} \quad (3.31)$$

c) Aquífero radial finito com pressão constante no limite externo (realimentado)

**Problema:**

E.D.P.:

$$\frac{\partial^2 p_D}{\partial r_D^2} + \frac{1}{r_D} \frac{\partial p_D}{\partial r_D} = \frac{\partial p_D}{\partial t_D} \quad (3.32)$$

C.I.:

$$p_D(r_D; t_D = 0) = 0 \quad (3.33)$$

C.C.I.:

$$p_D(r_D = 1; t_D) = 1 \quad (3.34)$$

C.C.E.:

$$p_D(r_D = r_e/r_o; t_D) = 0 \quad (3.35)$$

**Eq. geral do influxo acumulado:**

$$W_D \equiv \int_0^{t_D} q_D(t_D) dt_D = - \int_0^{t_D} \left( r_D \frac{\partial p_D}{\partial r_D} \right)_{r_D=1} dt_D \quad (3.36)$$

**Solução:**

$$\overline{W}_D(u) = \frac{I_0(r_{eD}\sqrt{u})k_1(\sqrt{u}) + I_1(\sqrt{u})k_0(r_{eD}\sqrt{u})}{u^{3/2} [I_0(r_{eD}\sqrt{u})k_0(\sqrt{u}) - I_0(\sqrt{u})k_0(r_{eD}\sqrt{u})]} \quad (3.37)$$

### Aquífero Linear

A Figura 3.5 mostra um sistema reservatório-aquífero para o modelo de fluxo linear:

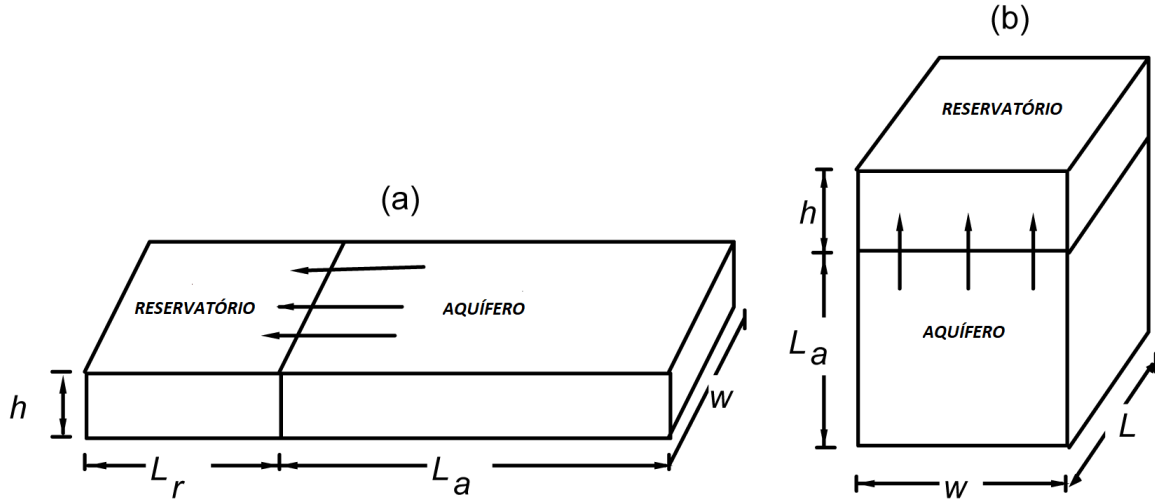


Figura 3.5: Modelo de aquífero linear: (a) na lateral e (b) na base

Normalmente o comprimento do aquífero,  $L$ , é utilizado como comprimento de referência nas definições de variáveis adimensionais. Neste caso as variáveis adimensionais são dadas por:

comprimento adimensional:

$$x_D = \frac{x}{L} \quad (3.38)$$

tempo adimensional:

$$t_D = \frac{kt}{\phi\mu c_t L^2} \quad (3.39)$$

pressão adimensional:

$$p_D = \frac{p_i - p}{p_i - p_0} = \frac{p_i - p}{\Delta p_0} \quad (3.40)$$

onde:

$\Delta p_0 = p_i - p_0$ : queda de pressão constante no contato, tomada como referência.

**obs.:** Para o modelo de aquífero infinito,  $L$  passa a ser apenas um comprimento de referência arbitrário, sem qualquer significado físico.

A vazão que o aquífero fornece, isto é, a vazão no ponto  $x = 0$  é dada pela lei de Darcy para o fluxo linear:

$$q = \frac{kA}{\mu} \left( \frac{\partial p}{\partial x} \right)_{x=0} \quad (3.41)$$

onde:

$A$ : área aberta ao fluxo, ou seja,  $A = wh$ .

Usando as definições dadas pelas equações 3.38 e 3.40, a equação 3.41 pode ser escrita em termos de variáveis adimensionais como:

$$-\left(\frac{\partial p_D}{\partial x_D}\right)_{x_D=0} = \frac{q\mu L}{kA\Delta p_0} \equiv q_D(t_D) \quad (3.42)$$

onde:

$q_D(t_D)$ : vazão adimensional fornecida pelo aquífero, isto é, a vazão adimensional calculada no contato reservatório-aquífero (ponto  $x_D = 0$ ).

Como o interesse está na solução em termos do influxo acumulado (integral da vazão), resolvendo a equação 3.42 para a vazão e integrando em relação ao tempo resulta em:

$$W_e \equiv \int_0^t q dt = \frac{kA\Delta p_0}{\mu L} \int_0^{t_D} q_D \frac{dt}{dt_D} dt_D \quad (3.43)$$

Mas da equação 3.39 têm-se que:

$$\frac{dt}{dt_D} = \frac{\phi\mu c_t L^2}{k} \quad (3.44)$$

Finalmente, substituindo a equação 3.44 na equação 3.43, obtém-se:

$$W_e = wLh\phi c_t \Delta p_0 \int_0^{t_D} q_D dt_D \quad (3.45)$$

ou ainda:

$$W_e = U\Delta p_0 W_D(t_D) \quad (3.46)$$

onde:

$$U = wLh\phi c_t \quad (3.47)$$

sendo que:

$W_D$ : influxo acumulado adimensional para uma queda de pressão  $\Delta p_0$  constante no contato é obtido resolvendo o problema do fluxo no aquífero para os modelos de interesse.

Consideram-se aqui também três modelos de aquíferos lineares:

- a) Aquífero **infinito**;
- b) Aquífero finito **selado** no limite externo;
- c) Aquífero finito com manutenção de pressão no limite externo (**realimentado**).

Utilizando as definições de variáveis adimensionais e a equação da difusividade hidráulica em termos de variáveis reais, neste caso o problema do fluxo no aquífero pode ser escrito matematicamente como:

E.D.P.:

$$\frac{\partial^2 p_D}{\partial x_D^2} = \frac{\partial p_D}{\partial t_D} \quad (3.48)$$

C.I.:

$$p_D(x_D; t_D = 0) = 0 \quad (3.49)$$

C.C.I.:

$$p_D(x_D = 0; t_D) = 1 \quad (3.50)$$

A condição de contorno externa (C.C.E.) depende do modelo considerado:

a) Aquífero **infinito**:

$$p_D(x_D \rightarrow \infty; t_D) = 0 \quad (3.51)$$

b) Aquífero finito **selado** no limite externo:

$$\left( \frac{\partial p_D}{\partial x_D} \right)_{x_D=1} = 0 \quad (3.52)$$

c) Aquífero finito com pressão constante no limite externo (**realimentado**):

$$p_D(x_D = 1; t_D) = 0 \quad (3.53)$$

O influxo acumulado adimensional pode ser calculado com a seguinte equação:

$$W_D \equiv \int_0^{t_D} q_D(t_D) dt_D = - \int_0^{t_D} \left( \frac{\partial p_D}{\partial x_D} \right)_{x_D=0} dt_D \quad (3.54)$$

As soluções para os problemas formados pelas equações 3.48 a 3.54 são também obtidas aplicando-se o conceito de Transformada de Laplace e as suas soluções serão apresentadas no tópico de 3.2.2. Neste caso, existe solução analítica no campo real para o modelo de aquífero infinito. As soluções dos outros modelos são obtidas analiticamente apenas no campo de Laplace. Novamente o algoritmo de Stehfest [Stehfest, 1970] é utilizado para inverter numericamente as soluções e obter gráficos de  $W_D$  versus  $t_D$ .

Uma vez conhecido o influxo adimensional  $W_D$ , o influxo dimensional  $W_e$  é obtido com a eq. 3.46. O comportamento do influxo acumulado adimensional em função do tipo de condição de contorno externa é mostrado na Figura 3.6.



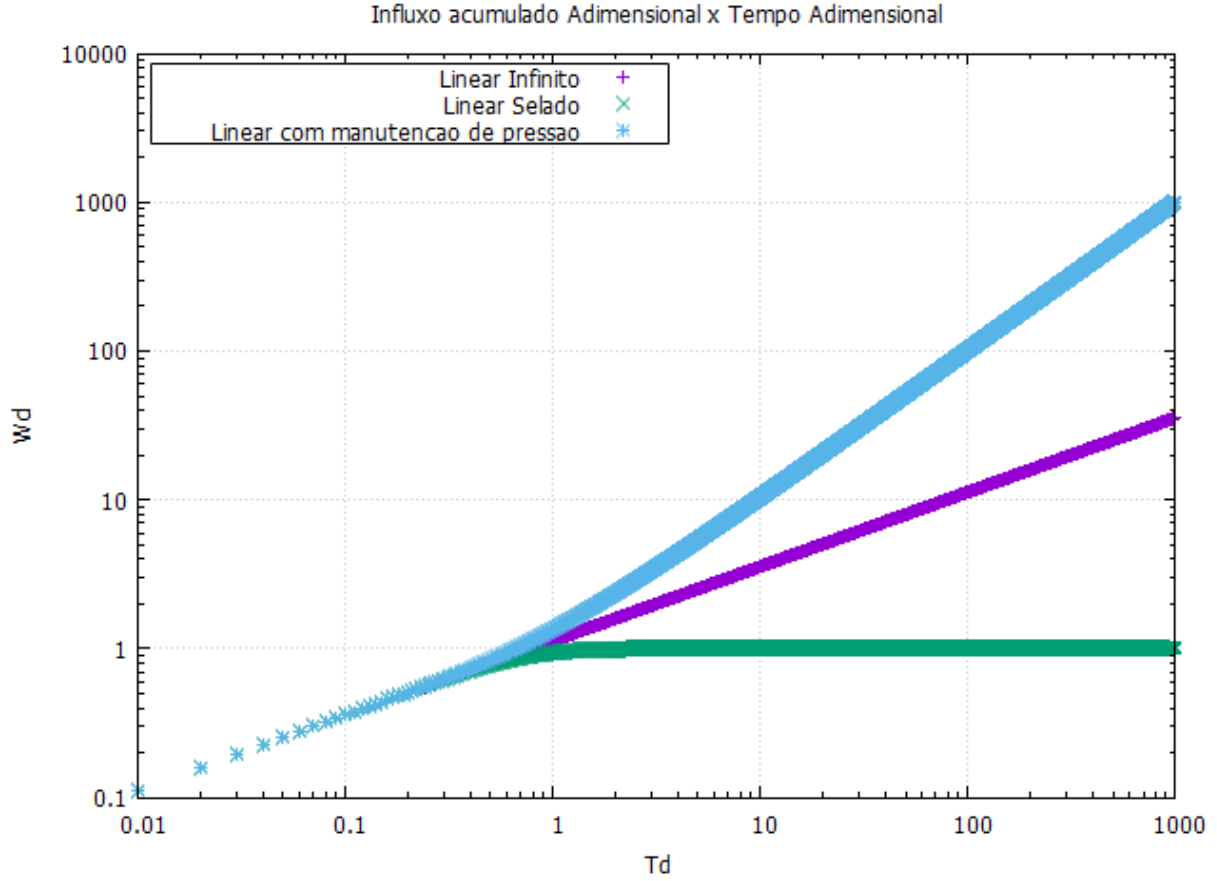


Figura 3.6: Influxo adimensional  $W_D$  em função do tempo adimensional  $t_D$  contemplando os três modelos de aquíferos lineares. Fonte: Elaborado pelo autor com dados de Van Everdingen e Hurst (1949)

### Soluções clássicas para o aquífero linear

As soluções dos problemas apresentados neste tópico foram obtidas utilizando o método de Transformada de Laplace, onde:

$\overline{W}_D(u)$ : denota a Transformada de Laplace de  $W_D(t_D)$ ;

$u$ : variável de Laplace.

#### a) Aquífero linear infinito

**Problema:**

E.D.P.:

$$\frac{\partial^2 p_D}{\partial x_D^2} = \frac{\partial p_D}{\partial t_D} \quad (3.55)$$

C.I.:

$$p_D(x_D, t_D = 0) = 0 \quad (3.56)$$

C.C.I.:

$$p_D(x_D = 0; t_D) = 1 \quad (3.57)$$

C.C.E.:

$$p_D(x_D \rightarrow \infty; t_D) = 0 \quad (3.58)$$

**Eq. geral do influxo acumulado:**

$$W_D \equiv \int_0^{t_D} q_D(t_D) dt_D = - \int_0^{t_D} \left( \frac{\partial p_D}{\partial x_D} \right)_{x_D=0} dt_D \quad (3.59)$$

**Solução:**

$$\mathbf{W}_D(\mathbf{t}_D) = 2\sqrt{\frac{\mathbf{t}_D}{\pi}} \quad (3.60)$$

**b) Aquífero linear selado no limite externo**

**Problema:**

E.D.P:

$$\frac{\partial^2 p_D}{\partial x_D^2} = \frac{\partial p_D}{\partial t_D} \quad (3.61)$$

C.I:

$$p_D(x_D; t_D = 0) = 0 \quad (3.62)$$

C.C.I.:

$$p_D(x_D = 0; t_D) = 1 \quad (3.63)$$

C.C.E.:

$$\left( \frac{\partial p_D}{\partial x_D} \right)_{x_D=1} = 0 \quad (3.64)$$

**Eq. geral do influxo acumulado:**

$$W_D \equiv \int_0^{t_D} q_D(t_D) dt_D = - \int_0^{t_D} \left( \frac{\partial p_D}{\partial x_D} \right)_{x_D=0} dt_D \quad (3.65)$$

**Solução:**

$$\overline{\mathbf{W}}_D(\mathbf{u}) = \frac{1 - \exp(-2\sqrt{\mathbf{u}})}{\mathbf{u}^{3/2} [1 + \exp(-2\sqrt{\mathbf{u}})]} \quad (3.66)$$

**c) Aquífero linear finito com pressão constante no limite externo (realimentado)**

**Problema:**

E.D.P.:

$$\frac{\partial^2 p_D}{\partial x_D^2} = \frac{\partial p_D}{\partial t_D} \quad (3.67)$$

C.I.:

$$p_D(x_D; t_D = 0) = 0 \quad (3.68)$$

C.C.I.:

$$p_D(x_D = 0; t_D) = 1 \quad (3.69)$$

C.C.E.:

$$p_D(x_D = 1; t_D) = 0 \quad (3.70)$$

**Eq. geral do influxo acumulado:**

$$W_D \equiv \int_0^{t_D} q_D(t_D) dt_D = - \int_0^{t_D} \left( \frac{\partial p_D}{\partial x_D} \right)_{x_D=0} dt_D \quad (3.71)$$

**Solução:**

$$\overline{W}_D(u) = \frac{1 + \exp(-2\sqrt{u})}{u^{3/2} [1 - \exp(-2\sqrt{u})]} \quad (3.72)$$

### 3.2.3 Modelo de Carter-Tracy (1960)

Este método é uma solução aproximada para a equação da difusividade. Ele pode ser combinado convenientemente com uma equação de balanço de materiais adequada para prever o desempenho de reservatórios de água. Além disso, destaca-se sua aplicabilidade em aquíferos de ação finita e infinita, assim como àqueles identificados como radiais e lineares [Okotie and Ikporo, 2019].

O modelo de Carter-Tracy é aplicável a qualquer geometria de fluxo, desde que se conheça a solução para a pressão adimensional em função do tempo na geometria de aquífero considerada. Esta abrangência de tipos de aquífero contemplados é uma grande vantagem sobre o modelo de Van Everdingen e Hurst. Salienta-se também que o modelo de Carter-Tracy, assim como o de Fetkovich, não requer a aplicação do princípio da superposição de efeitos no cálculo do influxo [Rosa, 2011].

O influxo acumulado de água desse modelo pode ser expresso através da integral de convolução [Carter and Tracy, 1960]:

$$W_e(t_{Dj}) = U \int_0^{t_{Dj}} \Delta p(\tau) \frac{dW_D(t_D - \tau)}{d\tau} d\tau \quad (3.73)$$

onde:

$t_D$ : tempo adimensional definido para cada geometria de aquífero;

$U$ : constante de influxo de água e que também depende da geometria do sistema;

$\Delta p(t_D) = p_i - p(t_D)$ : queda de pressão no contato;

$W_D(t_D)$ : influxo de água acumulado adimensional;

$\tau$ : variável de integração;

$j$ : discretização do tempo.

As definições de  $t_D$  e de  $U$  variam para os casos de fluxo radial e linear. Os valores de  $W_D(t_D)$  idem.

No modelo de Carter-Tracy o valor do influxo acumulado  $W_e$  é aproximado pela expressão:

$$W_e(t_{D_j}) = W_e(t_{D_{j-1}}) + a_{j-1}(t_{D_j} - t_{D_{j-1}}) \quad (3.74)$$

onde:

$a_{j-1}$ : é uma constante.

Esta equação admite que no intervalo entre  $t_{D_{j-1}}$  e  $t_{D_j}$  o influxo varia linearmente com o tempo.

Combinando-se as equações 3.73 e 3.74 obtém-se:

$$U \int_0^{t_{D_j}} \Delta p(\tau) \frac{dW_D(t_D - \tau)}{d\tau} d\tau = W_e(t_{D_{j-1}}) + a_{j-1}(t_{D_j} - t_{D_{j-1}}) \quad (3.75)$$

Aplicando-se a Transformada de Laplace em relação ao tempo adimensional a equação 3.75, obtém-se:

$$U u \overline{\Delta p(u)} \overline{W_D}(u) = \frac{W_e(t_{D_{j-1}}) - a_{j-1} t_{D_{j-1}}}{u} + \frac{a_{j-1}}{u^2} \quad (3.76)$$

onde:

$u$ : variável de Laplace.

Segundo Van Everdingen e Hurst (1949), para o problema em questão é possível provar que:

$$\frac{1}{u^2} = u \overline{p_D}(u) \overline{W_D}(u) \quad (3.77)$$

onde:

$p_D(t_D)$ : solução para a pressão adimensional na face interna de um aquífero produzindo com vazão constante;

$W_D(t_D)$ : influxo adimensional para o caso de pressão constante no contato.

Substituindo-se a equação 3.77 na equação 3.76 e explicitando-se a transformada da queda de pressão no contato óleo/água, obtém-se a equação:

$$\overline{\Delta p}(u) = \frac{1}{U} \{ [W_e(t_{D_{j-1}}) - a_{j-1}t_{D_{j-1}}] u\overline{p_D}(u) + a_{j-1}\overline{p_D}(u) \} \quad (3.78)$$

cujas inversões resultam em:

$$\Delta p(t_{D_j}) = \frac{1}{U} \{ [W_e(t_{D_{j-1}}) - a_{j-1}t_{D_{j-1}}] p'_D(t_{D_j}) + a_{j-1}p_D(t_{D_j}) \} \quad (3.79)$$

onde  $p'_D(t_{D_j})$  é a derivada da pressão adimensional em relação ao tempo adimensional.

Explicitando-se a constante  $a_{j-1}$  na eq. 3.79:

$$a_{j-1} = \frac{U \Delta p(t_{D_j}) - W_e(t_{D_{j-1}}) p'_D(t_{D_j})}{p_D(t_{D_j}) - t_{D_{j-1}} p'_D(t_{D_j})} \quad (3.80)$$

e substituindo a expressão resultante na eq. 3.74, obtém-se:

$$W_e(t_{D_j}) = W_e(t_{D_{j-1}}) + \frac{U \Delta p(t_{D_j}) - W_e(t_{D_{j-1}}) p'_D(t_{D_j})}{p_D(t_{D_j}) - t_{D_{j-1}} p'_D(t_{D_j})} (t_{D_j} - t_{D_{j-1}}) \quad (3.81)$$

que é a equação para o cálculo do influxo acumulado.

Conforme mencionado anteriormente, a função  $p_D(t_D)$  representa a pressão adimensional na face interna de um aquífero produzindo com vazão constante. No caso de um aquífero linear infinito, isto é, de um aquífero que se comporta ainda no regime transiente de fluxo, a pressão adimensional é determinada pela expressão:

$$p_D(t_D) = 2\sqrt{\frac{t_D}{\pi}} \quad (3.82)$$

e no caso de um aquífero radial infinito essa expressão é, aproximadamente:

$$p_D(t_D) = \frac{1}{2} [\ln(t_D) + 0,80907] \quad (3.83)$$

A seguir, a Figura 3.7 ilustra a aplicação do modelo de Carter-Tracy em um reservatório circundado por um aquífero radial selado:

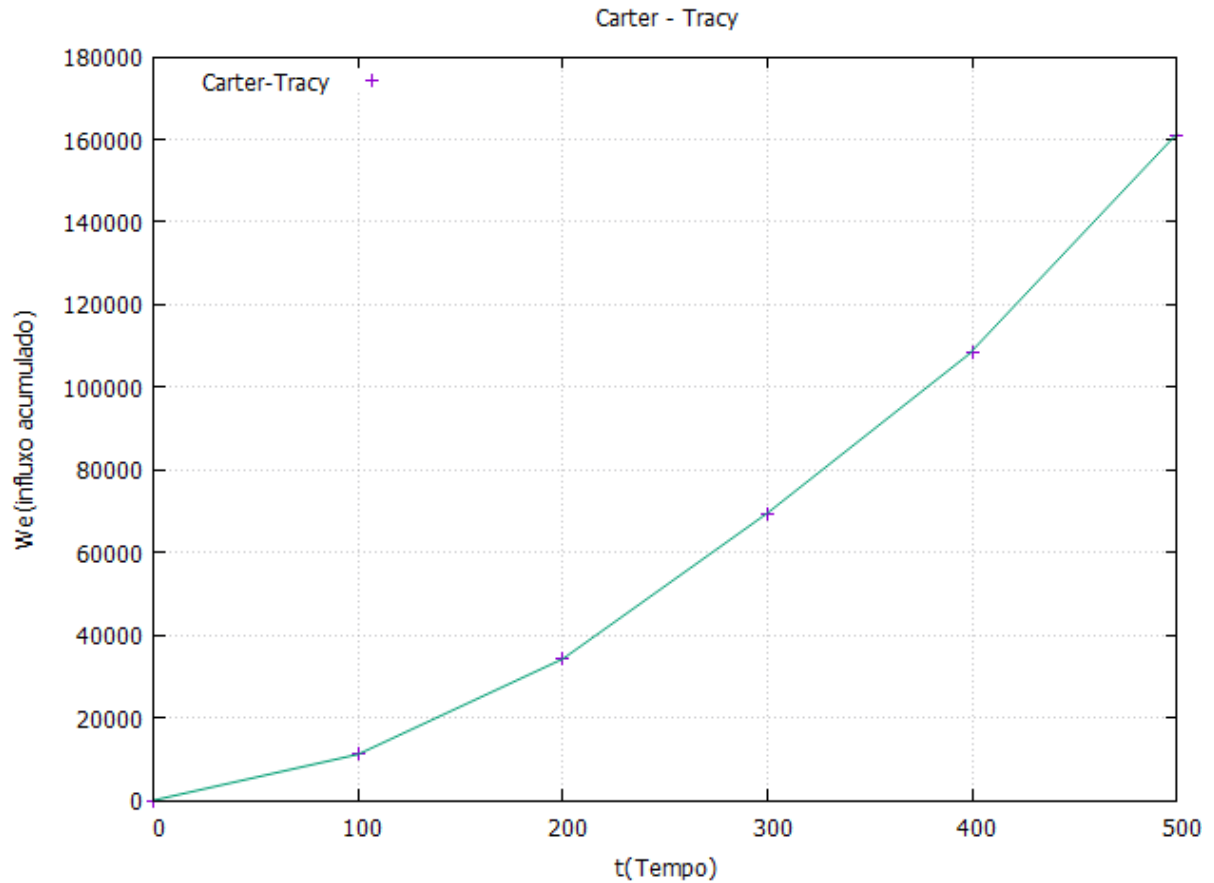


Figura 3.7: Gráfico de  $W_e \times t$  utilizando o modelo de Carter-Tracy (1960) aplicado a um reservatório circundado por um aquífero radial selado. Fonte: Elaborado pelo autor com dados do [Rosa, 2011]

### 3.2.4 Modelo Aproximado de Fetkovich (1971)

Fetkovich (1971) propôs um modelo para simplificar ainda mais os cálculos do influxo de água. Este método usa um índice de produtividade do aquífero ( $J$ ) em estado pseudo-permanente e um balanço de materiais do aquífero para representar a compressibilidade do sistema [Okotie and Ikporo, 2019].

Como o modelo de Carter-Tracy, o modelo de Fetkovich elimina o uso de superposição de efeitos e, portanto, é muito mais simples do que o método de Van Everdingen e Hurst, pois o cálculo do passo seguinte não necessita dos cálculos dos passos anteriores. No entanto, como Fetkovich negligencia o período de tempo transitório inicial nesses cálculos, o influxo de água calculado será sempre menor do que os valores previstos pelos outros dois modelos explicitados neste trabalho [Okotie and Ikporo, 2019].

Segundo [Fetkovich, 1971], admite-se o cálculo do fluxo do aquífero para o reservatório através da seguinte equação:

$$q = \frac{dW_e}{dt} = J(\bar{p}_a - p) \quad (3.84)$$

onde:

$J$ : índice de produtividade do aquífero;

$\bar{p}_a$ : pressão média do aquífero;

$p$ : pressão no contato reservatório-aquífero.

A partir do balanço de materiais no aquífero pode-se escrever que:

$$W_e = c_t W_i (p_i - \bar{p}_a) \quad (3.85)$$

onde:

$c_t = c_w + c_f$ : compressibilidade total do aquífero;

$W_i$ : volume inicial de água.

Rearranjando a equação 3.85, têm-se:

$$\bar{p} = p_i \left( 1 - \frac{W_e}{c_t W_i p_i} \right) \quad (3.86)$$

Seja  $W_{ei}$  o influxo máximo que um aquífero selado pode fornecer, correspondente à expansão da água do aquífero ao ser despressurizada de  $p_i$  para a pressão zero. Da equação 3.85,

$$W_{ei} = c_t W_i p_i \quad (3.87)$$

Substituindo a equação 3.87 na equação 3.86, obtém-se:

$$\bar{p}_a = p_i \left( 1 - \frac{W_e}{W_{ei}} \right) \quad (3.88)$$

cuja derivada em relação ao tempo é dada por:

$$\frac{d\bar{p}_a}{dt} = -\frac{p_i}{W_{ei}} \frac{dW_e}{dt} \quad (3.89)$$

Substituindo a equação 3.84 na equação 3.89, obtém-se:

$$\frac{d\bar{p}_a}{dt} = -\frac{p_i}{W_{ei}} J (\bar{p}_a - p) \quad (3.90)$$

ou

$$-\frac{J p_i}{W_{ei}} dt = \int_{p_i}^{\bar{p}_a} \frac{d\bar{p}_a}{\bar{p}_a - p} \quad (3.91)$$

após separar as variáveis. Esta equação pode ser integrada de  $t = 0$  (quando  $W_e = 0$  e  $\bar{p}_a = p_i$ ) a  $t$ , isto é, pode-se escrever:

$$-\frac{J p_i}{W_{ei}} \int_0^t dt = \int_{p_i}^{\bar{p}_a} \frac{d\bar{p}_a}{\bar{p}_a - p} \quad (3.92)$$

Resolvendo as integrais, a equação 3.92 simplifica-se para:

$$-\frac{Jp_i}{W_{ei}}t = \ln \left( \frac{\bar{p}_a - p}{p_i - p} \right) \quad (3.93)$$

ou ainda:

$$\bar{p}_a - p = (p_i - p) \exp \left( -\frac{Jp_i}{W_{ei}}t \right) \quad (3.94)$$

Substituindo a equação 3.94 na equação 3.84 :

$$q = J(p_i - p) \exp \left( -\frac{Jp_i}{W_{ei}}t \right) \quad (3.95)$$

A equação 3.95 é a equação da vazão com que a água flui do aquífero para o reservatório em função do tempo e da queda de pressão no contato,  $(p_i - p)$ . Esta equação é geral e independe da geometria do aquífero. A equação 3.95 pode ser integrada para se obter o influxo. Assim,

$$W_e \equiv \int_0^t q dt = J(p_i - p) \int_0^t \exp \left( -\frac{Jp_i}{W_{ei}}t \right) dt \quad (3.96)$$

Finalmente, resolvendo a integral da eq. 3.96 chega-se a:

$$W_e = \frac{W_{ei}}{p_i} (p_i - p) \left[ 1 - \exp \left( -\frac{Jp_i}{W_{ei}}t \right) \right] \quad (3.97)$$

A seguir, apresenta-se um exemplo da aplicação do modelo de Fetkovich em um reservatório circundado por um aquífero radial selado:



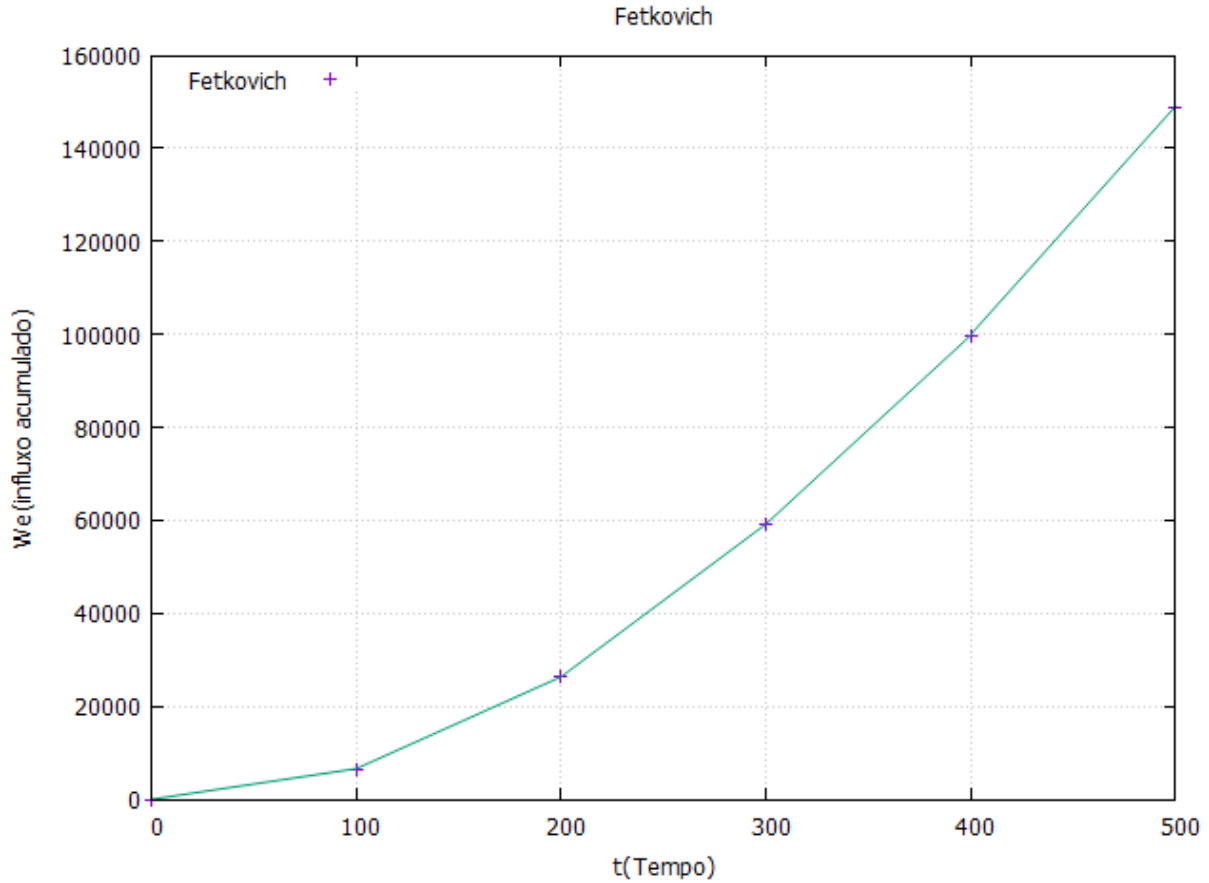


Figura 3.8: Gráfico de  $W_e$  x  $t$  utilizando o modelo de Fetkovich (1971) aplicado a um reservatório circundado por um aquífero radial selado. Fonte: Elaborado pelo autor com dados do Rosa

Ainda sobre a eq. 3.97, é importante destacar que ela fornece o influxo do aquífero em função do tempo para uma queda de pressão constante,  $(p_i - p)$ , no contato. Com isso, algumas observações podem ser feitas a respeito das equações do modelo de Fetkovich:

#### a) Observação 1

Com o passar do tempo, a vazão fornecida pelo aquífero, eq. 3.95, decresce exponencialmente tendendo a zero. Ou seja, o influxo dado pela eq. 3.97 tende a um valor máximo. Tomando o limite da eq. 3.97 para  $t \rightarrow \infty$  e usando a eq. 3.87, o influxo máximo pode ser escrito como:

$$W_{emáx} = \frac{W_{ei}}{p_i}(p_i - p) = c_t W_i(p_i - p) \quad (3.98)$$

#### b) Observação 2

Na prática a queda de pressão no contato não é constante e a eq. 3.97 não é diretamente aplicável. Fetkovich mostrou uma forma de utilizar a eq. 3.97 quando a pressão varia no

contato, sem fazer a superposição de efeitos.

O influxo durante o primeiro intervalo de tempo ( $\Delta t_1$ ) pode ser expresso por:

$$\Delta W_{e1} = \frac{W_{ei}}{p_i} (p_i - \bar{p}) \left[ 1 - \exp \left( -\frac{J p_i}{W_{ei}} \Delta t_1 \right) \right] \quad (3.99)$$

onde  $\bar{p}_1$  é a média das pressões no contato no intervalo de tempo  $\Delta t_1$ .

Para o segundo intervalo de tempo ( $\Delta t_2$ ):

$$\Delta W_{e2} = \frac{W_{ei}}{p_i} (\bar{p}_{a1} - \bar{p}_2) \left[ 1 - \exp \left( -\frac{J p_i}{W_{ei}} \Delta t_2 \right) \right] \quad (3.100)$$

onde  $\bar{p}_{a1}$  é a pressão média do aquífero no final do primeiro intervalo de tempo e é calculada a partir da equação de balanço de materiais no aquífero, eq. 3.90,

$$\bar{p}_{a1} = p_i \left( 1 - \frac{\Delta W_{e1}}{W_{ei}} \right) \quad (3.101)$$

e  $\bar{p}_2$  é a média das pressões no contato no intervalo de tempo  $\Delta t_2$ .

Para um intervalo de tempo  $\Delta t_n$ ,

$$\Delta W_{en} = \frac{W_{ei}}{p_i} (\bar{p}_{an-1} - \bar{p}_n) \left[ 1 - \exp \left( -\frac{J p_i}{W_{ei}} \Delta t_n \right) \right] \quad (3.102)$$

onde:

$$\bar{p}_{an-1} = p_i \left( 1 - \frac{1}{W_{ei}} \sum_{j=1}^{n-1} \Delta W_{ej} \right) = p_i \left( 1 - \frac{W_{en-1}}{W_{ei}} \right) \quad (3.103)$$

e

$$\bar{p}_n = \frac{p_{n-1} + p_n}{2} \quad (3.104)$$

Fetkovich mostrou para diferentes geometrias que o seu método produz resultados semelhantes aos do modelo de Van Everdingen e Hurst para aquíferos finitos.

### c) Observação 3

Ao utilizar o índice de produtividade do aquífero,  $J$ , para fluxo permanente, admite-se que o aquífero seja realimentado de modo que a pressão no seu limite externo se mantém constante e igual a  $p_i$ . A condição de fluxo permanente implica que não há limite para o influxo máximo, isto é,  $W_{ei}$  é infinito. Neste caso, a vazão do aquífero, eq. 3.95, reduz-se a:

$$q \equiv \frac{dW_e}{dt} = J(p_i - p) \quad (3.105)$$

cuja integral é o influxo acumulado:

$$W_e = J \int_0^t (p_i - p) dt \quad (3.106)$$

A eq. 3.106 é um caso particular do modelo de Fetkovich e foi apresentada por Schilthuis [Schilthuis, 1936].

#### d) Observação 4

A tabela 3.1 apresenta o índice de produtividade do aquífero,  $J$ , para os modelos de aquíferos radial e linear, regimes de fluxo permanente e pseudopermanente.

Tabela 3.1: Índice de produtividade do aquífero para os fluxos radial e linear. Fonte: [Rosa, 2011]

Condição de fluxo	Aquífero radial	Aquífero linear
Pseudopermanente	$J = \frac{2\pi fkh}{\mu \left[ \frac{r_e^2}{r_o^2} \ln\left(\frac{r_e}{r_o}\right) - \frac{3r_e^4 - 4r_o^4}{4(r_e^2 - 1)^2} \ln r_e \right]}$	$J = \frac{3khw}{\mu L}$
Permanente	$J = \frac{2\pi fkh}{\mu \ln\left(\frac{r_e}{r_o}\right)}$	$J = \frac{khw}{\mu L}$

Para outras geometrias, o índice de produtividade para o regime pseudopermanente pode ser definido como:

$$J = \frac{2\pi kh}{\frac{\mu}{2} \ln\left(\frac{4A}{\gamma C_A r_o^2}\right)} \quad (3.107)$$

onde:

$C_A$ : fator de forma [Dietz, 1965];

$A$ : área do aquífero;

$\gamma$ : exponencial da constante de Euler (  $\gamma = 1,78108$  );

$r_o$ : raio do reservatório circularizado.

O tempo adimensional  $t_{DA}$  é definido como:

$$t_{DA} = \frac{kt}{\phi \mu c_t A} \quad (3.108)$$

Quando  $r_e \gg r_o$  a equação do índice de produtividade para fluxo radial pseudopermanente, mostrada na 3.1, simplifica-se para:

$$J = \frac{2\pi fkh}{\mu \left[ \ln\left(\frac{r_e}{r_o}\right) - \frac{3}{4} \right]} \quad (3.109)$$

Um ponto a se destacar do modelo de Fetkovich é a sua facilidade de aplicação. Com isso, este modelo é frequentemente utilizado em modelos de simulação numérica apresentando resultados similares aos de Van Everdingen & Hurst [Ahmed, 2006].

### 3.3 Identificação de pacotes – assuntos

Os pacotes identificados são:

**Reservatório:** Esse pacote recebe os dados do usuário e/ou faz a leitura dos dados carregados no disco. E alguns desses dados são: raio, espessura, porosidade, permeabilidade, viscosidade, compressibilidade e pressão. Com todos esses parâmetros em consonância, se faz possível o cálculo do influxo de água pelo software.

**CSolverVanEverdingen e CSolverInfluxo:** Estes itens possuem os modelos que objetivam o estudo e fazem com que seus dados realizem a previsão do influxo de água em cada reservatório, em cada modelo de aquífero, ou seja, as classes que descrevem o modelo de Van Everdingen & Hurst, Carter-Tracy e Fetkovitch. Tais classes estão separadas nos componentes para os reservatórios que esses modelos são apropriados.

**Modelagem Numérica Computacional:** Contém os algoritmos matemáticos necessários para a solução dos diferentes modelos de aquíferos. Este pacote contém os algoritmos: Transformada de Laplace, Algoritmo de Inversão Numérica de Stehfest e Equação de Bessel Modificada.

**Gráfico:** Onde está inserida a biblioteca do gnuplot, necessária para a geração dos gráficos dos parâmetros do reservatório para o cálculo de influxo de água. Sabe-se que existe uma demanda por programa que gere gráficos, sendo assim, este software implementará a saída gráfica dos parâmetros calculados.

**Biblioteca:** Dentre as bibliotecas utilizadas, estarão as bibliotecas padrão de C++ (STL) e bibliotecas como a iostream, iomanip, etc.

### 3.4 Diagrama de pacotes – assuntos

O diagrama de pacotes da Figura 3.9 mostra as relações existentes entre os pacotes deste software.

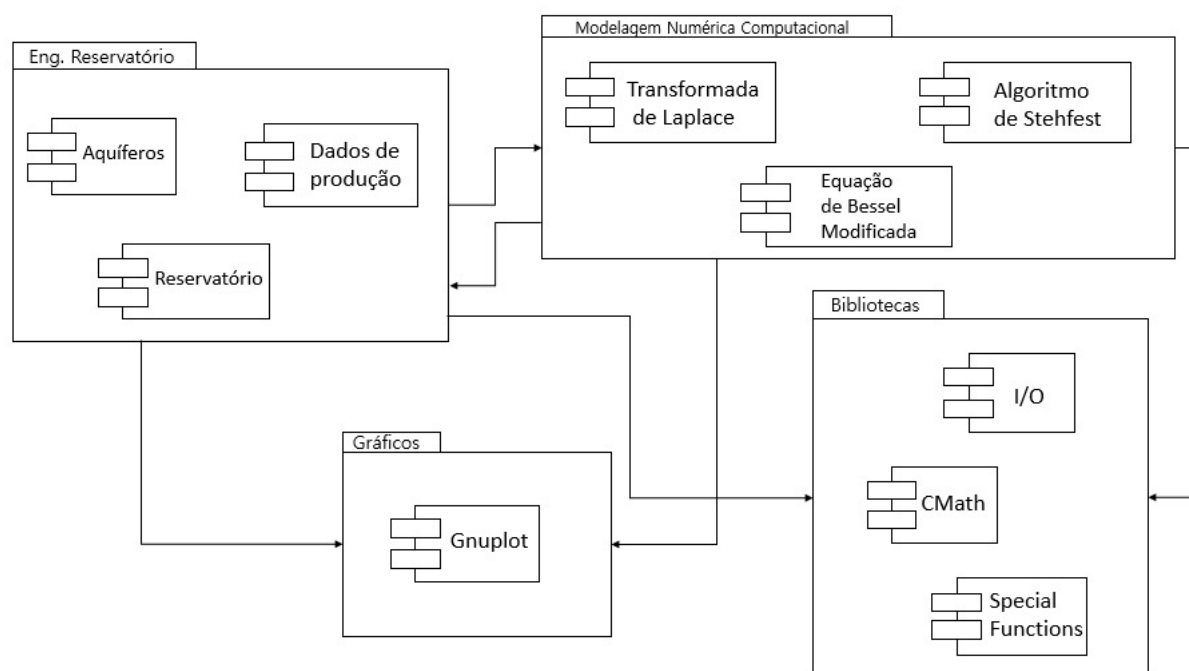


Figura 3.9: Diagrama de pacotes

# Capítulo 4

## AOO – Análise Orientada a Objeto

A terceira etapa do desenvolvimento de um projeto de engenharia, é a AOO – Análise Orientada a Objeto. Ela utiliza algumas regras para identificar os objetos de interesse, as relações entre os pacotes, as classes, os atributos, os métodos, as heranças, as associações, as agregações, as composições e as dependências. O resultado da análise é um conjunto de diagramas que identificam os objetos e seus relacionamentos.

### 4.1 Diagramas de classes

Os diagramas de classes estão entre os tipos mais úteis de diagramas UML, pois mapeiam de forma clara a estrutura de um determinado sistema ao modelar suas classes, seus atributos, operações e relações entre objetos. Esta definição fica evidenciada nas Figuras 4.1 e 4.2.

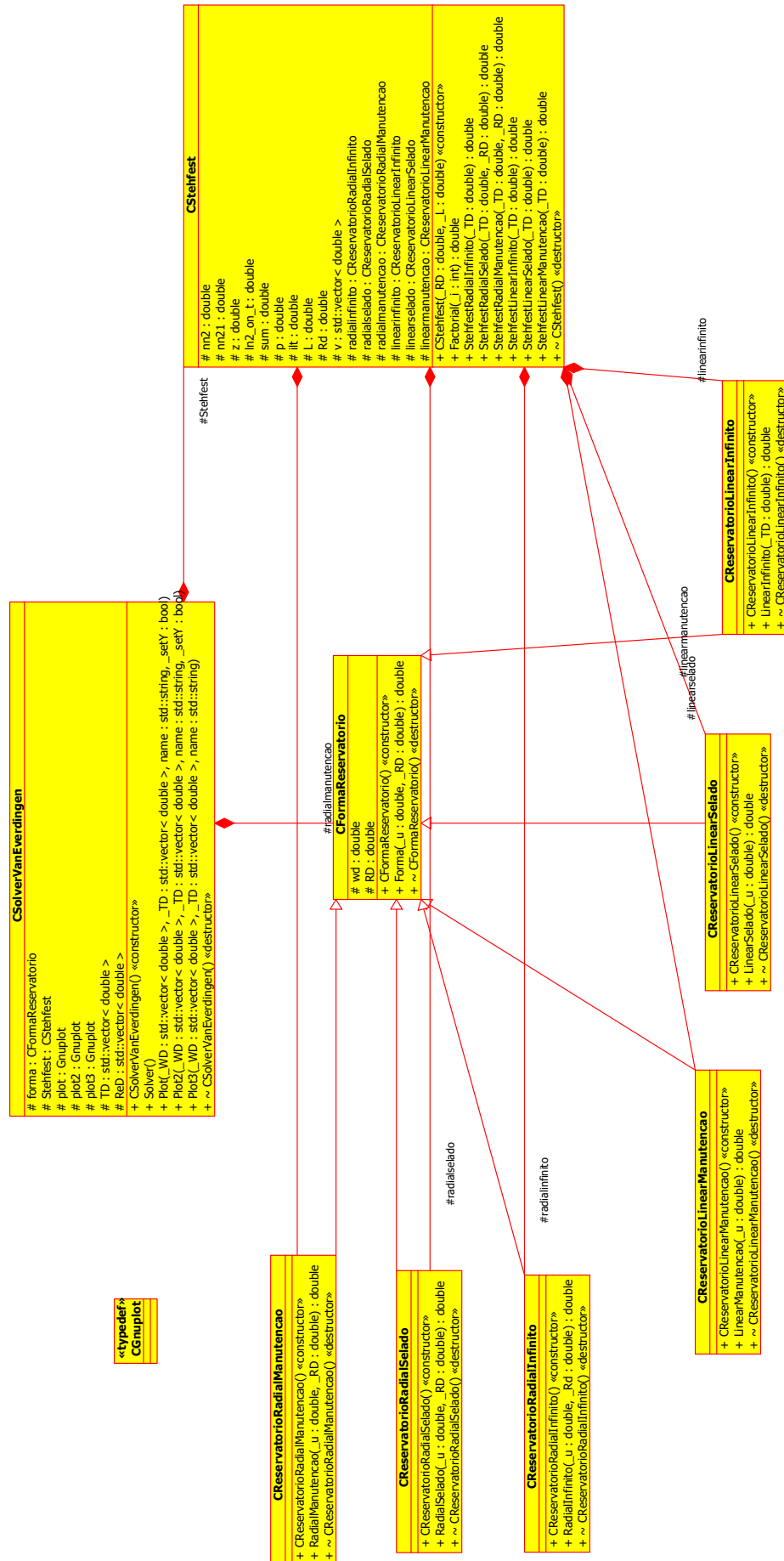


Figura 4.1: Diagrama de classes - Van Everdingen



Figura 4.2: Diagrama de classes - Carter-Tracy e Fetkovich



### 4.1.1 Dicionário de classes

- CAdimensional: Classe que representa as propriedades adimensionais nos problemas.
- CCarterTracy: Classe que resolve o problema para o Modelo de Carter-Tracy.
- CFetkovich: Classe herdeira da CAdimensional que resolve o problema para o Modelo Aproximado de Fetkovich.
- CFluido: Classe que representa um fluido e algumas de suas propriedades básicas: viscosidade e compressibilidade.
- CFormaReservatorio: Classe abstrata que representa a forma do reservatório. Ela é herdada pelas classes CReservatorioLinearInfinito, CReservatorioLinearManutencao, CReservatorioLinearSelado e CReservatorioRadialInfinito, CReservatorioRadialManutencao e CReservatorioRadialSelado.
- CGnuplot: Classe que fornece os métodos necessários para a geração de gráficos.
- CPoco: Classe que representa um poço e suas propriedades.
- CReservatorio: Classe que representa uma rocha reservatório e possui os atributos específicos do mesmo, como porosidade e permeabilidade.
- CReservatorioLinearInfinito: Classe herdeira de CFormaReservatorio, representa uma geometria linear, com largura e comprimento, do reservatório infinito.
- CReservatorioLinearManutencao: Classe herdeira de CFormaReservatorio, representa uma geometria linear, com largura e comprimento, do reservatório realimentado.
- CReservatorioLinearSelado: Classe herdeira de CFormaReservatorio, representa uma geometria linear, com largura e comprimento, do reservatório selado.
- CReservatorioRadialInfinito: Classe herdeira de CFormaReservatorio, representa uma geometria radial, com um raio, do reservatório infinito.
- CReservatorioRadialManutencao: Classe herdeira de CFormaReservatorio, representa uma geometria radial, com um raio, do reservatório realimentado.
- CReservatorioRadialSelado: Classe herdeira de CFormaReservatorio, representa uma geometria radial, com um raio, do reservatório selado.
- CRocha: Classe que representa uma rocha, possuindo os atributos pressão, porosidade e espessura média.

- CSolverInfluxo: Classe-mãe com os atributos necessários para a simulação de reservatórios com influxo de água, para os modelos de Carter Tracy e Fetkovich. Possui os objetos CCarterTracy, CFetkovich, CReservatorio, CFluido, CPoco, CRocha, CAdimensional e CGunplot.
- CSolverVanEverdingen: Classe-mãe com os atributos necessários para a simulação de reservatórios com influxo de água, para o modelo de Van Everdingen. Possui os objetos CFormaReservatorio, CGnuplot e CStehfest.
- CStehfest: Classe que realiza a inversão numérica dos influxos adimensionais no Campo de Laplace. Possui os objetos CReservatorioLinearInfinito, CReservatorioLinearManutencao, CReservatorioLinearSelado, CReservatorioRadialInfinito, CReservatorioRadialManutencao e CReservatorioRadialSelado.

## 4.2 Diagrama de sequência – eventos e mensagens

O diagrama de sequência salienta a troca de eventos e mensagens e sua ordem no tempo. O mesmo é uma fração do modelo dinâmico da análise orientada a objeto.

### 4.2.1 Diagrama de sequência geral

As Figuras 4.3 e 4.4 mostram os diagramas de sequência para a resolução de um problema de influxo de água.

No diagrama de sequências da Figura 4.3, o usuário cria uma classe CFormaReservatorio que contém os objetos CReservatorioRadial e CReservatorioLinear, onde essas duas classes representam os 3 reservatórios radiais e os 3 reservatórios lineares, e poderá carregar ou fornecer os dados necessários. Então é criada uma classe mãe CSolverVanEverdingen, que possui agregado os objetos CGnuplot e CFormaReservatorio.

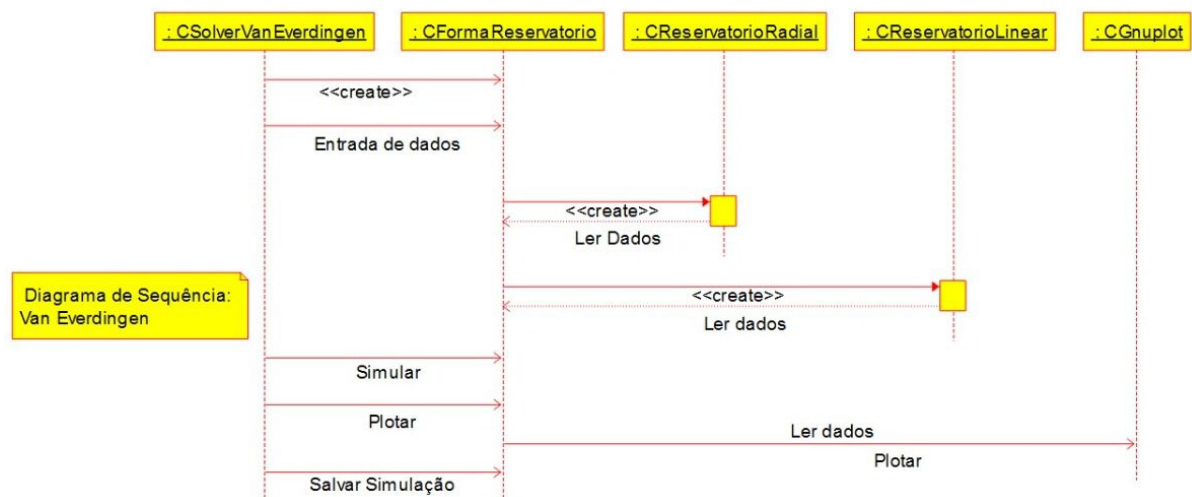


Figura 4.3: Diagrama de sequência - Van Everdingen

No diagrama de seqüências da Figura 4.4, o usuário cria uma classe mãe CSolverInfluxo com os atributos necessários para a simulação de reservatórios com influxo de água, para os modelos de Carter Tracy e Fetkovich. Possui os objetos CCarterTracy, CFetkovich, CReservatorio, CFluido, CPoco, CRocha, CAdimensional e CGunplot.

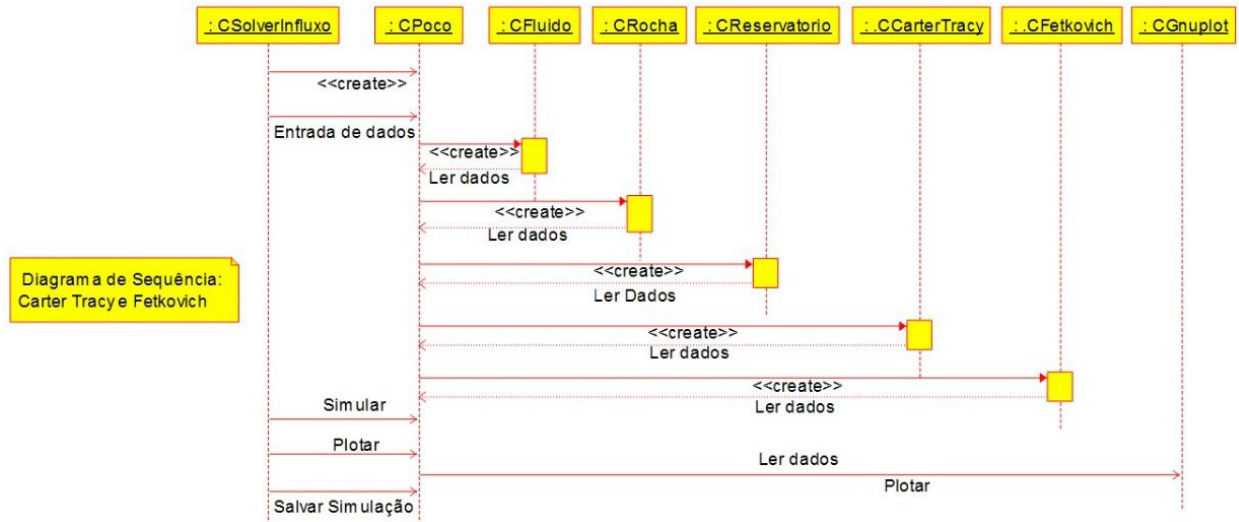


Figura 4.4: Diagrama de seqüência - Carter-Tracy e Fetkovich

### 4.3 Diagrama de comunicação – colaboração

Como o nome já sugere, o diagrama de comunicação é voltado para a interatividade entre os objetos, assim como a troca de dados e mensagens entre os mesmos.

As Figuras 4.5 e 4.6 representam os diagramas de comunicação das classes CSolverVanEverdingen e CSolverInfluxo respectivamente.

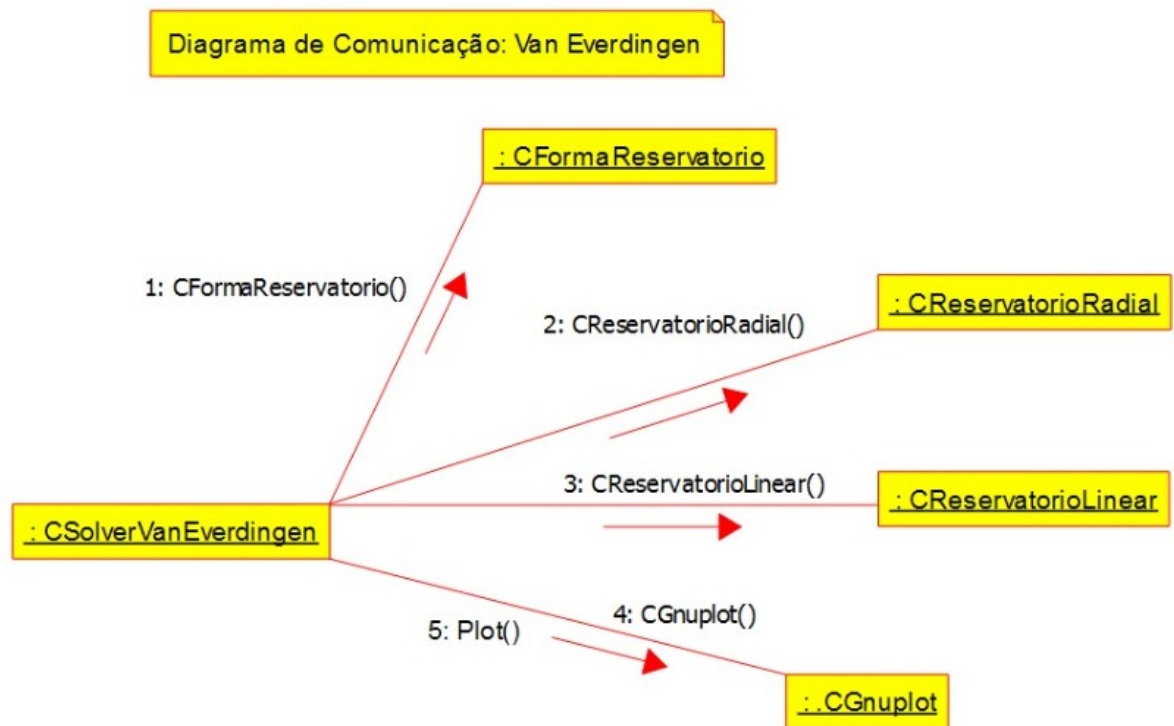


Figura 4.5: Diagrama de comunicação - Van Everdingen

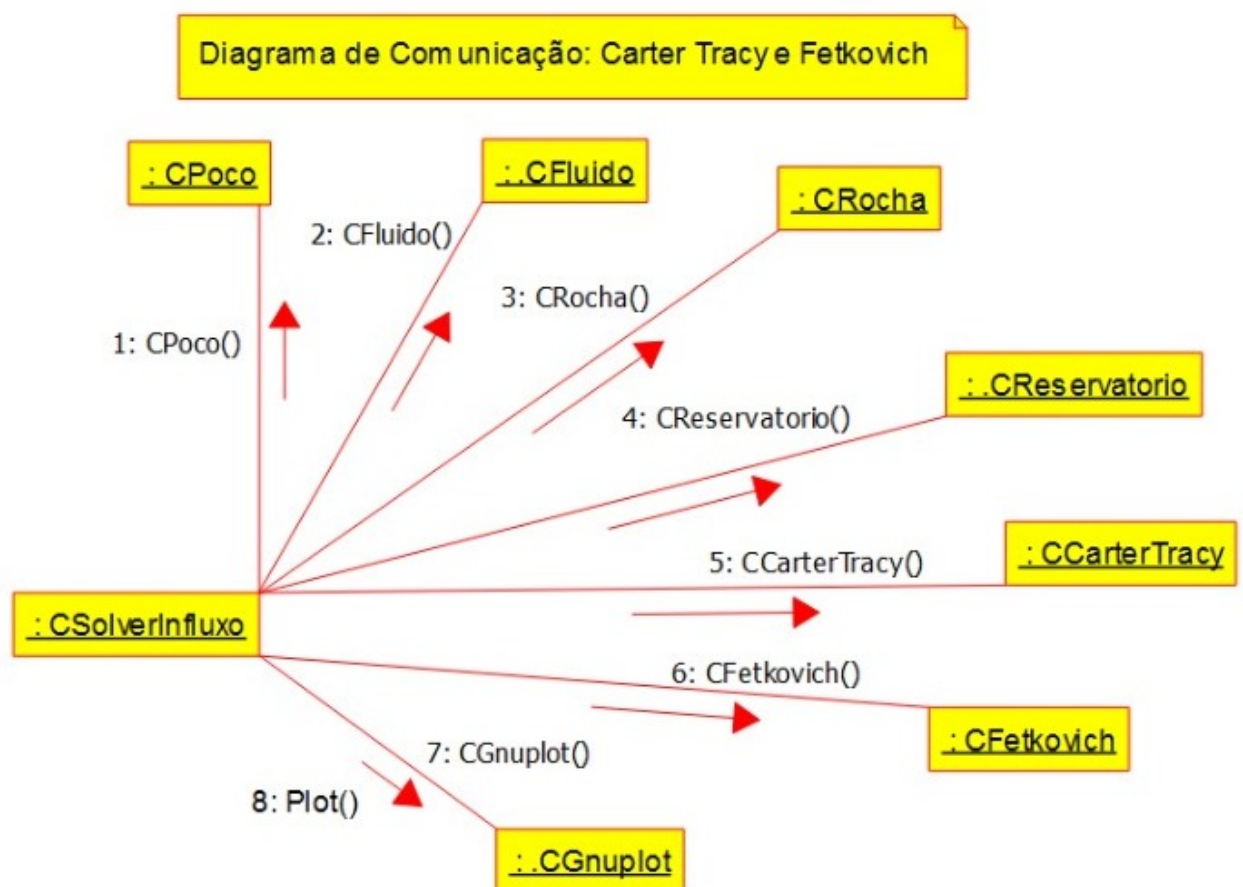


Figura 4.6: Diagrama de comunicação - Carter-Tracy e Fetkovich

## 4.4 Diagrama de máquina de estado

Todo objeto tem um tempo de vida. Entre sua criação e destruição o objeto está em operação, enviando e respondendo mensagens. Quando seu comportamento for variável ao longo do tempo, é útil especificá-lo por meio de uma máquina de estados.

O diagrama de máquina de estado demonstra o comportamento de um elemento através de um conjunto de transições de estado.

Um estado é a situação em que um objeto se encontra num determinado momento. Quando participa de um processo, portanto, ele pode demonstrar a espera pela ocorrência de um evento, a reação a um estímulo, a execução de alguma atividade ou a satisfação de alguma condição, o que ocasiona uma mudança no estado do sistema e de alguns objetos.

As Figuras 4.7 e 4.8 representam este diagrama para as classes CSolverVanEverdingen e CSolverInfluxo respectivamente.

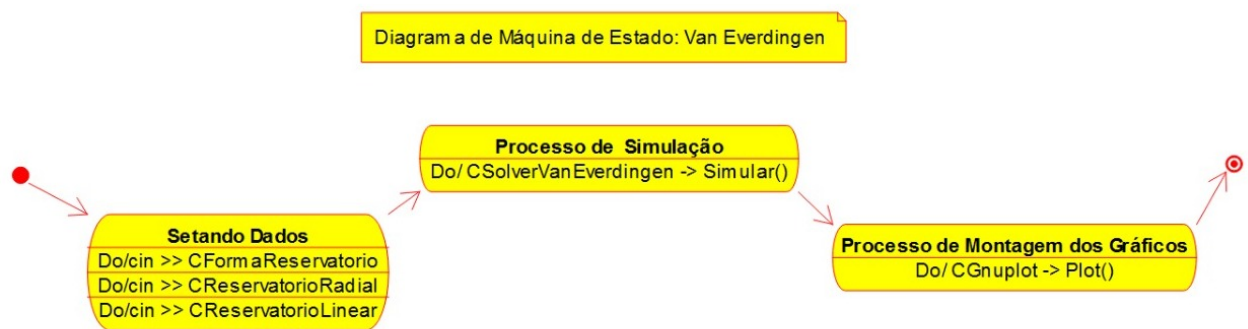


Figura 4.7: Diagrama de máquina de estado - Van Everdingen

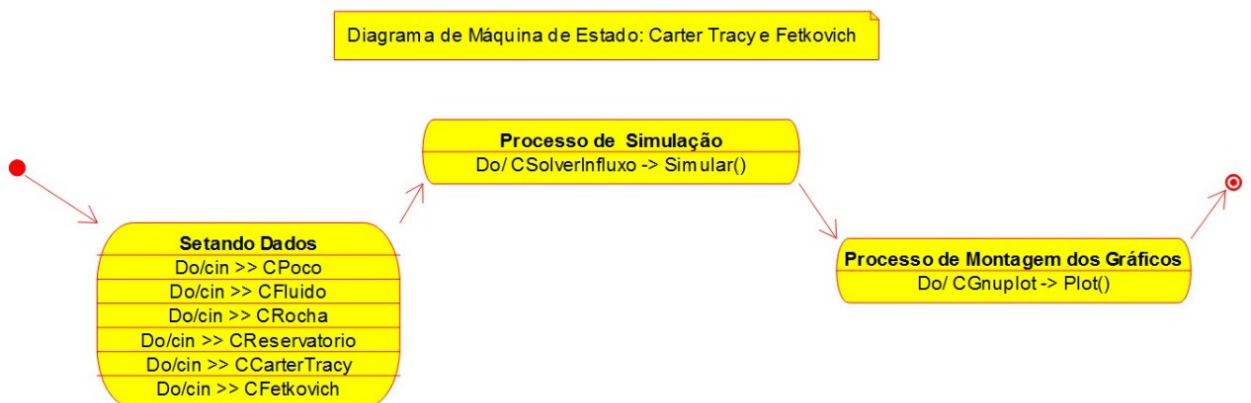


Figura 4.8: Diagrama de máquina de estado - Carter-Tracy e Fetkovich

## 4.5 Diagrama de atividades

O diagrama de atividades fornece uma visualização do comportamento de um sistema descrevendo a sequência de ações em um processo. Os diagramas de atividades são se-

melhantes a fluxogramas porque mostram o fluxo entre as ações em uma atividade. No entanto, os diagramas de atividades também podem mostrar fluxos paralelos ou simultâneos e fluxos alternativos.

O diagrama de atividades corresponde a uma atividade específica do diagrama de máquina de estado e nesta seção esses diagramas são representados pela Figuras 4.9 e 4.10 .

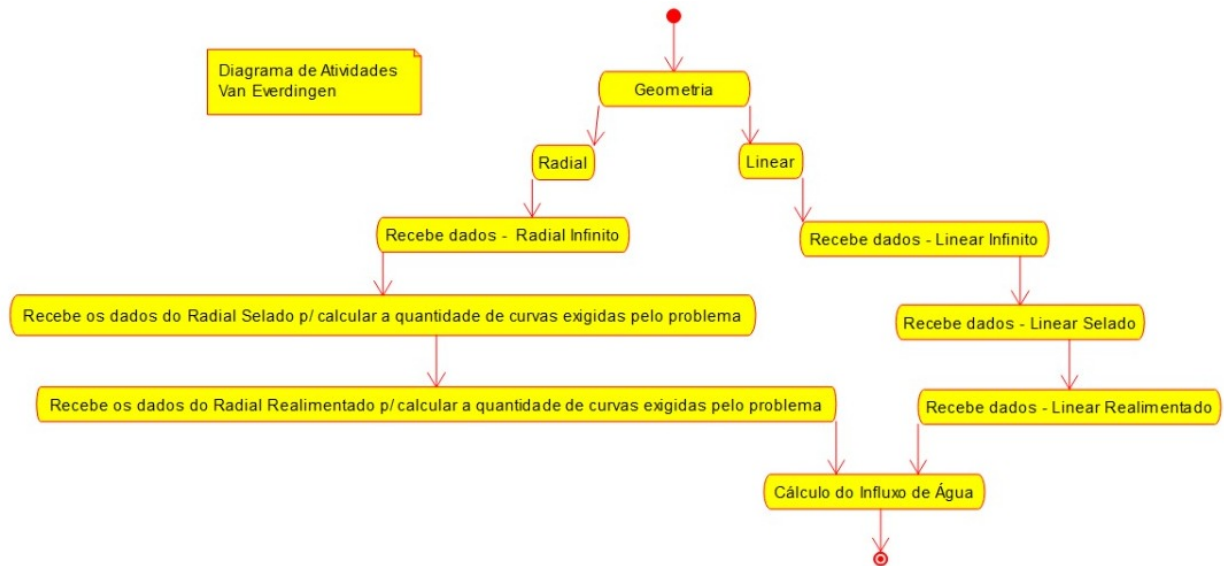


Figura 4.9: Diagrama de atividades - Van Everdingen

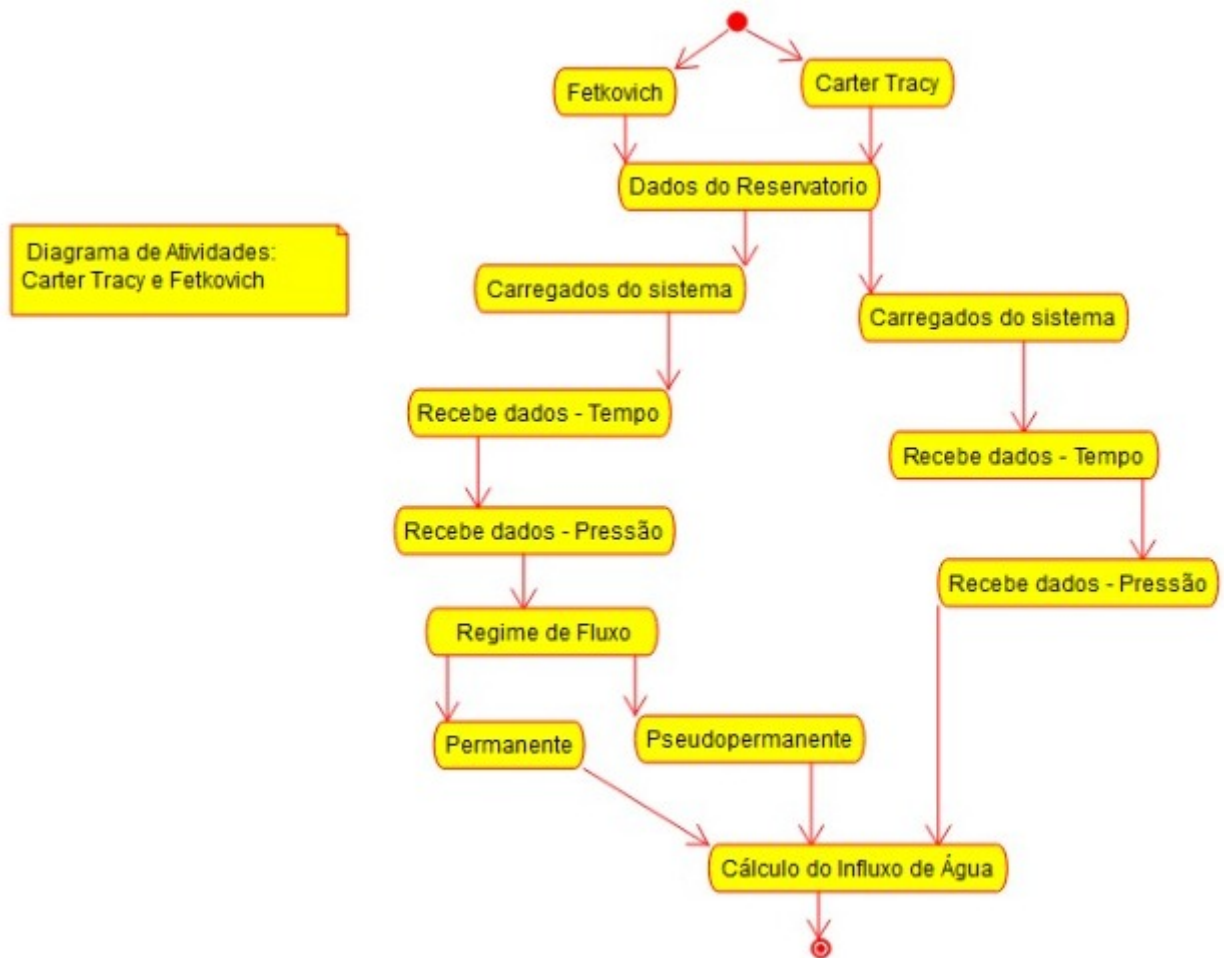


Figura 4.10: Diagrama de atividades - Carter-Tracy e Fetkovich

# Capítulo 5

## Projeto

Neste capítulo do projeto de engenharia são vistas questões associadas ao projeto do sistema, incluindo protocolos, recursos, plataformas suportadas, implicações nos diagramas feitos anteriormente, diagramas de componentes e implantação. Na segunda parte são revisados 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.

Foram definidos 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.

Segundo [Rumbaugh et al., 1994, Blaha and Rumbaugh, 2006], o projeto do sistema é a estratégia de alto nível para resolver o problema e elaborar uma solução. Deve-se atentar aos seguintes itens:

#### 1. Protocolos

- O programa utilizará biblioteca padrão C++ e o Gnuplot.
- Os arquivos de texto com os resultados da simulação terão o formato .dat ou .txt.
- Será utilizada uma biblioteca GSL para cálculos matemáticos especiais chamada special functions da GNU.



- O programa utilizará uma máquina computacional com HD, processador, teclado (para a entrada de dados e monitor (para a saída de dados). Os arquivos gerados pelo programa estarão em formato de texto em um banco de dados.

## 2. Recursos

- O simulador utiliza como recurso para plotagem de gráficos o programa externo Gnuplot, e, também faz uso do HD, CPU, RAM e periféricos.

## 3. Controle

- Este software possui um controle sequencial.

## 4. Plataformas

- O software é multi-plataforma, logo, permite performar tanto em Windows quanto em GNU-Linux.
- Linguagem de software utilizada: C++ orientada a objeto.
- Serão utilizadas as seguintes bibliotecas: gnuplot e GSL special functions.

## 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 software). Passa pelo maior detalhamento do funcionamento do software, acrescentando atributos e métodos que envolvem a solução de problemas específicos não identificados durante a análise.

Ademais, a POO também engloba a otimização da estrutura de dados e dos algoritmos, a minimização do tempo de execução, de memória e de custos.

### Efeitos do projeto no modelo estrutural

- Se faz necessária a instalação do programa Gnuplot na máquina que irá performar o software.

**Efeitos do projeto no modelo dinâmico****Efeitos do projeto nos atributos****Efeitos do projeto nos métodos****Efeitos do projeto nas heranças****Efeitos do projeto nas associações****Efeitos do projeto nas otimizações****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.

As Figuras 5.1 e 5.2 retratam o diagrama de componentes. Com eles, têm-se ciência dos componentes fundamentais para a performance do software.

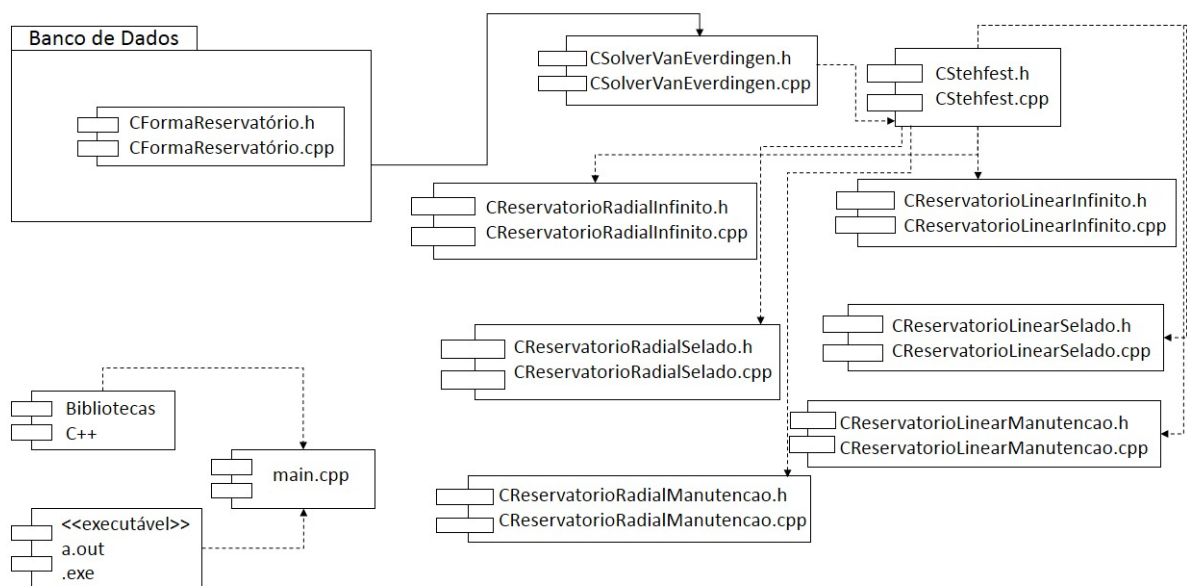


Figura 5.1: Diagrama de componentes do modelo de Van Everdingen

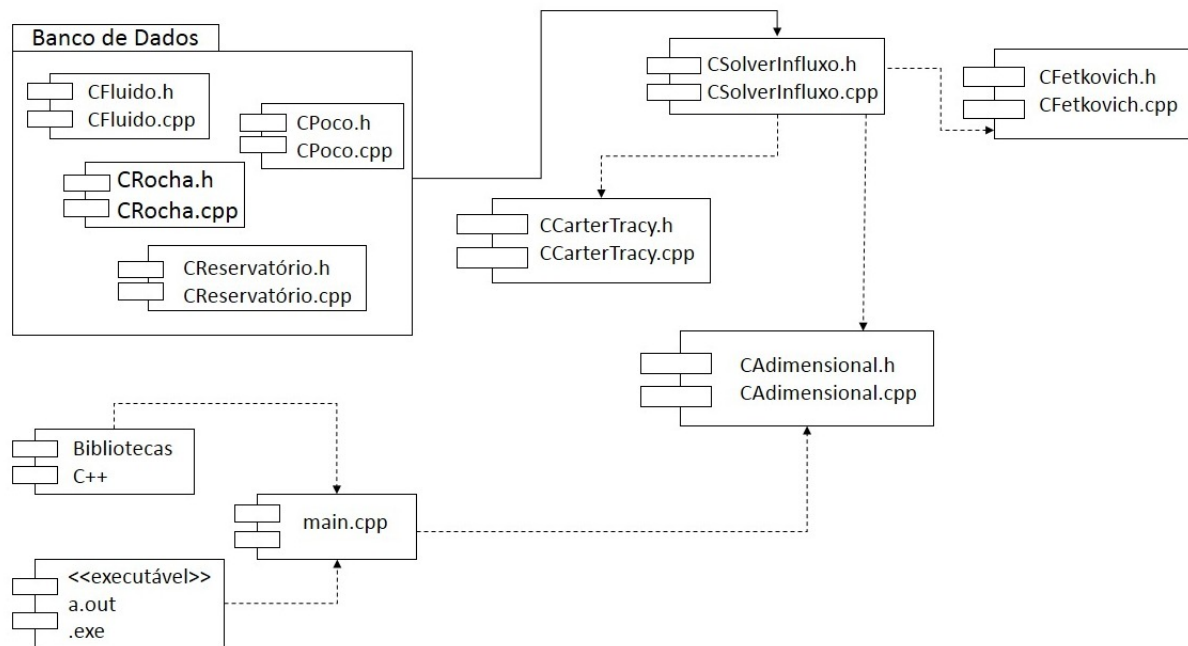


Figura 5.2: Diagrama de componentes dos modelos de Carter-Tracy e Fetkovich

## 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.

A Figura 5.3 reproduz o diagrama de implantação deste software.

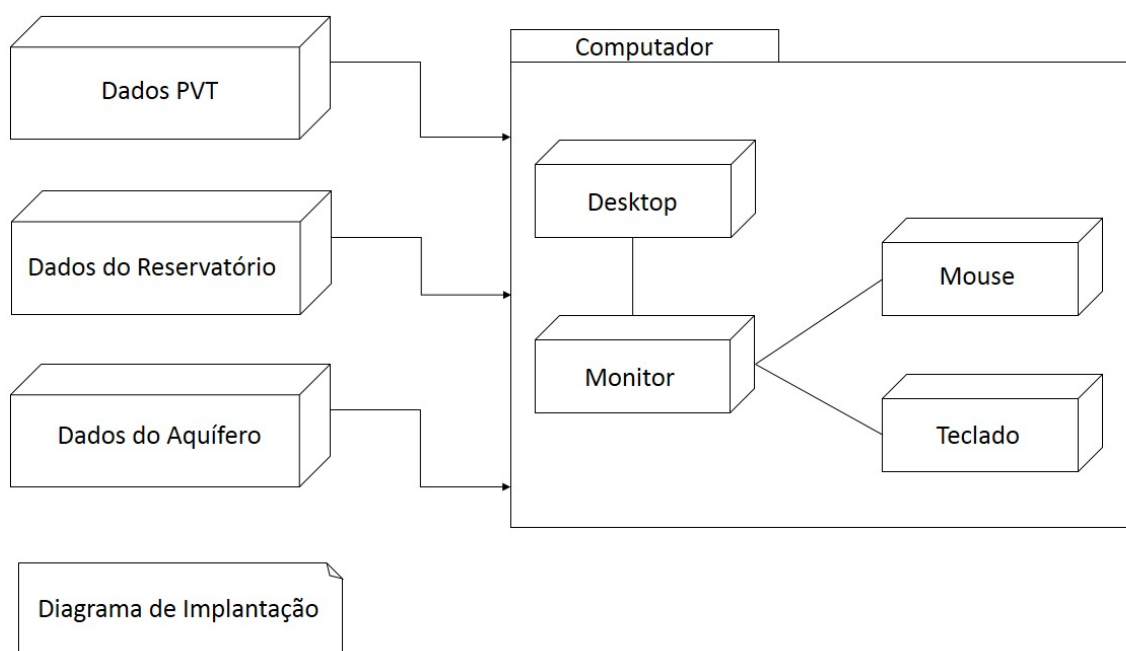


Figura 5.3: Diagrama de implantação

# 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` dos softwares.

Apresenta-se na listagem 6.1 o arquivo com código da classe `CFormaReservatorio`.

Listing 6.1: Arquivo de cabeçalho da classe `CFormaReservatorio`.

---

```
1 #ifndef CFORMARESERVATORIO_H_
2 #define CFORMARESERVATORIO_H_
3
4 class CFormaReservatorio
5 {
6     protected:
7
8         double wd, RD; //Influxo Adimensional //Raio
9                             externo adimensional
10
11     public:
12
13         CFormaReservatorio(){};
14
15         virtual double Forma(double _u, double _RD);
16
17         ~CFormaReservatorio(){};
18 };
```

19

20 `#endif`

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

Listing 6.2: Arquivo de implementação da classe CFormaReservatorio.

---

```
1 #include "CFormaReservatorio.h"
2
3 double CFormaReservatorio::Forma(double _u, double _RD)
4 {
5     return wd;
6 }
```

---

Apresenta-se na listagem 6.30 o arquivo com código da classe CGnuplot.

Listing 6.3: Arquivo de cabeçalho da classe CGnuplot.

---

```
1 //
2 //
3 //
4 //
5 //
6 //
7 //
8 //
9 //
10 //
11 //
12 //
13 //
14 //
15 //
16 //
17 //
18 //
```

---

```

19//      por Bueno.A.D. (30/07/08)
20//      Tarefas:
21//      (v1)
22//      Documentar toda classe
23//      Adicionar novos métodos, criando atributos adicionais se
        necessario.
24//      Adotar padrao C++, isto e, usar sobrecarga nas chamadas.
25//      (v2)
26//      Criar classe herdeira CGnuplot, que inclui somente a nova
        interface.
27//      como e herdeira, o usuario vai poder usar nome antigos.
28//      Vantagem: preserva classe original, cria nova interface,
        fica a critério do usuário
29//      qual interface utilizar.
30//
        //////////////////////////////////////
31// Requisitos:
32// - O programa gnuplot deve estar instalado (veja http://www.gnuplot.info/download.html)
33// - No Windows: setar a Path do Gnuplot (i.e. C:/program files/
        gnuplot/bin)
34//      ou setar a path usando: Gnuplot::set_GNUPlotPath(
        const std::string &path);
35//      Gnuplot::set_GNUPlotPath("C:/program files/gnuplot
        /bin");
36// - Para um melhor uso, consulte o manual do gnuplot,
37//      no GNU/Linux digite: man gnuplot ou info gnuplot.
38//
39// - Veja aula em http://www.lenep.uenf.br/~bueno/DisciplinaSL/
40//
41//
        //////////////////////////////////////
42
43
44#ifdef CGnuplot_h
45#define CGnuplot_h
46#include <iostream>                // Para teste
47#include <string>
48#include <vector>
49#include <stdexcept>              // Heranca da classe std::

```

```

        runtime_error em GnuplotException
50 #include <cstdio>                                // Para acesso a arquivos FILE
51
52 /**
53 @brief Erros em tempo de execucao
54 @class GnuplotException
55 @file GnuplotException.h
56 */
57 class GnuplotException : public std::runtime_error
58 {
59 public:
60     /// Construtor
61     GnuplotException (const std::string & msg):std::runtime_error (
        msg) {}
62 };
63
64 /**
65 @brief Classe de interface para acesso ao programa gnuplot.
66 @class Gnuplot
67 @file gnuplot_i.hpp
68 */
69 class Gnuplot
70 {
71 private:
72     //
    -----
    Atributos
73     FILE *   gnucmd;          ///< Ponteiro para stream que escreve no
        pipe.
74     bool     valid;           ///< Flag que indica se a sessao do
        gnuplot esta valida.
75     bool     two_dim;         ///< true = verdadeiro = 2d, false =
        falso = 3d.
76     int      nplots;          ///< Numero de graficos (plots) na sessao
        .
77     std::string pstyle;       ///< Estilo utilizado para visualizacao
        das funcoes e dados.
78     std::string smooth;       ///< interpolate and approximate data in
        defined styles (e.g. spline).
79     std::vector<std::string> tmpfile_list; ///< Lista com nome dos
        arquivos temporarios.
80

```



```

81  //
    -----

    flags
82  bool fgrid;           ///< 0 sem grid,          1 com grid
83  bool fhidden3d;       ///< 0 nao oculta,          1 oculta
84  bool fcontour;        ///< 0 sem contorno,        1 com contorno
85  bool fsurface;        ///< 0 sem superficie,      1 com superficie
86  bool flegend;         ///< 0 sem legendad,        1 com legenda
87  bool ftitle;          ///< 0 sem titulo,          1 com titulo
88  bool fxlogscale;       ///< 0 desativa escala log, 1 ativa
    escala log
89  bool fylogscale;       ///< 0 desativa escala log, 1 ativa
    escala log
90  bool fzlogscale;       ///< 0 desativa escala log, 1 ativa
    escala log
91  bool fsmooth;          ///< 0 desativa,            1 ativa
92
93  //
    -----

94  // Atributos estaticos (compartilhados por todos os objetos)
95  static int tmpfile_num;           ///< Numero total de
    arquivos temporarios (numero restrito).
96  static std::string m_sGNUPlotFileName; ///< Nome do arquivo
    executavel do gnuplot.
97  static std::string m_sGNUPlotPath;   ///< Caminho para
    executavel do gnuplot.
98  static std::string terminal_std;     ///< Terminal padrao (
    standart), usado para visualizacoes.
99
100 //
    -----

    Metodos
101 // Funcoes membro (métodos membro) (funcoes auxiliares)
102 /// @brief Cria arquivo temporario e retorna seu nome.
103 /// Usa get_program_path(); e popen();
104 void init ();
105
106 /// @brief Cria arquivo temporario e retorna seu nome.
107 /// Usa get_program_path(); e popen();
108 void Init() { init(); }
109

```

```

110  /// @brief Cria arquivo temporario.
111  std::string create_tmpfile (std::ofstream & tmp);
112
113  /// @brief Cria arquivo temporario.
114  std::string CreateTmpFile (std::ofstream & tmp) { return
        create_tmpfile(tmp); }
115
116  //
        -----
117  // Funcoes estaticas (static functions)
118  /// @brief Retorna verdadeiro se a path esta presente.
119  static bool get_program_path ();
120
121  /// @brief Retorna verdadeiro se a path esta presente.
122  static bool Path() { return get_program_path(); }
123
124  /// @brief Checa se o arquivo existe.
125  static bool file_exists (const std::string & filename, int
        mode = 0);
126
127  /// @brief Checa se o arquivo existe.
128  static bool FileExists (const std::string & filename, int
        mode = 0)
129
        { return file_exists( filename, mode
        ); }
130
131  //
        -----
132 public:
133  // Opcional: Seta path do gnuplot manualmente
134  // No windows: a path (caminho) deve ser dada usando '/' e nao
        backslash '\'
135  /// @brief Seta caminho para path do gnuplot.
136  //ex: CGnuplot::set_GNUPlotPath ("\"C:/program files/gnuplot/
        bin/\"");
137
138  static bool set_GNUPlotPath (const std::string & path);
139
140  /// @brief Seta caminho para path do gnuplot.
141  static bool Path(const std::string & path) { return

```

```

        set_GNUPlotPath(path); }
142 //
143 /// @brief Opcional: Seta terminal padrao (standart), usado
        para visualizacao dos graficos.
144 /// Valores padroes (default): Windows - win, Linux - x11, Mac
        - aqua
145 static void set_terminal_std (const std::string & type);
146
147 /// @brief Opcional: Seta terminal padrao (standart), usado
        para visualizacao dos graficos.
148 /// Para retornar para terminal janela precisa chamar
        ShowOnScreen().
149 /// Valores padroes (default): Windows - win, Linux - x11 ou
        wxt (fedora9), Mac - aqua
150 static void Terminal (const std::string & type) {
        set_terminal_std(type); }
151
152 //
        -----
        Construtores
153 /// @brief Construtor, seta o estilo do grafico na construcao.
154 Gnuplot (const std::string & style = "points");
155
156 /// @brief Construtor, plota um grafico a partir de um vector,
        diretamente na construcao.
157 Gnuplot (const std::vector < double >&x,
158          const std::string & title = "",
159          const std::string & style = "points",
160          const std::string & labelx = "x",
161          const std::string & labely = "y");
162
163 /// @brief Construtor, plota um grafico do tipo x_y a partir de
        vetores, diretamente na construcao.
164 Gnuplot (const std::vector < double >&x,
165          const std::vector < double >&y,
166          const std::string & title = "",
167          const std::string & style = "points",
168          const std::string & labelx = "x",
169          const std::string & labely = "y");
170
171 /// @brief Construtor, plota um grafico de x_y_z a partir de
        vetores, diretamente na construcao.

```

```

172 Gnuplot (const std::vector < double >&x,
173          const std::vector < double >&y,
174          const std::vector < double >&z,
175          const std::string & title = "",
176          const std::string & style = "points",
177          const std::string & labelx = "x",
178          const std::string & labely = "y",
179          const std::string & labelz = "z");
180
181 /// @brief Destrutor, necessario para deletar arquivos
182   temporarios.
183 ~Gnuplot ();
184 //
185 -----
186
187 /// @brief Envia comando para o gnuplot.
188 Gnuplot & cmd (const std::string & cmdstr);
189
190 /// @brief Envia comando para o gnuplot.
191 Gnuplot & Cmd (const std::string & cmdstr) { return cmd(
192   cmdstr); }
193
194 /// @brief Envia comando para o gnuplot.
195 Gnuplot & Command (const std::string & cmdstr) { return cmd(
196   cmdstr); }
197
198 /// @brief Sobrecarga operador <<, funciona como Comando.
199 Gnuplot & operator<< (const std::string & cmdstr);
200
201 //
202 -----
203
204 /// @brief Mostrar na tela ou escrever no arquivo, seta o tipo
205   de terminal para terminal_std.
206 Gnuplot & showonscreen (); // Janela de saida e
207   setada como default (win/x11/aqua)
208
209 /// @brief Mostrar na tela ou escrever no arquivo, seta o tipo
210   de terminal para terminal_std.
211 Gnuplot & ShowOnScreen () { return
212   showonscreen(); };

```

```
203
204 /// @brief Salva sessao do gnuplot para um arquivo postscript,
      informe o nome do arquivo sem extensao.
205 /// Depois retorna para modo terminal
206 Gnuplot & savetops (const std::string & filename = "
      gnuplot_output");
207
208 /// @brief Salva sessao do gnuplot para um arquivo postscript,
      informe o nome do arquivo sem extensao
209 /// Depois retorna para modo terminal
210 Gnuplot & SaveTops (const std::string & filename = "
      gnuplot_output")
211                                     { return
                                          savetops(
                                          filename); }
212
213 /// @brief Salva sessao do gnuplot para um arquivo png, nome do
      arquivo sem extensao
214 /// Depois retorna para modo terminal
215 Gnuplot & savetopng (const std::string & filename = "
      gnuplot_output");
216
217 /// @brief Salva sessao do gnuplot para um arquivo png, nome do
      arquivo sem extensao
218 /// Depois retorna para modo terminal
219 Gnuplot & SaveTopng (const std::string & filename = "
      gnuplot_output")
220                                     { return
                                          savetopng(
                                          filename); }
221
222 /// @brief Salva sessao do gnuplot para um arquivo jpg, nome do
      arquivo sem extensao
223 /// Depois retorna para modo terminal
224 Gnuplot & savetojpeg (const std::string & filename = "
      gnuplot_output");
225
226 /// @brief Salva sessao do gnuplot para um arquivo jpg, nome do
      arquivo sem extensao
227 /// Depois retorna para modo terminal
228 Gnuplot & SaveTojpeg (const std::string & filename = "
      gnuplot_output")
```

```

229                                     { return
                                         savetojpeg(
                                             filename); }

230
231 /// @brief Salva sessao do gnuplot para um arquivo filename,
        usando o terminal_type e algum flag adicional
232 /// Ex:
233 /// grafico.SaveTo("pressao_X_temperatura","png", "enhanced
        size 1280,960");
234 /// Para melhor uso dos flags adicionais consulte o manual do
        gnuplot (help term)
235 Gnuplot& SaveTo(const std::string &filename,const std::string &
        terminal_type, std::string flags="");
236
237 //
        -----
        set e unset
238 /// @brief Seta estilos de linhas (em alguns casos sao
        necessarias informacoes adicionais).
239 /// lines, points, linespoints, impulses, dots, steps, fsteps,
        histeps,
240 /// boxes, histograms, filledcurves
241 Gnuplot & set_style (const std::string & stylestr = "points");
242
243 /// @brief Seta estilos de linhas (em alguns casos sao
        necessarias informacoes adicionais).
244 /// lines, points, linespoints, impulses, dots, steps, fsteps,
        histeps,
245 /// boxes, histograms, filledcurves
246 Gnuplot & Style (const std::string & stylestr = "points")
247                                     { return
                                         set_style(
                                             stylestr); }
248
249 /// @brief Ativa suavizacao.
250 /// Argumentos para interpolacoes e aproximacoes.
251 /// csplines, bezier, acsplines (para dados com valor > 0),
        sbezier, unique,
252 /// frequency (funciona somente com plot_x, plot_xy, plotfile_x
        ,
253 /// plotfile_xy (se a suavizacao esta ativa, set_style nao tem
        efeito na plotagem dos graficos)

```

```

254 Gnuplot & set_smooth (const std::string & stylestr = "csplines
    ");
255
256 /// @brief Desativa suavizacao.
257 Gnuplot & unset_smooth (); // A suavizacao nao e
    setada por padrao (default)
258
259 /// @brief Ativa suavizacao.
260 /// Argumentos para interpolacoes e aproximacoes.
261 /// csplines, bezier, acsplines (para dados com valor > 0),
    sbezier, unique,
262 /// frequency (funciona somente com plot_x, plot_xy, plotfile_x
    ,
263 /// plotfile_xy (se a suavizacao esta ativa, set_style nao tem
    efeito na plotagem dos graficos)
264 Gnuplot & Smooth(const std::string & stylestr = "csplines")
265 { return
    set_smooth(
    stylestr); }
266
267 Gnuplot & Smooth( int _fsmooth )
268 { if( fsmooth =
    _fsmooth )
269     return
    set_contour
    ();
270     else
271     return
    unset_contour
    ();
272 }
273 /// @brief Desativa suavizacao.
274 ///Gnuplot & UnsetSmooth() { return
    unset_smooth (); }
275
276 /// @brief Escala o tamanho do ponto usado na plotagem.
277 Gnuplot & set_pointsize (const double pointsize = 1.0);
278
279 /// @brief Escala o tamanho do ponto usado na plotagem.
280 Gnuplot & PointSize (const double pointsize = 1.0)
281 { return
    set_pointsize(

```

```

pointsized); }

282
283 /// @brief Ativa o grid (padrao = desativado).
284 Gnuplot & set_grid ();
285
286 /// @brief Desativa o grid (padrao = desativado).
287 Gnuplot & unset_grid ();
288
289 /// @brief Ativa/Desativa o grid (padrao = desativado).
290 Gnuplot & Grid(bool _fgrid = 1)
291                                     { if(fgrid =
                                     _fgrid)
292                                         return
                                     set_grid();
293                                         else
294                                         return
                                     unset_grid
                                     (); }
295
296 /// @brief Seta taxa de amostragem das funcoes, ou dos dados de
interpolacao.
297 Gnuplot & set_samples (const int samples = 100);
298
299 /// @brief Seta taxa de amostragem das funcoes, ou dos dados de
interpolacao.
300 Gnuplot & Samples(const int samples = 100) { return
set_samples(samples); }
301
302 /// @brief Seta densidade de isolinhas para plotagem de funcoes
como superficies (para plotagen 3d).
303 Gnuplot & set_isosamples (const int isolines = 10);
304
305 /// @brief Seta densidade de isolinhas para plotagem de funcoes
como superficies (para plotagen 3d).
306 Gnuplot & IsoSamples (const int isolines = 10){ return
set_isosamples(isolines); }
307
308 /// @brief Ativa remocao de linhas ocultas na plotagem de
superficies (para plotagen 3d).
309 Gnuplot & set_hidden3d ();
310
311 /// @brief Desativa remocao de linhas ocultas na plotagem de

```



```

    superficies (para plotagen 3d).
312 Gnuplot & unset_hidden3d ();          // hidden3d nao e setado
    por padrao (default)

313
314 /// @brief Ativa/Desativa remocao de linhas ocultas na plotagem
    de superficies (para plotagen 3d).
315 Gnuplot & Hidden3d(bool _fhidden3d = 1)
316                                     { if(fhidden3d =
                                     _fhidden3d)
317                                     return
                                     set_hidden3d
                                     ();
318                                     else
319                                     return
                                     unset_hidden3d
                                     ();
320                                     }
321
322 /// @brief Ativa desenho do contorno em superficies (para
    plotagen 3d).
323 /// @param base, surface, both.
324 Gnuplot & set_contour (const std::string & position = "base");
325
326 /// @brief Desativa desenho do contorno em superficies (para
    plotagen 3d).
327 Gnuplot & unset_contour ();          // contour nao e setado
    por default
328
329 /// @brief Ativa/Desativa desenho do contorno em superficies (
    para plotagen 3d).
330 /// @param base, surface, both.
331 Gnuplot & Contour(const std::string & position = "base")
332                                     { return
                                     set_contour(
                                     position); }
333
334 Gnuplot & Contour( int _fcontour )
335                                     { if( fcontour =
                                     _fcontour )
336                                     return
                                     set_contour
                                     ();

```

```

337                                     else
338                                     return
                                     unset_contour
                                     ();
339                                     }
340 /// @brief Ativa a visualizacao da superficie (para plotagen 3
      d).
341 Gnuplot & set_surface ();           // surface e setado por
      padrao (default)
342
343 /// @brief Desativa a visualizacao da superficie (para
      plotagen 3d).
344 Gnuplot & unset_surface ();
345
346 /// @brief Ativa/Desativa a visualizacao da superficie (para
      plotagen 3d).
347 Gnuplot & Surface( int _fsurface = 1 )
348                                     { if(fsurface =
                                     _fsurface)
349                                     return
                                     set_surface
                                     ();
350                                     else
351                                     return
                                     unset_surface
                                     ();
352                                     }
353 /// @brief Ativa a legenda (a legenda é setada por padrao).
354 /// Posicao: inside/outside, left/center/right, top/center/
      bottom, nobox/box
355 Gnuplot & set_legend (const std::string & position = "default"
      );
356
357 /// @brief Desativa a legenda (a legenda é setada por padrao).
358 Gnuplot & unset_legend ();
359
360 /// @brief Ativa/Desativa a legenda (a legenda é setada por
      padrao).
361 Gnuplot & Legend(const std::string & position = "default")
362                                     { return
                                     set_legend(
                                     position); }

```

```

363
364 /// @brief Ativa/Desativa a legenda (a legenda é setada por
      padrao).
365 Gnuplot & Legend(int _flegend)
366
367                                     { if(flegend =
                                          _flegend)
                                          return
                                          set_legend
                                          ();
368                                     else
369                                     return
                                          unset_legend
                                          ();
370                                     }
371
372 /// @brief Ativa o titulo da secao do gnuplot.
373 Gnuplot & set_title (const std::string & title = "");
374
375 /// @brief Desativa o titulo da secao do gnuplot.
376 Gnuplot & unset_title ();          // 0 title nao e setado
      por padrao (default)
377
378 /// @brief Ativa/Desativa o titulo da secao do gnuplot.
379 Gnuplot & Title(const std::string & title = "")
380
381                                     {
382                                     return set_title
                                          (title);
383                                     }
384
385 Gnuplot & Title(int _ftitle)
386
387                                     {
388                                     if(ftitle =
                                          _ftitle)
389                                     return
                                          set_title();
390                                     else
391                                     return
                                          unset_title
                                          ();
392                                     }
393
394 /// @brief Seta o rotulo (nome) do eixo y.
395 Gnuplot & set_ylabel (const std::string & label = "y");

```

```

393
394 /// @brief Seta o rotulo (nome) do eixo y.
395 /// Ex: set ylabel "{/Symbol s}[MPa]" font "Times Italic, 10"
396 Gnuplot & YLabel(const std::string & label = "y")
397 { return
    set_ylabel(
    label); }

398
399 /// @brief Seta o rotulo (nome) do eixo x.
400 Gnuplot & set_xlabel (const std::string & label = "x");
401
402 /// @brief Seta o rotulo (nome) do eixo x.
403 Gnuplot & XLabel(const std::string & label = "x")
404 { return
    set_xlabel(
    label); }

405
406 /// @brief Seta o rotulo (nome) do eixo z.
407 Gnuplot & set_zlabel (const std::string & label = "z");
408
409 /// @brief Seta o rotulo (nome) do eixo z.
410 Gnuplot & ZLabel(const std::string & label = "z")
411 { return
    set_zlabel(
    label); }

412
413 /// @brief Seta intervalo do eixo x.
414 Gnuplot & set_xrange (const int iFrom, const int iTo);
415
416 /// @brief Seta intervalo do eixo x.
417 Gnuplot & XRange (const int iFrom, const int iTo)
418 { return
    set_xrange(
    iFrom,iTo); }

419
420 /// @brief Seta intervalo do eixo y.
421 Gnuplot & set_yrange (const int iFrom, const int iTo);
422
423 /// @brief Seta intervalo do eixo y.
424 Gnuplot & YRange (const int iFrom, const int iTo)
425 { return
    set_yrange(

```

```

                                                                    iFrom,iTo); }
426
427 /// @brief Seta intervalo do eixo z.
428 Gnuplot & set_zrange (const int iFrom, const int iTo);
429
430 /// @brief Seta intervalo do eixo z.
431 Gnuplot & ZRange (const int iFrom, const int iTo)
432 { return
                                set_zrange(
                                iFrom,iTo); }
433
434 /// @brief Seta escalonamento automatico do eixo x (default).
435 Gnuplot & set_xautoscale ();
436
437 /// @brief Seta escalonamento automatico do eixo x (default).
438 Gnuplot & XAutoscale()
                                { return
                                set_xautoscale (); }
439
440 /// @brief Seta escalonamento automatico do eixo y (default).
441 Gnuplot & set_yautoscale ();
442
443 /// @brief Seta escalonamento automatico do eixo y (default).
444 Gnuplot & YAutoscale()
                                { return
                                set_yautoscale (); }
445
446 /// @brief Seta escalonamento automatico do eixo z (default).
447 Gnuplot & set_zautoscale ();
448
449 /// @brief Seta escalonamento automatico do eixo z (default).
450 Gnuplot & ZAutoscale()
                                { return
                                set_zautoscale (); }
451
452 /// @brief Ativa escala logaritma do eixo x (logscale nao e
                                setado por default).
453 Gnuplot & set_xlogscale (const double base = 10);
454
455 /// @brief Desativa escala logaritma do eixo x (logscale nao e
                                setado por default).
456 Gnuplot & unset_xlogscale ();
457
458 /// @brief Ativa escala logaritma do eixo x (logscale nao e
                                setado por default).
```

```
459 Gnuplot & XLogscale (const double base = 10) { //if(base)
460                                     return
                                     set_xlogscale
                                     (base);
461                                     //else
462                                     //return
                                     unset_xlogscale
                                     ();
463                                     }
464
465 /// @brief Ativa/Desativa escala logaritma do eixo x (logscale
    nao e setado por default).
466 Gnuplot & XLogscale(bool _fxlogscale)
467                                     { if(fxlogscale =
                                     _fxlogscale)
468                                     return
                                     set_xlogscale
                                     ();
469                                     else
470                                     return
                                     unset_xlogscale
                                     ();
471                                     }
472
473 /// @brief Ativa escala logaritma do eixo y (logscale nao e
    setado por default).
474 Gnuplot & set_ylogscale (const double base = 10);
475
476 /// @brief Ativa escala logaritma do eixo y (logscale nao e
    setado por default).
477 Gnuplot & YLogscale (const double base = 10) { return
    set_ylogscale (base); }
478
479 /// @brief Desativa escala logaritma do eixo y (logscale nao e
    setado por default).
480 Gnuplot & unset_ylogscale ();
481
482 /// @brief Ativa/Desativa escala logaritma do eixo y (logscale
    nao e setado por default).
483 Gnuplot & YLogscale(bool _fylogscale)
484                                     { if(fylogscale =
                                     _fylogscale)
```

```

485                                     return
                                     set_ylogscale
                                     ();
486                                     else
487                                     return
                                     unset_ylogscale
                                     ();
488                                     }
489
490     /// @brief Ativa escala logaritma do eixo y (logscale nao e
        setado por default).
491     Gnuplot & set_zlogscale (const double base = 10);
492
493     /// @brief Ativa escala logaritma do eixo y (logscale nao e
        setado por default).
494     Gnuplot & ZLogscale (const double base = 10) { return
        set_zlogscale (base); }
495
496     /// @brief Desativa escala logaritma do eixo z (logscale nao e
        setado por default).
497     Gnuplot & unset_zlogscale ();
498
499     /// @brief Ativa/Desativa escala logaritma do eixo y (logscale
        nao e setado por default).
500     Gnuplot & ZLogscale(bool _fzlogscale)
501                                     { if(fzlogscale =
                                     _fzlogscale)
502                                     return
                                     set_zlogscale
                                     ();
503                                     else
504                                     return
                                     unset_zlogscale
                                     ();
505                                     }
506
507
508     /// @brief Seta intervalo da palette (autoscale por padrao).
509     Gnuplot & set_cbrange (const int iFrom, const int iTo);
510
511     /// @brief Seta intervalo da palette (autoscale por padrao).
512     Gnuplot & CBRange(const int iFrom, const int iTo)

```

```

513                                     { return
                                         set_cbrange(
                                             iFrom, iTo); }

514
515 //
-----

516 /// @brief Plota dados de um arquivo de disco.
517 Gnuplot & plotfile_x (const std::string & filename,
518                       const int column = 1, const std::string & title = "
                    ");
519
520 /// @brief Plota dados de um arquivo de disco.
521 Gnuplot & PlotFile (const std::string & filename,
522                   const int column = 1, const std::string & title = "
                    ")
523                                     { return
                                         plotfile_x(
                                             filename,
                                             column, title);
                                         }

524
525 /// @brief Plota dados de um vector.
526 Gnuplot & plot_x (const std::vector < double >&x, const std::
                    string & title = "");
527
528 /// @brief Plota dados de um vector.
529 Gnuplot & PlotVector (const std::vector < double >&x, const
                    std::string & title = "")
530                                     { return plot_x(
                                         x, title ); }

531
532 /// @brief Plota pares x,y a partir de um arquivo de disco.
533 Gnuplot & plotfile_xy (const std::string & filename,
534                       const int column_x = 1,
535                       const int column_y = 2, const std::string & title
                    = "");
536
537 /// @brief Plota pares x,y a partir de um arquivo de disco.
538 Gnuplot & PlotFile (const std::string & filename,
539                   const int column_x = 1,
540                   const int column_y = 2, const std::string & title
                    = "")

```



```

540                                     {
541                                     return
                                         plotfile_xy(
                                         filename,
                                         column_x,
                                         column_y, title
                                         );
542                                     }
543
544     /// @brief Plota pares x,y a partir de vetores.
545     Gnuplot & plot_xy (const std::vector < double >&x,
546                       const std::vector < double >&y, const std::string &
547                       title = "");
548
549     /// @brief Plota pares x,y a partir de vetores.
550     Gnuplot & PlotVector (const std::vector < double >&x,
551                          const std::vector < double >&y, const std::string &
552                          title = "")
553
554                                     { return plot_xy
555                                     ( x, y,title );
556                                     }
557
558     /// @brief Plota pares x,y com barra de erro dy a partir de um
559     arquivo.
560     Gnuplot & plotfile_xy_err (const std::string & filename,
561                               const int column_x = 1,
562                               const int column_y = 2,
563                               const int column_dy = 3, const std::string &
564                               title = "");
565
566     /// @brief Plota pares x,y com barra de erro dy a partir de um
567     arquivo.
568     Gnuplot & PlotFileXYErrorBar(const std::string & filename,
569                                  const int column_x = 1,
570                                  const int column_y = 2,
571                                  const int column_dy = 3, const std::string &
572                                  title = "")
573
574                                     { return
575                                     plotfile_xy_err
576                                     (filename,
577                                     column_x,
578                                     column_y,

```

```

                                                                    column_dy ,
                                                                    title ); }

566
567  /// @brief Plota pares x,y com barra de erro dy a partir de
    vetores.
568  Gnuplot & plot_xy_err (const std::vector < double >&x,
569                          const std::vector < double >&y,
570                          const std::vector < double >&dy,
571                          const std::string & title = "");
572
573  /// @brief Plota pares x,y com barra de erro dy a partir de
    vetores.
574  Gnuplot & PlotVectorXYErrorBar(const std::vector < double >&x,
575                                const std::vector < double >&y,
576                                const std::vector < double >&dy,
577                                const std::string & title = "")
578
                                                                    { return
                                                                    plot_xy_err
                                                                    (x, y, dy,
                                                                    title); }

579
580  /// @brief Plota valores de x,y,z a partir de um arquivo de
    disco.
581  Gnuplot & plotfile_xyz (const std::string & filename,
582                          const int column_x = 1,
583                          const int column_y = 2,
584                          const int column_z = 3, const std::string & title
                                                                    = "");
585  /// @brief Plota valores de x,y,z a partir de um arquivo de
    disco.
586  Gnuplot & PlotFile (const std::string & filename,
587                     const int column_x = 1,
588                     const int column_y = 2,
589                     const int column_z = 3, const std::string & title
                                                                    = "")
590
                                                                    { return
                                                                    plotfile_xyz
                                                                    (filename,
                                                                    column_x,
                                                                    column_y,
                                                                    column_z)
                                                                    ; }

591

```

```
592
593 /// @brief Plota valores de x,y,z a partir de vetores.
594 Gnuplot & plot_xyz (const std::vector < double >&x,
595                     const std::vector < double >&y,
596                     const std::vector < double >&z, const std::string &
                        title = "");
597
598 /// @brief Plota valores de x,y,z a partir de vetores.
599 Gnuplot & PlotVector(const std::vector < double >&x,
600                     const std::vector < double >&y,
601                     const std::vector < double >&z, const std::string &
                        title = "")
602                                     { return
                                                plot_xyz(x,
                                                    y, z,
                                                    title); }
603
604 /// @brief Plota uma equacao da forma y = ax + b, voce fornece
        os coeficientes a e b.
605 Gnuplot & plot_slope (const double a, const double b, const
                        std::string & title = "");
606
607 /// @brief Plota uma equacao da forma y = ax + b, voce fornece
        os coeficientes a e b.
608 Gnuplot & PlotSlope (const double a, const double b, const
                        std::string & title = "")
609                                     { return
                                                plot_slope(
                                                    a,b,title);
                                                }
610
611 /// @brief Plota uma equacao fornecida como uma std::string y=
        f(x).
612 /// Escrever somente a funcao f(x) e nao y=
613 /// A variavel independente deve ser x
614 /// Os operadores binarios aceitos sao:
615 /// ** exponenciacao,
616 /// * multiplicacao,
617 /// / divisao,
618 /// + adicao,
619 /// - subtracao,
620 /// % modulo
```

```

621  /// Os operadores unarios aceitos sao:
622  /// - menos,
623  /// ! fatorial
624  /// Funcoes elementares:
625  /// rand(x), abs(x), sgn(x), ceil(x), floor(x), int(x), imag(x)
        , real(x), arg(x),
626  /// sqrt(x), exp(x), log(x), log10(x), sin(x), cos(x), tan(x),
        asin(x), acos(x),
627  /// atan(x), atan2(y,x), sinh(x), cosh(x), tanh(x), asinh(x),
        acosh(x), atanh(x)
628  /// Funcoes especiais:
629  /// erf(x), erfc(x), inverf(x), gamma(x), igamma(a,x), lgamma(x)
        ), ibeta(p,q,x),
630  /// besj0(x), besj1(x), besy0(x), besy1(x), lambertw(x)
631  /// Funcoes estatisticas:
632  /// norm(x), invnorm(x)
633  Gnuplot & plot_equation (const std::string & equation,
634                          const std::string & title = "");
635
636  /// @brief Plota uma equacao fornecida como uma std::string y=
        f(x).
637  /// Escrever somente a funcao f(x) e nao y=
638  /// A variavel independente deve ser x.
639  /// Exemplo: gnuplot->PlotEquation(CFuncao& obj);
640  /// Deve receber um CFuncao, que tem cast para string.
641  Gnuplot & PlotEquation(const std::string & equation,
642                        const std::string & title = "")
643  { return
        plot_equation(
            equation, title )
        ; }
644
645  /// @brief Plota uma equacao fornecida na forma de uma std::
        string z=f(x,y).
646  /// Escrever somente a funcao f(x,y) e nao z=, as variaveis
        independentes sao x e y.
647  Gnuplot & plot_equation3d (const std::string & equation, const
        std::string & title = "");
648
649  /// @brief Plota uma equacao fornecida na forma de uma std::
        string z=f(x,y).
650  /// Escrever somente a funcao f(x,y) e nao z=, as vaiaveis

```

```

        independentes sao x e y.
651 // gnuplot->PlotEquation3d(CPolinomio());
652 Gnuplot & PlotEquation3d (const std::string & equation,
653                             const std::string & title = "")
654                             { return
                                plot_equation3d(
                                    equation, title )
                                ; }

655
656 /// @brief Plota uma imagem.
657 Gnuplot & plot_image (const unsigned char *ucPicBuf,
658                       const int iWidth, const int iHeight, const std::
659                           string & title = "");
660
661 /// @brief Plota uma imagem.
662 Gnuplot & PlotImage (const unsigned char *ucPicBuf,
663                     const int iWidth, const int iHeight,
664                       const std::string & title = "")
665                     { return plot_image
                        (ucPicBuf,
                         iWidth, iHeight,
                         title); }
666
667 //
668 -----
669
670 // Repete o ultimo comando de plotagem, seja plot (2D) ou splot
671 // (3D)
672 // Usado para visualizar plotagens, após mudar algumas opcoes
673 // de plotagem
674 // ou quando gerando o mesmo grafico para diferentes
675 // dispositivos (showonscreen, savetops)
676 Gnuplot & replot ();
677
678 // Repete o ultimo comando de plotagem, seja plot (2D) ou splot
679 // (3D)
680 // Usado para visualizar plotagens, após mudar algumas opcoes
681 // de plotagem
682 // ou quando gerando o mesmo grafico para diferentes
683 // dispositivos (showonscreen, savetops)
684 Gnuplot & Replot() { return replot()
    ; }

```

```

675
676 // Reseta uma sessao do gnuplot (próxima plotagem apaga
    definicoes previas)
677 Gnuplot & reset_plot ();
678
679 // Reseta uma sessao do gnuplot (próxima plotagem apaga
    definicoes previas)
680 Gnuplot & ResetPlot() { return
    reset_plot(); }
681
682 // Chama função reset do gnuplot
683 Gnuplot & Reset() { this->cmd("reset");
    return *this; }
684
685 // Reseta uma sessao do gnuplot e seta todas as variaveis para
    o default
686 Gnuplot & reset_all ();
687
688 // Reseta uma sessao do gnuplot e seta todas as variaveis para
    o default
689 Gnuplot & ResetAll () { return
    reset_all(); }
690
691 // Verifica se a sessao esta valida
692 bool is_valid ();
693
694 // Verifica se a sessao esta valida
695 bool IsValid () { return is_valid
    (); };
696
697 };
698 typedef Gnuplot CGnuplot;
699 #endif

```

Apresenta-se na listagem 6.31 o arquivo de implementação da classe CGnuplot.

Listing 6.4: Arquivo de implementação da classe CGnuplot.

```

1 //////////////////////////////////////
2 //
3 // A C++ interface to gnuplot.
4 //
5 // This is a direct translation from the C interface
6 // written by Nicolas Devillard (ndevilla@free.fr) which

```

```

7// is available from http://ndevilla.free.fr/gnuplot/.
8//
9// As in the C interface this uses pipes and so wont
10// run on a system that doesn't have POSIX pipe support
11//
12// Rajarshi Guha
13// e-mail: rguha@indiana.edu, rajarshi@presidency.com
14// http://cheminfo.informatics.indiana.edu/~rguha/code/cc++/
15//
16// 07/03/03
17//
18////////////////////////////////////
19//
20// A little correction for Win32 compatibility
21// and MS VC 6.0 done by V.Chyzhdenka
22//
23// Notes:
24// 1. Added private method Gnuplot::init().
25// 2. Temporary file is created in the current
26//     folder but not in /tmp.
27// 3. Added #ifdef WIN32 e.t.c. where is needed.
28// 4. Added private member m_sGNUPlotFileName is
29//     a name of executed GNUPlot file.
30//
31// Viktor Chyzhdenka
32// e-mail: chyzhdenka@mail.ru
33//
34// 20/05/03
35//
36////////////////////////////////////
37//
38// corrections for Win32 and Linux compatibility
39//
40// some member functions added:
41//   set_GNUPlotPath, set_terminal_std,
42//   create_tmpfile, get_program_path, file_exists,
43//   operator<<, replot, reset_all, savetops, showonscreen,
44//   plotfile_*, plot_xy_err, plot_equation3d
45// set, unset: pointsize, grid, *logscale, *autoscale,
46// smooth, title, legend, samples, isosamples,
47// hidden3d, cbrange, contour
48//

```

```

49// Markus Burgis
50// e-mail: mail@burgis.info
51//
52// 10/03/08
53//
54////////////////////////////////////
55//
56// Modificacoes:
57// Traducaao para o portugues
58// Adicao de novos nomes para os metodos(funcoes)
59// Uso de documentacao no formato javadoc/doxygen
60// Bueno A.D.
61// e-mail: bueno@lenep.uenf.br
62// 20/07/08
63//
64////////////////////////////////////
65#include <fstream>           // for std::ifstream
66#include <sstream>           // for std::ostringstream
67#include <list>              // for std::list
68#include <cstdio>            // for FILE, fputs(), fflush(),
    popen()
69#include <cstdlib>           // for getenv()
70#include "CGnuplot.h"
71
72// Se estamos no windows // defined for 32 and 64-bit
    environments
73#if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)
74 #include <io.h>             // for _access(), _mktemp()
75 #define GP_MAX_TMP_FILES 27 // 27 temporary files it's
    Microsoft restriction
76// Se estamos no unix, GNU/Linux, Mac Os X
77#elif defined(unix) || defined(__unix) || defined(__unix__) ||
    defined(__APPLE__) //all UNIX-like OSs (Linux, *BSD, MacOSX,
    Solaris, ...)
78 #include <unistd.h>         // for access(), mkstemp()
79 #define GP_MAX_TMP_FILES 64
80#else
81 #error unsupported or unknown operating system
82#endif
83
84//

```



```

-----

85 //
86 // initialize static data
87 //
88 int Gnuplot::tmpfile_num = 0;
89
90 // Se estamos no windows
91 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)
92 std::string Gnuplot::m_sGnuplotFileName = "gnuplot.exe";
93 std::string Gnuplot::m_sGnuplotPath = "C:/gnuplot/bin/";
94 // Se estamos no unix, GNU/Linux, Mac Os X
95 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
    defined(__APPLE__)
96 std::string Gnuplot::m_sGnuplotFileName = "gnuplot";
97 std::string Gnuplot::m_sGnuplotPath = "/usr/bin/";
98 #endif
99
100 // Se estamos no windows
101 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)
102 std::string Gnuplot::terminal_std = "windows";
103 // Se estamos no unix, GNU/Linux
104 #elif ( defined(unix) || defined(__unix) || defined(__unix__) )
    && !defined(__APPLE__)
105 std::string Gnuplot::terminal_std = "x11";
106 // Se estamos Mac Os X
107 #elif defined(__APPLE__)
108 std::string Gnuplot::terminal_std = "aqua";
109 #endif
110
111 //
-----

112 //
113 // define static member function: set Gnuplot path manual
114 //   for windows: path with slash '/' not backslash '\'
115 //
116 bool Gnuplot::set_GNUPlotPath(const std::string &path)
117 {
118     std::string tmp = path + "/" + Gnuplot::m_sGnuplotFileName;

```

```

119 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)
120     if ( Gnuplot::file_exists(tmp,0) ) // check existence
121 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
    defined(__APPLE__)
122     if ( Gnuplot::file_exists(tmp,1) ) // check existence and
        execution permission
123 #endif
124     {
125         Gnuplot::m_sGnUPlotPath = path;
126         return true;
127     }
128     else
129     {
130         Gnuplot::m_sGnUPlotPath.clear();
131         return false;
132     }
133 }
134
135 //
    -----
136 // define static member function: set standart terminal, used by
    showonscreen
137 // defaults: Windows - win, Linux - x11, Mac - aqua
138 void Gnuplot::set_terminal_std(const std::string &type)
139 {
140 #if defined(unix) || defined(__unix) || defined(__unix__) ||
    defined(__APPLE__)
141     if (type.find("x11") != std::string::npos && getenv("DISPLAY"
        ) == NULL)
142     {
143         throw GnuplotException("Can't find DISPLAY variable");
144     }
145 #endif
146
147
148     Gnuplot::terminal_std = type;
149     return;
150 }
151
152

```

```
153 //
-----

154 // A string tokenizer taken from http://www.sunsite.ualberta.ca/
    Documentation/
155 // /Gnu/libstdc++-2.90.8/html/21_strings/stringtok_std_h.txt
156 template <typename Container>
157 void stringtok (Container &container,
158                 std::string const &in,
159                 const char * const delimiters = "\t\n")
160 {
161     const std::string::size_type len = in.length();
162     std::string::size_type i = 0;
163
164     while ( i < len )
165     {
166         // eat leading whitespace
167         i = in.find_first_not_of (delimiters, i);
168
169         if (i == std::string::npos)
170             return;    // nothing left but white space
171
172         // find the end of the token
173         std::string::size_type j = in.find_first_of (delimiters,
174             i);
175
176         // push token
177         if (j == std::string::npos)
178         {
179             container.push_back (in.substr(i));
180             return;
181         }
182         else
183             container.push_back (in.substr(i, j-i));
184
185         // set up for next loop
186         i = j + 1;
187     }
188     return;
189 }
190
```

```
191 //
-----

192 //
193 // constructor: set a style during construction
194 //
195 Gnuplot::Gnuplot(const std::string &style)
196 {
197     this->init();
198     this->set_style(style);
199 }
200
201 //
-----

202 // constructor: open a new session, plot a signal (x)
203 Gnuplot::Gnuplot(const std::vector<double> &x,
204                 const std::string &title,
205                 const std::string &style,
206                 const std::string &labelx,
207                 const std::string &labely)
208 {
209     this->init();
210
211     this->set_style(style);
212     this->set_xlabel(labelx);
213     this->set_ylabel(labely);
214
215     this->plot_x(x, title);
216 }
217
218 //
-----

219 // constructor: open a new session, plot a signal (x,y)
220 Gnuplot::Gnuplot(const std::vector<double> &x,
221                 const std::vector<double> &y,
222                 const std::string &title,
223                 const std::string &style,
224                 const std::string &labelx,
225                 const std::string &labely)
226 {
```

```

227     this->init();
228
229     this->set_style(style);
230     this->set_xlabel(labelx);
231     this->set_ylabel(labely);
232
233     this->plot_xy(x,y,title);
234 }
235
236 //
-----

237 // constructor: open a new session, plot a signal (x,y,z)
238 Gnuplot::Gnuplot(const std::vector<double> &x,
239                 const std::vector<double> &y,
240                 const std::vector<double> &z,
241                 const std::string &title,
242                 const std::string &style,
243                 const std::string &labelx,
244                 const std::string &labely,
245                 const std::string &labelz)
246 {
247     this->init();
248
249     this->set_style(style);
250     this->set_xlabel(labelx);
251     this->set_ylabel(labely);
252     this->set_zlabel(labelz);
253
254     this->plot_xyz(x,y,z,title);
255 }
256
257 //
-----

258 // Destructor: needed to delete temporary files
259 Gnuplot::~Gnuplot()
260 {
261     if ((this->tmpfile_list).size() > 0)
262     {
263         for (unsigned int i = 0; i < this->tmpfile_list.size(); i
                ++)
```

```

264         remove( this->tmpfile_list[i].c_str() );
265
266         Gnuplot::tmpfile_num -= this->tmpfile_list.size();
267     }
268
269     // A stream opened by popen() should be closed by pclose()
270 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)
271     if (_pclose(this->gnucmd) == -1)
272 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
    defined(__APPLE__)
273     if (pclose(this->gnucmd) == -1)
274 #endif
275         true; //throw GnuplotException("Problem closing
                communication to gnuplot");
276 }
277
278 //
    -----
279 // Resets a gnuplot session (next plot will erase previous ones)
280 Gnuplot& Gnuplot::reset_plot()
281 {
282     if (this->tmpfile_list.size() > 0)
283     {
284         for (unsigned int i = 0; i < this->tmpfile_list.size(); i
            ++ )
285             remove(this->tmpfile_list[i].c_str());
286
287         Gnuplot::tmpfile_num -= this->tmpfile_list.size();
288         this->tmpfile_list.clear();
289     }
290
291     this->nplots = 0;
292
293     return *this;
294 }
295
296 //
    -----
297 // resets a gnuplot session and sets all variables to default

```

```

298 Gnuplot& Gnuplot::reset_all()
299 {
300     if (this->tmpfile_list.size() > 0)
301     {
302         for (unsigned int i = 0; i < this->tmpfile_list.size(); i
            ++
303             remove(this->tmpfile_list[i].c_str());
304
305         Gnuplot::tmpfile_num -= this->tmpfile_list.size();
306         this->tmpfile_list.clear();
307     }
308
309     this->nplots = 0;
310     this->cmd("reset");
311     this->cmd("clear");
312     this->pstyle = "points";
313     this->smooth = "";
314     this->showonscreen();
315
316     return *this;
317 }
318
319 //
-----

320 // Find out if valid is true
321 bool Gnuplot::is_valid()
322 {
323     return(this->valid);
324 }
325
326 //
-----

327 // replot repeats the last plot or splot command
328 Gnuplot& Gnuplot::replot()
329 {
330     if (this->nplots > 0)
331     {
332         this->cmd("replot");
333     }
334

```

```

335     return *this;
336 }
337
338
339 //
-----

340 // Change the plotting style of a gnuplot session
341 Gnuplot& Gnuplot::set_style(const std::string &stylestr)
342 {
343     if (stylestr.find("lines")           == std::string::npos    &&
344         stylestr.find("points")          == std::string::npos    &&
345         stylestr.find("linespoints")     == std::string::npos    &&
346         stylestr.find("impulses")        == std::string::npos    &&
347         stylestr.find("dots")             == std::string::npos    &&
348         stylestr.find("steps")            == std::string::npos    &&
349         stylestr.find("fsteps")           == std::string::npos    &&
350         stylestr.find("histeps")          == std::string::npos    &&
351         stylestr.find("boxes")            == std::string::npos    &&
352         // 1-4 columns of data are required
353         stylestr.find("filledcurves")     == std::string::npos    &&
354         stylestr.find("histograms")       == std::string::npos    )
355         //only for one data column
356         stylestr.find("labels")           == std::string::npos
357         && // 3 columns of data are required
358         stylestr.find("xerrorbars")       == std::string::npos
359         && // 3-4 columns of data are required
360         stylestr.find("xerrorlines")      == std::string::npos
361         && // 3-4 columns of data are required
362         stylestr.find("errorbars")        == std::string::npos
363         && // 3-4 columns of data are required
364         stylestr.find("errorlines")       == std::string::npos
365         && // 3-4 columns of data are required
366         stylestr.find("yerrorbars")       == std::string::npos
367         && // 3-4 columns of data are required
368         stylestr.find("yerrorlines")      == std::string::npos
369         && // 3-4 columns of data are required
370         stylestr.find("boxerrorbars")     == std::string::npos
371         && // 3-5 columns of data are required
372         stylestr.find("xyerrorbars")      == std::string::npos
373         && // 4,6,7 columns of data are required
374         stylestr.find("xyerrorlines")     == std::string::npos

```



```

    && // 4,6,7 columns of data are required
364 //     stylestr.find("boxxyerrorbars") == std::string::npos
    && // 4,6,7 columns of data are required
365 //     stylestr.find("financebars") == std::string::npos
    && // 5 columns of data are required
366 //     stylestr.find("candlesticks") == std::string::npos
    && // 5 columns of data are required
367 //     stylestr.find("vectors") == std::string::npos
    &&
368 //     stylestr.find("image") == std::string::npos
    &&
369 //     stylestr.find("rgbimage") == std::string::npos
    &&
370 //     stylestr.find("pm3d") == std::string::npos )
371 {
372     this->pstyle = std::string("points");
373 }
374 else
375 {
376     this->pstyle = stylestr;
377 }
378
379 return *this;
380 }
381
382 //
-----

383 // smooth: interpolation and approximation of data
384 Gnuplot& Gnuplot::set_smooth(const std::string &stylestr)
385 {
386     if (stylestr.find("unique") == std::string::npos &&
387         stylestr.find("frequency") == std::string::npos &&
388         stylestr.find("csplines") == std::string::npos &&
389         stylestr.find("acsplines") == std::string::npos &&
390         stylestr.find("bezier") == std::string::npos &&
391         stylestr.find("sbezier") == std::string::npos )
392     {
393         this->smooth = "";
394     }
395     else
396     {

```

```

397         this->smooth = stylestr;
398     }
399
400     return *this;
401 }
402
403 //
-----

404 // unset smooth
405 Gnuplot& Gnuplot::unset_smooth()
406 {
407     this->smooth = "";
408
409     return *this;
410 }
411
412 //
-----

413 // sets terminal type to windows / x11
414 Gnuplot& Gnuplot::showonscreen()
415 {
416     this->cmd("set output");
417     this->cmd("set terminal" + Gnuplot::terminal_std);
418
419     return *this;
420 }
421
422 //
-----

423 // saves a gnuplot session to a postscript file
424 Gnuplot& Gnuplot::savetops(const std::string &filename)
425 {
426 //     this->cmd("set terminal postscript color");           //
        Tipo de terminal (tipo de arquivo)
427 //
428 //     std::ostream cmdstr;                                 //
        Muda o nome do arquivo
429 //     cmdstr << "set output \"" << filename << ".ps\"";    //
        Nome do arquivo

```

```

430 //      this->cmd(cmdstr.str());
431 //      this->replot();                                     //
      Replota o gráfico, agora salvando o arquivo ps
432 //
433 //      ShowOnScreen ();                                     //
      Volta para terminal modo janela
434
435      this->cmd("set term png size 1800,1200");
436
437      std::ostream cmdstr;
438      cmdstr << "set output \"./images/\" << filename << ".png\"";
439      this->cmd(cmdstr.str());
440      this->Replot();
441
442      return *this;
443 }
444 //
      -----
445 // saves a gnuplot session to a png file and return do on screen
      terminal
446 Gnuplot& Gnuplot::savetopng(const std::string &filename)
447 {
448 //                                     //
      Muda o terminal
449 //      this->cmd("set term png enhanced size 1280,960");    //
      Tipo de terminal (tipo de arquivo)
450 //
451 //      std::ostream cmdstr;                                     //
      Muda o nome do arquivo
452 //      cmdstr << "set output \"" << filename << ".png\"";    //
      Nome do arquivo
453 //      this->cmd(cmdstr.str());
454 //      this->replot();                                         //
      Replota o gráfico, agora salvando o arquivo ps
455 //                                     //
      Retorna o terminal para o padrão janela
456 //      ShowOnScreen ();                                     //
      Volta para terminal modo janela
457 //      this->replot();                                         //
      Replota o gráfico, agora na tela
458      SaveTo(filename,"png", "enhanced size 1280,960");

```

```

459
460     return *this;
461 }
462
463 //
-----

464 // saves a gnuplot session to a jpeg file and return do on screen
    terminal
465 Gnuplot& Gnuplot::savetojpeg(const std::string &filename)
466 {
467                                     // Muda o
                                     terminal
468 //      this->cmd("set term jpeg enhanced size 1280,960"); //
    Tipo de terminal (tipo de arquivo)
469 //
470 //      std::ostringstream cmdstr;                                     //
    Muda o nome do arquivo
471 //      cmdstr << "set output \"" << filename << ".jpeg\""; //
    Nome do arquivo
472 //      this->cmd(cmdstr.str());
473 //      this->replot();                                               //
    Replota o gráfico, agora salvando o arquivo ps
474 //                                                                    //
    Retorna o terminal para o padrão janela
475 //      ShowOnScreen ();                                           //
    Volta para terminal modo janela
476 //      this->replot();                                               //
    Replota o gráfico, agora na tela
477     SaveTo(filename,"jpeg", "enhanced_size_1280,960");
478
479     return *this;
480 }
481
482
483 //
-----

484 // saves a gnuplot session to spectific terminal and output file
485 // @filename: name of disc file

```

```

486 // @terminal_type: type of terminal
487 // @flags: additional information specific to terminal type
488 // Ex:
489 // grafico.SaveTo("pressao_X_temperatura","png", "enhanced size
      1280,960");
490 // grafico.TerminalType("png").SaveFile(pressao_X_temperatura);
      pense nisso?
491 Gnuplot& Gnuplot::SaveTo(const std::string &filename, const std::
      string &terminal_type, std::string flags)
492 {
      // Muda o
      terminal
493   this->cmd("set term " + terminal_type + " " + flags); //
      Tipo de terminal (tipo de arquivo) e flags adicionais
494   std::ostream cmdstr; // Muda o
      nome do arquivo
495   cmdstr << "set output \" " << filename << "." << terminal_type
      << "\"";
496   this->cmd(cmdstr.str());
497   this->replot(); //
      Replota o gráfico, agora salvando o arquivo ps
498 //
      Retorna
      o
      terminal
      para o
      padrão
      janela

499   ShowOnScreen ();
500   this->replot(); //
      Replota o gráfico, agora na tela
501
502   return *this;
503 }
504
505 //
-----

506 // Switches legend on
507 Gnuplot& Gnuplot::set_legend(const std::string &position)
508 {
509   std::ostream cmdstr;
510   cmdstr << "set key " << position;

```

```
511
512     this->cmd(cmdstr.str());
513
514     return *this;
515 }
516
517 //
-----

518 // Switches legend off
519 Gnuplot& Gnuplot::unset_legend()
520 {
521     this->cmd("unset _key");
522
523     return *this;
524 }
525
526 //
-----

527 // Turns grid on
528 Gnuplot& Gnuplot::set_grid()
529 {
530     this->cmd("set _grid");
531
532     return *this;
533 }
534
535 //
-----

536 // Turns grid off
537 Gnuplot& Gnuplot::unset_grid()
538 {
539     this->cmd("unset _grid");
540
541     return *this;
542 }
543
544 //
-----
```

```
545 // turns on log scaling for the x axis
546 Gnuplot& Gnuplot::set_xlogscale(const double base)
547 {
548     std::ostringstream cmdstr;
549
550     cmdstr << "set_xlogscale_x" << base;
551     this->cmd(cmdstr.str());
552
553     return *this;
554 }
555
556 //
-----

557 // turns on log scaling for the y axis
558 Gnuplot& Gnuplot::set_ylogscale(const double base)
559 {
560     std::ostringstream cmdstr;
561
562     cmdstr << "set_ylogscale_y" << base;
563     this->cmd(cmdstr.str());
564
565     return *this;
566 }
567
568 //
-----

569 // turns on log scaling for the z axis
570 Gnuplot& Gnuplot::set_zlogscale(const double base)
571 {
572     std::ostringstream cmdstr;
573
574     cmdstr << "set_zlogscale_z" << base;
575     this->cmd(cmdstr.str());
576
577     return *this;
578 }
579
580 //
```

```
581// turns off log scaling for the x axis
582Gnuplot& Gnuplot::unset_xlogscale()
583{
584    this->cmd("unset_logscale_x");
585    return *this;
586}
587
588//
-----

589// turns off log scaling for the y axis
590Gnuplot& Gnuplot::unset_ylogscale()
591{
592    this->cmd("unset_logscale_y");
593    return *this;
594}
595
596//
-----

597// turns off log scaling for the z axis
598Gnuplot& Gnuplot::unset_zlogscale()
599{
600    this->cmd("unset_logscale_z");
601    return *this;
602}
603
604
605//
-----

606// scales the size of the points used in plots
607Gnuplot& Gnuplot::set_pointsize(const double pointsize)
608{
609    std::ostringstream cmdstr;
610    cmdstr << "set_pointsize_" << pointsize;
611    this->cmd(cmdstr.str());
612
613    return *this;
614}
615
616//
```



```
-----

617// set isoline density (grid) for plotting functions as surfaces
618Gnuplot& Gnuplot::set_samples(const int samples)
619{
620    std::ostringstream cmdstr;
621    cmdstr << "set_samples_" << samples;
622    this->cmd(cmdstr.str());
623
624    return *this;
625}
626
627//
-----

628// set isoline density (grid) for plotting functions as surfaces
629Gnuplot& Gnuplot::set_isosamples(const int isolines)
630{
631    std::ostringstream cmdstr;
632    cmdstr << "set_isosamples_" << isolines;
633    this->cmd(cmdstr.str());
634
635    return *this;
636}
637
638//
-----

639// enables hidden line removal for surface plotting
640Gnuplot& Gnuplot::set_hidden3d()
641{
642    this->cmd("set_hidden3d");
643
644    return *this;
645}
646
647//
-----

648// disables hidden line removal for surface plotting
649Gnuplot& Gnuplot::unset_hidden3d()
650{
```

```

651     this->cmd("unset_hidden3d");
652
653     return *this;
654 }
655
656 //
-----

657 // enables contour drawing for surfaces set contour {base |
        surface | both}
658 Gnuplot& Gnuplot::set_contour(const std::string &position)
659 {
660     if (position.find("base")      == std::string::npos    &&
661         position.find("surface") == std::string::npos    &&
662         position.find("both")     == std::string::npos    )
663     {
664         this->cmd("set_contour_base");
665     }
666     else
667     {
668         this->cmd("set_contour_" + position);
669     }
670
671     return *this;
672 }
673
674 //
-----

675 // disables contour drawing for surfaces
676 Gnuplot& Gnuplot::unset_contour()
677 {
678     this->cmd("unset_contour");
679
680     return *this;
681 }
682
683 //
-----

684 // enables the display of surfaces (for 3d plot)
685 Gnuplot& Gnuplot::set_surface()

```

```
686 {
687     this->cmd("set_surface");
688
689     return *this;
690 }
691
692 //
-----
693 // disables the display of surfaces (for 3d plot)
694 Gnuplot& Gnuplot::unset_surface()
695 {
696     this->cmd("unset_surface");
697
698     return *this;
699 }
700
701 //
-----
702 // Sets the title of a gnuplot session
703 Gnuplot& Gnuplot::set_title(const std::string &title)
704 {
705     std::ostringstream cmdstr;
706
707     cmdstr << "set_title\" << title << "\"";
708     this->cmd(cmdstr.str());
709
710     return *this;
711 }
712
713 //
-----
714 // Clears the title of a gnuplot session
715 Gnuplot& Gnuplot::unset_title()
716 {
717     this->set_title("");
718
719     return *this;
720 }
721
```

```
722 //
-----

723 // set labels
724 // set the xlabel
725 Gnuplot& Gnuplot::set_xlabel(const std::string &label)
726 {
727     std::ostringstream cmdstr;
728
729     cmdstr << "set_xlabel\" << label << "\"";
730     this->cmd(cmdstr.str());
731
732     return *this;
733 }
734
735 //
-----

736 // set the ylabel
737 Gnuplot& Gnuplot::set_ylabel(const std::string &label)
738 {
739     std::ostringstream cmdstr;
740
741     cmdstr << "set_ylabel\" << label << "\"";
742     this->cmd(cmdstr.str());
743
744     return *this;
745 }
746
747 //
-----

748 // set the zlabel
749 Gnuplot& Gnuplot::set_zlabel(const std::string &label)
750 {
751     std::ostringstream cmdstr;
752
753     cmdstr << "set_zlabel\" << label << "\"";
754     this->cmd(cmdstr.str());
755
756     return *this;
757 }
```

```
758
759 //
-----

760 // set range
761 // set the xrange
762 Gnuplot& Gnuplot::set_xrange(const int iFrom,
763                               const int iTo)
764 {
765     std::ostringstream cmdstr;
766
767     cmdstr << "set_xrange[" << iFrom << ":" << iTo << "];";
768     this->cmd(cmdstr.str());
769
770     return *this;
771 }
772
773 //
-----

774 // set autoscale x
775 Gnuplot& Gnuplot::set_xautoscale()
776 {
777     this->cmd("set_xrange_restore");
778     this->cmd("set_autoscale_x");
779
780     return *this;
781 }
782
783 //
-----

784 // set the yrange
785 Gnuplot& Gnuplot::set_yrange(const int iFrom, const int iTo)
786 {
787     std::ostringstream cmdstr;
788
789     cmdstr << "set_yrange[" << iFrom << ":" << iTo << "];";
790     this->cmd(cmdstr.str());
791
792     return *this;
793 }
```

```
794
795 //
-----

796 // set autoscale y
797 Gnuplot& Gnuplot::set_yautoscale()
798 {
799     this->cmd("set_yrange_restore");
800     this->cmd("set_autoscale_y");
801
802     return *this;
803 }
804
805 //
-----

806 // set the zrange
807 Gnuplot& Gnuplot::set_zrange(const int iFrom,
808                               const int iTo)
809 {
810     std::ostringstream cmdstr;
811
812     cmdstr << "set_zrange[" << iFrom << ":" << iTo << "]";
813     this->cmd(cmdstr.str());
814
815     return *this;
816 }
817
818 //
-----

819 // set autoscale z
820 Gnuplot& Gnuplot::set_zautoscale()
821 {
822     this->cmd("set_zrange_restore");
823     this->cmd("set_autoscale_z");
824
825     return *this;
826 }
827
828 //
```

```

829 // set the palette range
830 Gnuplot& Gnuplot::set_cbrange(const int iFrom,
831                               const int iTo)
832 {
833     std::ostringstream cmdstr;
834
835     cmdstr << "set_cbrange[" << iFrom << ":" << iTo << "];
836     this->cmd(cmdstr.str());
837
838     return *this;
839 }
840
841 //
-----
842 // Plots a linear equation y=ax+b (where you supply the
843 // slope a and intercept b)
844 Gnuplot& Gnuplot::plot_slope(const double a,
845                               const double b,
846                               const std::string &title)
847 {
848     std::ostringstream cmdstr;
849
850     // command to be sent to gnuplot
851     if (this->nplots > 0 && this->two_dim == true)
852         cmdstr << "replot_";
853     else
854         cmdstr << "plot_";
855
856     cmdstr << a << "_*_x+_ " << b << "_title_\n";
857
858     if (title == "")
859         cmdstr << "f(x)_=_ " << a << "_*_x+_ " << b;
860     else
861         cmdstr << title;
862
863     cmdstr << "\"_with_ " << this->pstyle;
864
865     // Do the actual plot
866     this->cmd(cmdstr.str());
867

```

```

868     return *this;
869 }
870
871 //
-----

872 // Plot an equation which is supplied as a std::string y=f(x) (
      only f(x) expected)
873 Gnuplot& Gnuplot::plot_equation(const std::string &equation,
874                                const std::string &title)
875 {
876     std::ostringstream cmdstr;
877
878     // command to be sent to gnuplot
879     if (this->nplots > 0 && this->two_dim == true)
880         cmdstr << "replot_";
881     else
882         cmdstr << "plot_";
883
884     cmdstr << equation << "_title_" << title;
885
886     if (title == "")
887         cmdstr << "f(x)_" << equation;
888     else
889         cmdstr << title;
890
891     cmdstr << "\"_with_" << this->pstyle;
892
893     // Do the actual plot
894     this->cmd(cmdstr.str());
895
896     return *this;
897 }
898
899 //
-----

900 // plot an equation supplied as a std::string y=(x)
901 Gnuplot& Gnuplot::plot_equation3d(const std::string &equation,
902                                   const std::string &title)
903 {
904     std::ostringstream cmdstr;

```



```

905
906 // command to be sent to gnuplot
907 if (this->nplots > 0 && this->two_dim == false)
908     cmdstr << "replot_";
909 else
910     cmdstr << "splot_";
911
912 cmdstr << equation << "_title_\"";
913
914 if (title == "")
915     cmdstr << "f(x,y)_=_\" << equation;
916 else
917     cmdstr << title;
918
919 cmdstr << "\"_with_\" << this->pstyle;
920
921 // Do the actual plot
922 this->cmd(cmdstr.str());
923
924 return *this;
925 }
926
927 //
-----
928 // Plots a 2d graph from a list of doubles (x) saved in a file
929 Gnuplot& Gnuplot::plotfile_x(const std::string &filename,
930                             const int column,
931                             const std::string &title)
932 {
933     // check if file exists
934     if( !(Gnuplot::file_exists(filename,4)) ) // check existence
        and read permission
935     {
936         std::ostringstream except;
937         if( !(Gnuplot::file_exists(filename,0)) ) // check
            existence
938             except << "File_\" << filename << "\"_does_not_exist
                ";
939         else
940             except << "No_read_permission_for_File_\" <<
                filename << "\"";

```

```

941         throw GnuplotException( except.str() );
942         return *this;
943     }
944
945     std::ostringstream cmdstr;
946
947     // command to be sent to gnuplot
948     if (this->nplots > 0 && this->two_dim == true)
949         cmdstr << "replot_";
950     else
951         cmdstr << "plot_";
952
953     cmdstr << "\" " << filename << "\"_using_" << column;
954
955     if (title == "")
956         cmdstr << "_notitle_";
957     else
958         cmdstr << "_title_\" " << title << "\"_";
959
960     if(smooth == "")
961         cmdstr << "with_" << this->pstyle;
962     else
963         cmdstr << "smooth_" << this->smooth;
964
965     // Do the actual plot
966     this->cmd(cmdstr.str()); //nplots++; two_dim = true; already
        in this->cmd();
967
968     return *this;
969 }
970
971 //
-----
972 // Plots a 2d graph from a list of doubles: x
973 Gnuplot& Gnuplot::plot_x(const std::vector<double> &x,
974                          const std::string &title)
975 {
976     if (x.size() == 0)
977     {
978         throw GnuplotException("std::vector_too_small");
979         return *this;

```

```

980     }
981
982     std::ofstream tmp;
983     std::string name = create_tmpfile(tmp);
984     if (name == "")
985         return *this;
986
987     // write the data to file
988     for (unsigned int i = 0; i < x.size(); i++)
989         tmp << x[i] << std::endl;
990
991     tmp.flush();
992     tmp.close();
993
994     this->plotfile_x(name, 1, title);
995
996     return *this;
997 }
998
999 //
-----
1000 // Plots a 2d graph from a list of doubles (x y) saved in a file
1001 Gnuplot& Gnuplot::plotfile_xy(const std::string &filename,
1002                               const int column_x,
1003                               const int column_y,
1004                               const std::string &title)
1005 {
1006     // check if file exists
1007     if( !(Gnuplot::file_exists(filename,4)) ) // check existence
        and read permission
1008     {
1009         std::ostringstream except;
1010         if( !(Gnuplot::file_exists(filename,0)) ) // check
            existence
1011             except << "File_\n" << filename << "\n_does_not_exist
                ";
1012         else
1013             except << "No_read_permission_for_File_\n" <<
                filename << "\n";
1014         throw GnuplotException( except.str() );
1015         return *this;

```

```

1016     }
1017
1018     std::ostringstream cmdstr;
1019
1020     // command to be sent to gnuplot
1021     if (this->nplots > 0 && this->two_dim == true)
1022         cmdstr << "replot_";
1023     else
1024         cmdstr << "plot_";
1025
1026     cmdstr << "\"\" << filename << "\"_using_\" << column_x << ":"
        << column_y;
1027
1028     if (title == "")
1029         cmdstr << "_notitle_";
1030     else
1031         cmdstr << "_title_\"\" << title << "\"_\"";
1032
1033     if(smooth == "")
1034         cmdstr << "with_" << this->pstyle;
1035     else
1036         cmdstr << "smooth_" << this->smooth;
1037
1038     // Do the actual plot
1039     this->cmd(cmdstr.str());
1040
1041     return *this;
1042 }
1043
1044 //
-----
1045 // Plots a 2d graph from a list of doubles: x y
1046 Gnuplot& Gnuplot::plot_xy(const std::vector<double> &x,
1047                          const std::vector<double> &y,
1048                          const std::string &title)
1049 {
1050     if (x.size() == 0 || y.size() == 0)
1051     {
1052         throw GnuplotException("std::vectors_too_small");
1053         return *this;
1054     }

```

```

1055
1056     if (x.size() != y.size())
1057     {
1058         throw GnuplotException("Length of the std::vectors
1059                                differs");
1059         return *this;
1060     }
1061
1062     std::ofstream tmp;
1063     std::string name = create_tmpfile(tmp);
1064     if (name == "")
1065         return *this;
1066
1067     // write the data to file
1068     for (unsigned int i = 0; i < x.size(); i++)
1069         tmp << x[i] << " " << y[i] << std::endl;
1070
1071     tmp.flush();
1072     tmp.close();
1073
1074     this->plotfile_xy(name, 1, 2, title);
1075
1076     return *this;
1077 }
1078
1079 //
-----
1080 // Plots a 2d graph with errorbars from a list of doubles (x y dy
1081 // ) saved in a file
1082 Gnuplot& Gnuplot::plotfile_xy_err(const std::string &filename,
1083                                   const int column_x,
1084                                   const int column_y,
1085                                   const int column_dy,
1086                                   const std::string &title)
1087 {
1088     // check if file exists
1089     if( !(Gnuplot::file_exists(filename,4)) ) // check existence
1090         and read permission
1091     {
1092         std::ostringstream except;
1093         if( !(Gnuplot::file_exists(filename,0)) ) // check

```

```

        existence
1092         except << "File_" << filename << "\"_does_not_exist
            ";
1093     else
1094         except << "No_read_permission_for_File_" <<
            filename << "\"";
1095         throw GnuplotException( except.str() );
1096         return *this;
1097     }
1098
1099     std::ostringstream cmdstr;
1100
1101     // command to be sent to gnuplot
1102     if (this->nplots > 0 && this->two_dim == true)
1103         cmdstr << "replot_";
1104     else
1105         cmdstr << "plot_";
1106
1107     cmdstr << "\" " << filename << "\"_using_" << column_x << ":"
        << column_y;
1108
1109     if (title == "")
1110         cmdstr << "_notitle_";
1111     else
1112         cmdstr << "_title_" << title << "\"_";
1113
1114     cmdstr << "with_" << this->pstyle << ",_" << filename << "
        \"_using_"
1115         << column_x << ":" << column_y << ":" << column_dy <<
            "_notitle_with_errorbars";
1116
1117     // Do the actual plot
1118     this->cmd(cmdstr.str());
1119
1120     return *this;
1121 }
1122
1123 //
-----
1124 // plot x,y pairs with dy errorbars
1125 Gnuplot& Gnuplot::plot_xy_err(const std::vector<double> &x,

```

```

1126         const std::vector<double> &y,
1127         const std::vector<double> &dy,
1128         const std::string &title)
1129 {
1130     if (x.size() == 0 || y.size() == 0 || dy.size() == 0)
1131     {
1132         throw GnuplotException("std::vectors too small");
1133         return *this;
1134     }
1135
1136     if (x.size() != y.size() || y.size() != dy.size())
1137     {
1138         throw GnuplotException("Length of the std::vectors
1139             differs");
1139         return *this;
1140     }
1141
1142     std::ofstream tmp;
1143     std::string name = create_tmpfile(tmp);
1144     if (name == "")
1145         return *this;
1146
1147     // write the data to file
1148     for (unsigned int i = 0; i < x.size(); i++)
1149         tmp << x[i] << " " << y[i] << " " << dy[i] << std::endl;
1150
1151     tmp.flush();
1152     tmp.close();
1153
1154     // Do the actual plot
1155     this->plotfile_xy_err(name, 1, 2, 3, title);
1156
1157     return *this;
1158 }
1159
1160 //
-----
1161 // Plots a 3d graph from a list of doubles (x y z) saved in a
1162 // file
1163 Gnuplot& Gnuplot::plotfile_xyz(const std::string &filename,
1164                                const int column_x,

```

```

1164         const int column_y,
1165         const int column_z,
1166         const std::string &title)
1167 {
1168
1169     // check if file exists
1170     if( !(Gnuplot::file_exists(filename,4)) ) // check existence
        and read permission
1171     {
1172         std::ostringstream except;
1173         if( !(Gnuplot::file_exists(filename,0)) ) // check
            existence
1174             except << "File_\n" << filename << "\n_does_not_exist
                ";
1175         else
1176             except << "No_read_permission_for_File_\n" <<
                filename << "\n";
1177         throw GnuplotException( except.str() );
1178         return *this;
1179     }
1180
1181     std::ostringstream cmdstr;
1182
1183     // command to be sent to gnuplot
1184     if (this->nplots > 0 && this->two_dim == false)
1185         cmdstr << "replot\n";
1186     else
1187         cmdstr << "splot\n";
1188
1189     cmdstr << "\n" << filename << "\n_using\n" << column_x << ":"
        << column_y << ":" << column_z;
1190
1191     if (title == "")
1192         cmdstr << "\nnotitle\nwith\n" << this->pstyle;
1193     else
1194         cmdstr << "\ntitle_\n" << title << "\n_with\n" << this->
            pstyle;
1195
1196     // Do the actual plot
1197     this->cmd(cmdstr.str());
1198
1199     return *this;

```



```

1200 }
1201
1202 //
-----

1203 // Plots a 3d graph from a list of doubles: x y z
1204 Gnuplot& Gnuplot::plot_xyz(const std::vector<double> &x,
1205                             const std::vector<double> &y,
1206                             const std::vector<double> &z,
1207                             const std::string &title)
1208 {
1209     if (x.size() == 0 || y.size() == 0 || z.size() == 0)
1210     {
1211         throw GnuplotException("std::vectors too small");
1212         return *this;
1213     }
1214
1215     if (x.size() != y.size() || x.size() != z.size())
1216     {
1217         throw GnuplotException("Length of the std::vectors
1218                                 differs");
1219         return *this;
1220     }
1221
1222     std::ofstream tmp;
1223     std::string name = create_tmpfile(tmp);
1224     if (name == "")
1225         return *this;
1226
1227     // write the data to file
1228     for (unsigned int i = 0; i < x.size(); i++)
1229     {
1230         tmp << x[i] << " " << y[i] << " " << z[i] << std::endl;
1231     }
1232
1233     tmp.flush();
1234     tmp.close();
1235
1236
1237     this->plotfile_xyz(name, 1, 2, 3, title);
1238

```

```

1239     return *this;
1240 }
1241
1242
1243
1244 //
-----

1245 /// * note that this function is not valid for versions of
      GNUPlot below 4.2
1246 Gnuplot& Gnuplot::plot_image(const unsigned char * ucPicBuf,
1247                               const int iWidth,
1248                               const int iHeight,
1249                               const std::string &title)
1250 {
1251     std::ofstream tmp;
1252     std::string name = create_tmpfile(tmp);
1253     if (name == "")
1254         return *this;
1255
1256     // write the data to file
1257     int iIndex = 0;
1258     for(int iRow = 0; iRow < iHeight; iRow++)
1259     {
1260         for(int iColumn = 0; iColumn < iWidth; iColumn++)
1261         {
1262             tmp << iColumn << "_" << iRow << "_" << static_cast<
                float>(ucPicBuf[iIndex++]) << std::endl;
1263         }
1264     }
1265
1266     tmp.flush();
1267     tmp.close();
1268
1269
1270     std::ostringstream cmdstr;
1271
1272     // command to be sent to gnuplot
1273     if (this->nplots > 0 && this->two_dim == true)
1274         cmdstr << "replot_";
1275     else
1276         cmdstr << "plot_";

```

```

1277
1278     if (title == "")
1279         cmdstr << "\"\" << name << "\"_with_image";
1280     else
1281         cmdstr << "\"\" << name << "\"_title\"\" << title << "\"_
            with_image";
1282
1283     // Do the actual plot
1284     this->cmd(cmdstr.str());
1285
1286     return *this;
1287 }
1288
1289 //
    -----
1290 // Sends a command to an active gnuplot session
1291 Gnuplot& Gnuplot::cmd(const std::string &cmdstr)
1292 {
1293     if( !(this->valid) )
1294     {
1295         return *this;
1296     }
1297
1298     // int fputs ( const char * str, FILE * stream );
1299     // writes the string str to the stream.
1300     // The function begins copying from the address specified (
        str) until it reaches the
1301     // terminating null character ('\0'). This final null-
        character is not copied to the stream.
1302     fputs( (cmdstr+"\n").c_str(), this->gnucmd );
1303
1304     // int fflush ( FILE * stream );
1305     // If the given stream was open for writing and the last i/o
        operation was an output operation,
1306     // any unwritten data in the output buffer is written to the
        file.
1307     // If the argument is a null pointer, all open files are
        flushed.
1308     // The stream remains open after this call.
1309     fflush(this->gnucmd);
1310

```

```

1311
1312     if( cmdstr.find("replot") != std::string::npos )
1313     {
1314         return *this;
1315     }
1316     else if( cmdstr.find("splot") != std::string::npos )
1317     {
1318         this->two_dim = false;
1319         this->nplots++;
1320     }
1321     else if( cmdstr.find("plot") != std::string::npos )
1322     {
1323         this->two_dim = true;
1324         this->nplots++;
1325     }
1326     return *this;
1327 }
1328
1329 //
-----

1330 // Sends a command to an active gnuplot session, identical to cmd
1331 // ()
1332 Gnuplot& Gnuplot::operator<<(const std::string &cmdstr)
1333 {
1334     this->cmd(cmdstr);
1335     return *this;
1336 }
1337 //
-----

1338 // Opens up a gnuplot session, ready to receive commands
1339 void Gnuplot::init()
1340 {
1341     // char * getenv ( const char * name );  get value of an
1342     // environment variable
1343     // Retrieves a C string containing the value of the
1344     // environment variable whose
1345     // name is specified as argument.
1346     // If the requested variable is not part of the environment
1347     // list, the function returns a NULL pointer.

```

```

1345 #if ( defined(unix) || defined(__unix) || defined(__unix__) ) &&
    !defined(__APPLE__)
1346     if (getenv("DISPLAY") == NULL)
1347     {
1348         this->valid = false;
1349         throw GnuplotException("Can't find DISPLAY variable");
1350     }
1351 #endif
1352
1353 // if gnuplot not available
1354 if (!Gnuplot::get_program_path())
1355 {
1356     this->valid = false;
1357     throw GnuplotException("Can't find gnuplot");
1358 }
1359
1360 // open pipe
1361 std::string tmp = Gnuplot::m_sGnuplotPath + "/" + Gnuplot::
    m_sGnuplotFileName;
1362
1363 // FILE *popen(const char *command, const char *mode);
1364 // The popen() function shall execute the command specified
    by the string command,
1365 // create a pipe between the calling program and the executed
    command, and
1366 // return a pointer to a stream that can be used to either
    read from or write to the pipe.
1367 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)
1368     this->gnucmd = _popen(tmp.c_str(), "w");
1369 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
    defined(__APPLE__)
1370     this->gnucmd = popen(tmp.c_str(), "w");
1371 #endif
1372
1373 // popen() shall return a pointer to an open stream that can
    be used to read or write to the pipe.
1374 // Otherwise, it shall return a null pointer and may set
    errno to indicate the error.
1375 if (!this->gnucmd)
1376 {
1377     this->valid = false;

```

```

1378         throw GnuplotException("Couldn't open connection to
                                gnuplot");
1379     }
1380
1381     this->nplots = 0;
1382     this->valid = true;
1383     this->smooth = "";
1384
1385     //set terminal type
1386     this->showonscreen();
1387
1388     return;
1389 }
1390
1391 //
-----
1392 // Find out if a command lives in m_sGnuplotPath or in PATH
1393 bool Gnuplot::get_program_path()
1394 {
1395     // first look in m_sGnuplotPath for Gnuplot
1396     std::string tmp = Gnuplot::m_sGnuplotPath + "/" + Gnuplot::
        m_sGnuplotFileName;
1397
1398 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
        defined(__TOS_WIN__)
1399     if ( Gnuplot::file_exists(tmp,0) ) // check existence
1400 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
        defined(__APPLE__)
1401     if ( Gnuplot::file_exists(tmp,1) ) // check existence and
        execution permission
1402 #endif
1403     {
1404         return true;
1405     }
1406
1407     // second look in PATH for Gnuplot
1408     char *path;
1409     // Retrieves a C string containing the value of the
        environment variable PATH
1410     path = getenv("PATH");
1411

```

```

1412     if (path == NULL)
1413     {
1414         throw GnuplotException("Path_is_not_set");
1415         return false;
1416     }
1417     else
1418     {
1419         std::list<std::string> ls;
1420         //split path (one long string) into list ls of strings
1421 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
        defined(__TOS_WIN__)
1422         strtok(ls,path,";");
1423 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
        defined(__APPLE__)
1424         strtok(ls,path,":");
1425 #endif
1426         // scan list for Gnuplot program files
1427         for (std::list<std::string>::const_iterator i = ls.begin
            (); i != ls.end(); ++i)
1428         {
1429             tmp = (*i) + "/" + Gnuplot::m_sGnuplotFileName;
1430 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
        defined(__TOS_WIN__)
1431             if ( Gnuplot::file_exists(tmp,0) ) // check existence
1432 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
        defined(__APPLE__)
1433             if ( Gnuplot::file_exists(tmp,1) ) // check existence
                and execution permission
1434 #endif
1435             {
1436                 Gnuplot::m_sGnuplotPath = *i; // set
                    m_sGnuplotPath
1437                 return true;
1438             }
1439         }
1440
1441         tmp = "Can't_find_gnuplot_neither_in_PATH_nor_in\" +
            Gnuplot::m_sGnuplotPath + "\"";
1442         throw GnuplotException(tmp);
1443
1444         Gnuplot::m_sGnuplotPath = "";
1445         return false;

```

```

1446     }
1447 }
1448
1449 //
-----

1450 // check if file exists
1451 bool Gnuplot::file_exists(const std::string &filename, int mode)
1452 {
1453     if ( mode < 0 || mode > 7)
1454     {
1455         throw std::runtime_error("In function \"Gnuplot::
            file_exists\": mode has to be an integer between 0 and
            7");
1456         return false;
1457     }
1458
1459     // int _access(const char *path, int mode);
1460     // returns 0 if the file has the given mode,
1461     // it returns -1 if the named file does not exist or is not
        accessible in the given mode
1462     // mode = 0 (F_OK) (default): checks file for existence only
1463     // mode = 1 (X_OK): execution permission
1464     // mode = 2 (W_OK): write permission
1465     // mode = 4 (R_OK): read permission
1466     // mode = 6      : read and write permission
1467     // mode = 7      : read, write and execution permission
1468 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
        defined(__TOS_WIN__)
1469     if (_access(filename.c_str(), mode) == 0)
1470 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
        defined(__APPLE__)
1471     if (access(filename.c_str(), mode) == 0)
1472 #endif
1473     {
1474         return true;
1475     }
1476     else
1477     {
1478         return false;
1479     }
1480

```



```

1481 }
1482
1483 //
-----

1484 // Opens a temporary file
1485 std::string Gnuplot::create_tmpfile(std::ofstream &tmp)
1486 {
1487     #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
        defined(__TOS_WIN__)
1488         char name[] = "gnuplotiXXXXXX"; //tmp file in working
            directory
1489     #elif defined(unix) || defined(__unix) || defined(__unix__) ||
        defined(__APPLE__)
1490         char name[] = "/tmp/gnuplotiXXXXXX"; // tmp file in /tmp
1491     #endif
1492
1493     // check if maximum number of temporary files reached
1494     if (Gnuplot::tmpfile_num == GP_MAX_TMP_FILES - 1)
1495     {
1496         std::ostringstream except;
1497         except << "Maximum number of temporary files reached ("
            << GP_MAX_TMP_FILES
1498             << "): cannot open more files" << std::endl;
1499
1500         throw GnuplotException( except.str() );
1501         return "";
1502     }
1503
1504     //int mkstemp(char *name);
1505     // shall replace the contents of the string pointed to by "
        name" by a unique filename,
1506     // and return a file descriptor for the file open for reading
        and writing.
1507     // Otherwise, -1 shall be returned if no suitable file could
        be created.
1508     // The string in template should look like a filename with
        six trailing 'X' s;
1509     // mkstemp() replaces each 'X' with a character from the
        portable filename character set.
1510     // The characters are chosen such that the resulting name
        does not duplicate the name of an existing file at the time

```

```

        of a call to mkstemp()

1511
1512
1513     // open temporary files for output
1514 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
        defined(__TOS_WIN__)
1515     if (_mktemp(name) == NULL)
1516 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
        defined(__APPLE__)
1517     if (mkstemp(name) == -1)
1518 #endif
1519     {
1520         std::ostringstream except;
1521         except << "Cannot create temporary file \" << name << "
            << "\"";
1522         throw GnuplotException(except.str());
1523         return "";
1524     }
1525
1526     tmp.open(name);
1527     if (tmp.bad())
1528     {
1529         std::ostringstream except;
1530         except << "Cannot create temporary file \" << name << "
            << "\"";
1531         throw GnuplotException(except.str());
1532         return "";
1533     }
1534
1535     // Save the temporary filename
1536     this->tmpfile_list.push_back(name);
1537     Gnuplot::tmpfile_num++;
1538
1539     return name;
1540 }

```

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

Listing 6.5: Arquivo de cabeçalho da classe CReservatorioLinearInfinito.

```

1 #ifndef CRESERVATORIOLINEARINFINITO_H_
2 #define CRESERVATORIOLINEARINFINITO_H_
3

```

---

```

4#include "CFormaReservatorio.h"
5
6class CReservatorioLinearInfinito : CFormaReservatorio
7{
8
9    public:
10
11        CReservatorioLinearInfinito(){};
12
13        virtual double LinearInfinito(double _TD);
14
15        ~CReservatorioLinearInfinito(){};
16
17};
18
19#endif

```

---

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

Listing 6.6: Arquivo de implementação da classe CReservatorioLinearInfinito.

---

```

1#define _USE_MATH_DEFINES // define constantes matematicas (
    exemplo Pi = 3.141516...)
2#include <cmath> ////inclui biblioteca matematica padrao com
    funcoes bessell para c++17 ou superior
3
4#include "CReservatorioLinearInfinito.h"
5
6double CReservatorioLinearInfinito::LinearInfinito(double _TD)
7{
8
9    wd = 2.0*(sqrt(_TD/M_PI));
10
11    return wd;
12}

```

---

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

Listing 6.7: Arquivo de cabeçalho da classe CReservatorioLinearManutencao.

---

```

1#ifndef CRESERVATORIOLINEARMANUTENCAO_H_
2#define CRESERVATORIOLINEARMANUTENCAO_H_
3
4#include "CFormaReservatorio.h"

```

```

5
6 class CReservatorioLinearManutencao : CFormaReservatorio
7 {
8
9     public:
10
11         CReservatorioLinearManutencao(){};
12
13         double virtual LinearManutencao(double _u);
14
15         ~CReservatorioLinearManutencao(){};
16
17 };
18
19 #endif

```

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

Listing 6.8: Arquivo de implementação da classe CReservatorioLinearManutencao.

```

1 #define _USE_MATH_DEFINES // define constantes matematicas (
    exemplo Pi = 3.141516...)
2 #include <cmath> ////inclui biblioteca matematica padrao com
    funcoes bessel para c++17 ou superior
3
4 #include "CReservatorioLinearManutencao.h"
5
6 double CReservatorioLinearManutencao::LinearManutencao(double _u)
7 {
8
9     wd = (1.0+exp(-2.0*sqrt(_u)))/((pow(_u, (3.0/2.0)))*(1.0-
        exp(-2.0*sqrt(_u))));
10
11     return wd;
12 }

```

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

Listing 6.9: Arquivo de cabeçalho da classe CReservatorioLinearSelado.

```

1 #ifndef CRESERVATORIOLINEARSELADO_H_
2 #define CRESERVATORIOLINEARSELADO_H_
3
4 #include "CFormaReservatorio.h"

```

```

5
6 class CReservatorioLinearSelado : CFormaReservatorio
7 {
8
9     public:
10
11         CReservatorioLinearSelado(){};
12
13         virtual double LinearSelado(double _u);
14
15         ~CReservatorioLinearSelado(){};
16 };
17
18 #endif

```

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

Listing 6.10: Arquivo de implementação da classe CReservatorioLinearSelado.

```

1 #define _USE_MATH_DEFINES // define constantes matematicas (
    exemplo Pi = 3.141516...)
2 #include <cmath> //inclui biblioteca matematica padrao com
    funcoes bessel para c++17 ou superior
3
4 #include "CReservatorioLinearSelado.h"
5
6 double CReservatorioLinearSelado::LinearSelado(double _u)
7 {
8
9     wd = (1.0- exp(-2.0*sqrt(_u)))/ ((pow(_u, (3.0/2.0)))
        *(1.0+exp(-2.0*sqrt(_u))));
10
11     return wd;
12 }

```

Apresenta-se na listagem 6.11 o arquivo com código da classe CReservatorioRadialInfinito.

Listing 6.11: Arquivo de cabeçalho da classe CReservatorioRadialInfinito.

```

1 #ifndef CRESERVATORIORADIALINFINITO_H_
2 #define CRESERVATORIORADIALINFINITO_H_
3
4 #include "CFormaReservatorio.h"
5

```

```

6//Criacao da classe CReservatorioRadialInfinito.h
7
8class CReservatorioRadialInfinito : CFormaReservatorio
9{
10
11    // declaracao metodos publicos
12    public:
13
14
15    CReservatorioRadialInfinito(){}; //Construtor default
16
17    virtual double RadialInfinito(double _u, double _Rd);
18
19    ~CReservatorioRadialInfinito(){}; //Destrutor default
20
21};
22
23#endif

```

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

Listing 6.12: Arquivo de implementação da classe CReservatorioRadialInfinito.

```

1#define _USE_MATH_DEFINES // define constantes matematicas (
    exemplo Pi = 3.141516...)
2#include <cmath> ////inclui biblioteca matematica padrao com
    funcoes bessel para c++17 ou superior
3
4#include "CReservatorioRadialInfinito.h"
5
6using namespace std;
7
8//Forma da equação solucao do modelo de van everdinger para
    reservaatorio infinito e radial
9
10double CReservatorioRadialInfinito::RadialInfinito(double _u,
    double _Rd)
11{
12
13    wd = cyl_bessel_k(1, sqrt(_u))/((pow(_u,(3.0/2.0)))*
        cyl_bessel_k(0, sqrt(_u)));
14
15    return wd;

```

16  
17}

Apresenta-se na listagem 6.13 o arquivo com código da classe CReservatorioRadialManutencao.

Listing 6.13: Arquivo de cabeçalho da classe CReservatorioRadialManutencao.

---

```

1 #ifndef CRESERVATORIORADIALMANUTENCAO_H_
2 #define CRESERVATORIORADIALMANUTENCAO_H_
3
4 #include "CFormaReservatorio.h"
5
6 class CReservatorioRadialManutencao : CFormaReservatorio
7 {
8
9     public:
10
11         CReservatorioRadialManutencao(){};
12
13         virtual double RadialManutencao(double _u, double
14             _RD);
15
16         ~CReservatorioRadialManutencao(){};
17 };
18
19 #endif

```

---

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

Listing 6.14: Arquivo de implementação da classe CReservatorioRadialManutencao.

---

```

1 #define _USE_MATH_DEFINES // define constantes matematicas (
    exemplo Pi = 3.141516...)
2 #include <cmath> //inclui biblioteca matematica padrao com
    funcoes bessell para c++17 ou superior
3
4 #include "CReservatorioRadialManutencao.h"
5
6 using namespace std;
7
8 double CReservatorioRadialManutencao::RadialManutencao(double _u,
    double _RD)
9 {

```

---

---

```

10
11     wd = (((cyl_bessel_i(0, _RD*sqrt(_u))*cyl_bessel_k(1,
        sqrt(_u)))+(cyl_bessel_i(1, sqrt(_u))*cyl_bessel_k(0,
        _RD*sqrt(_u))))/((pow(_u, (3.0/2.0))*((cyl_bessel_i(0,
        _RD*sqrt(_u))*cyl_bessel_k(0, sqrt(_u)))-(cyl_bessel_i
        (0, sqrt(_u))*cyl_bessel_k(0, sqrt(_u))))));
12
13     return wd;
14
15 }

```

---

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

Listing 6.15: Arquivo de cabeçalho da classe CReservatorioRadialSelado.

---

```

1 #ifndef CRESERVATORIORADIALSELADO_H_
2 #define CRESERVATORIORADIALSELADO_H_
3
4 #include "CFormaReservatorio.h"
5
6 class CReservatorioRadialSelado : CFormaReservatorio
7 {
8
9     public:
10
11         CReservatorioRadialSelado(){};
12
13         virtual double RadialSelado(double _u, double _RD
14             );
15
16         ~CReservatorioRadialSelado(){};
17
18 };
19
20
21 #endif

```

---

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

Listing 6.16: Arquivo de implementação da classe CReservatorioRadialSelado.

---

```

1 #define _USE_MATH_DEFINES // define constantes matematicas (
    exemplo Pi = 3.141516...)

```

---



---

```

2#include <cmath> //incluir biblioteca matematica padrao com
   funcoes essel para c++17 ou superior
3
4#include "CReservatorioRadialSelado.h"
5
6using namespace std;
7
8double CReservatorioRadialSelado::RadialSelado(double _u, double
   _RD)
9{
10
11    wd = (((cyl_bessel_i(1, _RD*sqrt(_u))*cyl_bessel_k(1,
        sqrt(_u)))- (cyl_bessel_i(1, sqrt(_u))*cyl_bessel_k(1,
        _RD*sqrt(_u))))/((pow(_u, (3.0/2.0)))*((cyl_bessel_i(0,
        sqrt(_u))*cyl_bessel_k(1, _RD*sqrt(_u)))+(cyl_bessel_i
        (1, _RD*sqrt(_u))*cyl_bessel_k(0, sqrt(_u))))));
12
13
14    return wd;
15}

```

---

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

Listing 6.17: Arquivo de cabeçalho da classe CSolverVanEverdingen.

---

```

1#ifndef CSOLVERVANEVERDINGEN_H_
2#define CSOLVERVANEVERDINGER_H_
3
4#include <vector>
5#include <iostream>
6#include <string>
7
8#include "CStehfest.h"
9#include "CGnuplot.h"
10
11class CSolverVanEverdingen
12{
13
14    protected:
15
16        CFormaReservatorio forma;
17        CStehfest Stehfest;
18        Gnuplot plot, plot2, plot3;
19

```

```

20         std::vector <double> TD, ReD;
21
22     public:
23
24         CSolverVanEverdingen(){};
25
26         void Solver();
27         void Plot(std::vector <double> _WD, std::vector <
28             double> _TD, std::string name, bool _setY);
29         void Plot2(std::vector <double> _WD, std::vector
30             <double> _TD, std::string name, bool _setY);
31         void Plot3(std::vector <double> _WD, std::vector
32             <double> _TD, std::string name);
33
34         ~CSolverVanEverdingen(){};
35
36 #endif

```

Apresenta-se na listagem 6.18 o arquivo de implementação da classe CSolverVanEverdingen.

Listing 6.18: Arquivo de implementação da classe CSolverVanEverdingen.

```

1 #include "CSolverVanEverdingen.h"
2
3 using namespace std;
4
5 void CSolverVanEverdingen::Plot(vector <double> _WD, vector <
6     double> _TD, std::string name, bool _setY)
7 {
8     Gnuplot::Terminal("qt");
9
10    if (_setY)
11        plot.set_yrange(0, 50);
12
13    plot.Grid();
14    plot.set_xlabel("Td");
15    plot.set_ylabel("Wd");
16    plot.Title("Influxo_acumulado_Adimensional_x_Tempo_Adimensional");

```

```

17         plot.set_xlogscale();
18         plot.Legend("inside_left_top_box");
19         plot.ShowOnScreen();
20         plot.plot_xy(_TD, _WD, name);
21         plot.savetops(name);
22         cout << "Aperte ENTER para continuar" << endl;
23         cin.get();
24
25
26     }
27
28 void CSolverVanEverdingen::Plot2(vector<double> _WD, vector<
    double> _TD, std::string name, bool _setY)
29 {
30
31     Gnuplot::Terminal("qt");
32
33     if (_setY)
34         plot2.set_yrange(0, 300);
35
36     plot2.Grid();
37     plot2.set_xlabel("Td");
38     plot2.set_ylabel("Wd");
39     plot2.Title("Influxo_acumulado_Adimensional_x_Tempo_Adimensional");
40     plot2.set_xlogscale();
41     plot2.Legend("inside_left_top_box");
42     plot2.ShowOnScreen();
43     plot2.plot_xy(_TD, _WD, name);
44     plot2.savetops(name);
45     cout << "Aperte ENTER para continuar" << endl;
46     cin.get();
47
48
49 }
50
51 void CSolverVanEverdingen::Plot3(vector<double> _WD, vector<
    double> _TD, std::string name)
52 {
53
54     Gnuplot::Terminal("qt");
55

```

```

56     plot3.Grid();
57     plot3.set_xlabel("Td");
58     plot3.set_ylabel("Wd");
59     plot3.Title("Inflxo_acumulado_Adimensional_x_Tempo_Adimensional");
60     plot3.set_xlogscale();
61     plot3.set_ylogscale();
62     plot3.Legend("inside_left_top_box");
63     plot3.ShowOnScreen();
64     plot3.plot_xy(_TD, _WD, name);
65     plot3.savetops(name);
66     cout << "Aperte ENTER para continuar" << endl;
67     cin.get();
68
69
70 }
71
72 void CSolverVanEverdingen::Solver()
73 {
74
75     vector <double> radialinfinito, radialselado2,
        radialselado3, radialselado4, radialselado5,
        radialselado6, radialselado7, radialselado8,
        radialselado9, radialselado10, radialManutencao2,
        radialManutencao3, radialManutencao4, radialManutencao5,
        radialManutencao6, radialManutencao7,
        radialManutencao8, radialManutencao9,
        radialManutencao10;
76     vector <double> linearinfinito, linearmanutencao,
        linearselado;
77
78     for (double i=0.01; i<=1000; i+=.01)
79         TD.push_back(i);
80
81
82     for (double k=0; k < TD.size(); k++)
83         radialinfinito.push_back(Stehfest.
            StehfestRadialInfinito(TD[k]));
84
85     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;

```

```

86     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        "        %" << endl;
87     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        "        criando_plot_radial_infinito        %" << endl;
88     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        "        %" << endl;
89     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;

90
91
92         Plot(radialinfinito, TD, "RadialInfinito", 1);
93
94         for (double k=0; k < TD.size(); k++)
95             radialselado2.push_back(Stehfest.
                StehfestRadialSelado(TD[k], 2));
96
97     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
98     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        "        %" << endl;
99     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        "        criando_plot_radial_selado_para_ReD=2        %" << endl;
100    cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        "        %" << endl;
101    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;

102
103
104        Plot(radialselado2, TD, "RadialSelado-2-ReD=2",
            0);
105
106        for (double k=0; k < TD.size(); k++)
107            radialselado3.push_back(Stehfest.
                StehfestRadialSelado(TD[k], 3));
108
109    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
110    cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"

```

```

        "        " << endl;
111    cout << "%        criando_plot_radial_selado_para_ReD=_3        "
        "        " << endl;
112    cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        "        " << endl;
113    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;

114
115
116        Plot(radialselado3, TD, "RadialSelado_-ReD=_3",
            0);
117    for (double k=0; k < TD.size(); k++)
118        radialselado4.push_back(Stehfest.
            StehfestRadialSelado(TD[k], 4));
119
120    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
121    cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        "        " << endl;
122    cout << "%        criando_plot_radial_selado_para_ReD=_4        "
        "        " << endl;
123    cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        "        " << endl;
124    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;

125
126
127        Plot(radialselado4, TD, "RadialSelado_-ReD=_4",
            0);
128    for (double k=0; k < TD.size(); k++)
129        radialselado5.push_back(Stehfest.
            StehfestRadialSelado(TD[k], 5));
130
131    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
132    cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        "        " << endl;
133    cout << "%        criando_plot_radial_selado_para_ReD=_5        "

```

```

        "    " << endl;
134    cout << "%    " << endl;
        "    " << endl;
135    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;

136
137
138        Plot(radialselado5, TD, "RadialSelado-Red=5",
            0);

139
140        for (double k=0; k < TD.size(); k++)
141            radialselado6.push_back(Stehfest.
                StehfestRadialSelado(TD[k], 6));

142
143    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
144    cout << "%    " << endl;
        "    " << endl;
145    cout << "%    criando_plot_radial_selado_para_Red=6    "
        "    " << endl;
146    cout << "%    " << endl;
        "    " << endl;
147    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;

148
149
150        Plot(radialselado6, TD, "RadialSelado-Red=6",
            0);
151        for (double k=0; k < TD.size(); k++)
152            radialselado7.push_back(Stehfest.
                StehfestRadialSelado(TD[k], 7));

153
154    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
155    cout << "%    " << endl;
        "    " << endl;
156    cout << "%    criando_plot_radial_selado_para_Red=7    "
        "    " << endl;

```

```

157     cout << "%\n" << endl;
158     cout << "
        %\n" << endl;

159
160
161         Plot(radialselado7, TD, "RadialSelado-Red=7",
            0);

162
163     for (double k=0; k < TD.size(); k++)
164         radialselado8.push_back(Stehfest.
            StehfestRadialSelado(TD[k], 8));

165
166     cout << "
        %\n" << endl;
167     cout << "%\n" << endl;
168     cout << "%criandoplotradialseladoparaRed=8\n" << endl;
169     cout << "%\n" << endl;
170     cout << "
        %\n" << endl;

171
172
173         Plot(radialselado8, TD, "RadialSelado-Red=8",
            0);

174     for (double k=0; k < TD.size(); k++)
175         radialselado9.push_back(Stehfest.
            StehfestRadialSelado(TD[k], 9));

176
177     cout << "
        %\n" << endl;
178     cout << "%\n" << endl;
179     cout << "%criandoplotradialseladoparaRed=9\n" << endl;
180     cout << "%\n" << endl;

```



```

        "%%%" << endl;
181    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;

182
183
184        Plot(radialselado9, TD, "RadialSelado-Red=9",
            0);
185        for (double k=0; k < TD.size(); k++)
186            radialselado10.push_back(Stehfest.
                StehfestRadialSelado(TD[k], 10));

187
188    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
189    cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        " << endl;
190    cout << "%criando plot radial selado para Red=10
        " << endl;
191    cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        " << endl;
192    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;

193
194
195        Plot(radialselado10, TD, "RadialSelado-Red=10
            ", 0);

196
197
198    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
199    cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        " << endl;
200    cout << "%Criando novo terminal do gnuplot
        " << endl;
201    cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        " << endl;
202    cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;

```

```

203
204         Plot2(radialinfinito, TD, "RadialInfinito", 1);
205
206         for (double k=0; k < TD.size(); k++)
207             radialManutencao2.push_back(Stehfest.
                StehfestRadialManutencao(TD[k], 2));
208
209         cout << "
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
                << endl;
210         cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %%%%%%%%%%" << endl;
211         cout << "%%%%%%%%%criando_plot_radial_realimentado_para_ReD=
                %2%%%%%%%%%" << endl;
212         cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %%%%%%%%%%" << endl;
213         cout << "
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
                << endl;
214
215         Plot2(radialManutencao2, TD, "RadialManutencao_-
                ReD=%2", 0);
216
217         for (double k=0; k < TD.size(); k++)
218             radialManutencao3.push_back(Stehfest.
                StehfestRadialManutencao(TD[k], 3));
219
220         cout << "
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
                << endl;
221         cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %%%%%%%%%%" << endl;
222         cout << "%%%%%%%%%criando_plot_radial_realimentado_para_ReD=
                %3%%%%%%%%%" << endl;
223         cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %%%%%%%%%%" << endl;
224         cout << "
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
                << endl;
225
226         Plot2(radialManutencao3, TD, "RadialManutencao_-ReD=%3"
                , 0);

```

```

227
228         for (double k=0; k < TD.size(); k++)
229             radialManutencao4.push_back(Stehfest.
                StehfestRadialManutencao(TD[k], 4));
230
231     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
232     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%" << endl;
233     cout << "%criando_plot_radial_realimentado_para_ReD=
        4%" << endl;
234     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%" << endl;
235     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
236
237     Plot2(radialManutencao4, TD, "RadialManutencao- ReD=4"
        , 0);
238
239         for (double k=0; k < TD.size(); k++)
240             radialManutencao5.push_back(Stehfest.
                StehfestRadialManutencao(TD[k], 5));
241
242     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
243     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%" << endl;
244     cout << "%criando_plot_radial_realimentado_para_ReD=
        5%" << endl;
245     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%" << endl;
246     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
247
248     Plot2(radialManutencao5, TD, "RadialManutencao- ReD=5"
        , 0);
249
250         for (double k=0; k < TD.size(); k++)

```

```

251         radialManutencao6.push_back(Stehfest.
                StehfestRadialManutencao(TD[k], 6));
252
253     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
254     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%" << endl;
255     cout << "%criando plot radial realimentado para ReD=
        6%" << endl;
256     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%" << endl;
257     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
258
259     Plot2(radialManutencao6, TD, "RadialManutencao- ReD=6"
        , 0);
260
261     for (double k=0; k < TD.size(); k++)
262         radialManutencao7.push_back(Stehfest.
                StehfestRadialManutencao(TD[k], 7));
263
264     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
265     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%" << endl;
266     cout << "%criando plot radial realimentado para ReD=
        7%" << endl;
267     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%" << endl;
268     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
        << endl;
269
270     Plot2(radialManutencao7, TD, "RadialManutencao- ReD=7"
        , 0);
271
272     for (double k=0; k < TD.size(); k++)
273         radialManutencao8.push_back(Stehfest.
                StehfestRadialManutencao(TD[k], 8));

```

```

274
275     cout << "
           %%%%%%%%%%%"
           << endl;
276     cout << "%%%%%%%%%%%
           %%%%" << endl;
277     cout << "%criando_plot_radial_realimentado_para_ReD=
           8%" << endl;
278     cout << "%%%%%%%%%%%
           %%%%" << endl;
279     cout << "
           %%%%%%%%%%%"
           << endl;
280
281     Plot2(radialManutencao8, TD, "RadialManutencao- ReD=8"
           , 0);
282
283         for (double k=0; k < TD.size(); k++)
284             radialManutencao9.push_back(Stehfest.
                StehfestRadialManutencao(TD[k], 9));
285
286     cout << "
           %%%%%%%%%%%"
           << endl;
287     cout << "%%%%%%%%%%%
           %%%%" << endl;
288     cout << "%criando_plot_radial_realimentado_para_ReD=
           9%" << endl;
289     cout << "%%%%%%%%%%%
           %%%%" << endl;
290     cout << "
           %%%%%%%%%%%"
           << endl;
291
292     Plot2(radialManutencao9, TD, "RadialManutencao- ReD=9"
           , 0);
293
294         for (double k=0; k < TD.size(); k++)
295             radialManutencao10.push_back(Stehfest.
                StehfestRadialManutencao(TD[k], 10));
296
297     cout << "

```

```

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% "
        << endl;
298     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%" << endl;
299     cout << "%criando_plot_radial_realimentado_para_ReD=
        %10%" << endl;
300     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%" << endl;
301     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% "
        << endl;
302
303     Plot2(radialManutencao10, TD, "RadialManutencao- ReD=
        10", 0);
304
305     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% "
        << endl;
306     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%" << endl;
307     cout << "%Criando_novo_terminal_do_gnuplot
        %%%%%%%%%%" << endl;
308     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%" << endl;
309     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% "
        << endl;
310
311     for (double k=0; k < TD.size(); k++)
312         linearinfinito.push_back(Stehfest.
            StehfestLinearInfinito(TD[k]));
313
314     cout << "
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% "
        << endl;
315     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%" << endl;
316     cout << "%criando_plot_linear_infinito
        %%%%%%%%%%" << endl;
317     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%" << endl;
318     cout << "

```

```

318         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% "
319         << endl;
320     Plot3(linearinfinito, TD, "Linear_Infinito");
321
322     for (double k=0; k < TD.size(); k++)
323         linearselado.push_back(Stehfest.
324             StehfestLinearSelado(TD[k]));
325
326     cout << "
327         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% "
328         << endl;
329     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
330         %%%%%%%%%%" << endl;
331     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%criando_plot_linear_selado%%%%%%%%%%
332         %%%%%%%%%%" << endl;
333     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%
334         %%%%%%%%%%" << endl;
335     cout << "
336         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% "
337         << endl;
338     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%
339         %%%%%%%%%%" << endl;
340     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%criando_plot_linear_realimentado%%%%%%%%%%
341         %%%%%%%%%%" << endl;
342     cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%
343         %%%%%%%%%%" << endl;
344     Plot3(linearmanutencao, TD, "Linear_com_manutencao_de_

```

```

        pressao");
343
344        cout << "
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
            << endl;
345        cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%" << endl;
346        cout << "%Todos arquivos de saida estao salvos na pasta
            ./Src%" << endl;
347        cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%" << endl;
348        cout << "% Pressione ENTER para destruir arquivos
            temporarios%" << endl;
349        cout << "%%%%%%%%%%%%%%%%%%%%%%%%% e encerrar programa... %%%%%%%%%%%
            %%%%%%%%%%" << endl;
350        cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%" << endl;
351        cout << "
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
            << endl;
352
353        cin.get();
354
355 }

```

Apresenta-se na listagem 6.19 o arquivo com código da classe CStehfest.

Listing 6.19: Arquivo de cabeçalho da classe CStehfest.

```

1 #ifndef CSTEHFEST_H_
2 #define CSTEHFEST_H_
3
4 #include <vector>
5
6 #include "CReservatorioRadialInfinito.h"
7 #include "CReservatorioRadialSelado.h"
8 #include "CReservatorioRadialManutencao.h"
9 #include "CReservatorioLinearInfinito.h"
10 #include "CReservatorioLinearSelado.h"
11 #include "CReservatorioLinearManutencao.h"
12
13 class CStehfest
14 {
15

```



```

16         protected:
17
18             double nn2, nn21, z, ln2_on_t, sum, p, ilt, L, Rd
19                 ;
19             std::vector <double> v;
20             CReservatorioRadialInfinito radialinfinito;
21             CReservatorioRadialSelado radialselado;
22             CReservatorioRadialManutencao radialmanutencao;
23             CReservatorioLinearInfinito linearinfinito;
24             CReservatorioLinearSelado linearselado;
25             CReservatorioLinearManutencao linearmanutencao;
26
27         public:
28
29             CStehfest(double _RD = 1, double _L = 12) : L(_L)
30                 , Rd(_RD){};
31
32             double Factorial(int _i);
33             double StehfestRadialInfinito(double _TD);
34             double StehfestRadialSelado (double _TD, double
35                 _RD);
36             double StehfestRadialManutencao (double _TD,
37                 double _RD);
38             double StehfestLinearInfinito (double _TD);
39             double StehfestLinearSelado (double _TD);
40             double StehfestLinearManutencao (double _TD);
41
42             ~CStehfest(){};
43 };
44
45 #endif

```

Apresenta-se na listagem 6.20 o arquivo de implementação da classe CStehfest.

Listing 6.20: Arquivo de implementação da classe CStehfest.

```

1 #define _USE_MATH_DEFINES // define constantes matematicas (
    exemplo Pi = 3.141516...)
2 #include <cmath> //inclui biblioteca matematica padrao com
    funcoes bessel para c++17 ou superior
3
4 #include "CStehfest.h"
5 #include <iostream> //Biblioteca std usada somente pra dar cout e
    encontrar bugs vai ser apagada na versão final

```

```

6 #include <algorithm> // usar min(a, b)
7
8 using namespace std;
9
10 double CStehfest::Factorial(int _i)
11 {
12     double acumulador = 1;
13     if (_i==0 || _i==1)
14         return acumulador;
15     else
16     {
17         for (double j = 1; j <= _i; j++)
18             acumulador *=j;
19     }
20     return acumulador;
21 }
22
23 double CStehfest::StehfestRadialInfinito(double _TD)
24 {
25
26 nn2 = L/2.0;
27 nn21= nn2+1.0;
28
29 for (double n=1 ; n<=L;n++)
30 {
31     z=0.0;
32     for (int k = floor( ( n + 1.0 ) / 2.0 ); k <= min(n,nn2);
33         k++)
34         z = z + ((pow(k, nn2))*Factorial(2*k))/(
35             Factorial(nn2-k)*Factorial(k)*Factorial(k-1)*
36             Factorial(n-k)*Factorial(2*k - n));
37
38 v.push_back(pow((-1.0),(n+nn2))*z);
39
40 }
41 sum = 0.0;
42 ln2_on_t = log(2.0) / _TD;
43
44 for (double n = 1.0; n <= L; n++)
45 {
46     p = n * ln2_on_t;
47     sum = sum + v[n-1.0]*radialinfinito.RadialInfinito(p, Rd);
48 }

```

```

45     }
46         ilt = sum * ln2_on_t;
47         return ilt;
48     }
49
50 double CStehfest::StehfestRadialSelado (double _TD, double _RD)
51 {
52
53     nn2 = L/2.0;
54     nn21= nn2+1.0;
55
56     for (double n=1 ; n<=L;n++)
57     {
58         z=0.0;
59         for (int k = floor( ( n + 1.0 ) / 2.0 ); k <= min(n,nn2);
              k++)
60             z = z + ((pow(k, nn2))*Factorial(2*k))/(
                    Factorial(nn2-k)*Factorial(k)*Factorial(k-1)*
                    Factorial(n-k)*Factorial(2*k - n));
61
62         v.push_back(pow((-1.0),(n+nn2))*z);
63
64     }
65     sum = 0.0;
66     ln2_on_t = log(2.0) / _TD;
67
68     for (double n = 1.0; n <= L; n++)
69     {
70         p = n * ln2_on_t;
71         sum = sum + v[n-1.0]*radialselado.RadialSelado(p, _RD);
72     }
73
74     ilt = sum * ln2_on_t;
75     return ilt;
76 }
77 double CStehfest::StehfestRadialManutencao (double _TD, double
       _RD)
78 {
79
80     nn2 = L/2.0;
81     nn21= nn2+1.0;
82

```

```

83 for (double n=1 ; n<=L;n++)
84 {
85     z=0.0;
86     for (int k = floor( ( n + 1.0 ) / 2.0 ); k <= min(n,nn2);
          k++)
87         z = z + ((pow(k, nn2))*Factorial(2*k))/(
          Factorial(nn2-k)*Factorial(k)*Factorial(k-1)*
          Factorial(n-k)*Factorial(2*k - n));
88
89     v.push_back(pow((-1.0),(n+nn2))*z);
90
91 }
92 sum = 0.0;
93 ln2_on_t = log(2.0) / _TD;
94
95 for (double n = 1.0; n <= L; n++)
96 {
97     p = n * ln2_on_t;
98     sum = sum + v[n-1.0]*radialmanutencao.RadialManutencao(p, _RD
99         );
100 }
101     ilt = sum * ln2_on_t;
102     return ilt;
103 }
104 double CStehfest::StehfestLinearInfinito (double _TD)
105 {
106
107     ilt = linearinfinito.LinearInfinito(_TD);
108     return ilt;
109
110 }
111
112 double CStehfest::StehfestLinearSelado(double _TD)
113 {
114
115     nn2 = L/2.0;
116     nn21= nn2+1.0;
117
118     for (double n=1 ; n<=L;n++)
119     {
120         z=0.0;

```

```

121         for (int k = floor( ( n + 1.0 ) / 2.0 ); k <= min(n,nn2);
              k++)
122             z = z + ((pow(k, nn2))*Factorial(2*k))/(
                    Factorial(nn2-k)*Factorial(k)*Factorial(k-1)*
                    Factorial(n-k)*Factorial(2*k - n));
123
124     v.push_back(pow((-1.0),(n+nn2))*z);
125
126 }
127 sum = 0.0;
128 ln2_on_t = log(2.0) / _TD;
129
130 for (double n = 1.0; n <= L; n++)
131 {
132     p = n * ln2_on_t;
133     sum = sum + v[n-1.0]*linearselado.LinearSelado(p);
134 }
135     ilt = sum * ln2_on_t;
136     return ilt;
137 }
138
139 double CStehfest::StehfestLinearManutencao(double _TD)
140 {
141
142     nn2 = L/2.0;
143     nn21= nn2+1.0;
144
145     for (double n=1 ; n<=L;n++)
146     {
147         z=0.0;
148         for (int k = floor( ( n + 1.0 ) / 2.0 ); k <= min(n,nn2);
              k++)
149             z = z + ((pow(k, nn2))*Factorial(2*k))/(
                    Factorial(nn2-k)*Factorial(k)*Factorial(k-1)*
                    Factorial(n-k)*Factorial(2*k - n));
150
151         v.push_back(pow((-1.0),(n+nn2))*z);
152
153     }
154     sum = 0.0;
155     ln2_on_t = log(2.0) / _TD;
156

```

---

```

157 for (double n = 1.0; n <= L; n++)
158 {
159     p = n * ln2_on_t;
160     sum = sum + v[n-1.0]*linearmanutencao.LinearManutencao(p);
161 }
162     ilt = sum * ln2_on_t;
163     return ilt;
164 }

```

---

Apresenta-se na listagem 6.40 o programa que usa a classe main.

Listing 6.21: Arquivo de implementação da função main().

---

```

1 #include "CSolverVanEverdingen.h"
2
3
4 int main(void){
5
6     CSolverVanEverdingen executa;
7
8     executa.Solver();
9
10    return 0;
11 }

```

---

Apresenta-se na listagem 6.22 o arquivo com código da classe CAdimensional.

Listing 6.22: Arquivo de cabeçalho da classe CAdimensional.

---

```

1 #ifndef CADIMENSIONAL_H_
2 #define CADIMENSIONAL_H_
3
4 #include <vector>
5
6 class CAdimensional{
7
8     protected:
9
10         std::vector <double> delT, delTpbarra, TD, delP,
11             TDA, PD, PDbarra, WE;
12         double U, J, We, Wei;
13
14     public:
15
16         CAdimensional(){};

```

```

17         std::vector <double> DeltaT(std::vector <double>
           _t);
18         std::vector <double> DeltaPbarra(std::vector <
           double> _Pbarra);
19         std::vector <double> CalcTD(double _k, double
           _phi, double _mi, double _ct, double _Ro, std
           ::vector <double> _t);
20         std::vector <double> CalcdelP(std::vector <double
           > _P);
21         std::vector <double> CalcTDA(double _Ro, double
           _Re, std::vector <double> _TD);
22         std::vector <double> CalcPD(std::vector <double>
           _TDA, std::vector <double> _TD, double _Re,
           double _Ro);
23         std::vector <double> CalcPDbarra(std::vector <
           double> _TDA, std::vector <double> _TD, double
           _Re, double _Ro);
24         double CalcU(double _phi, double _ct, double _h,
           double _Ro);
25         double CalcJ(double _k, double _h, double _Re,
           double _Ro, double _mi, bool _regime);
26         double CalcWe(double _Re, double _Ro, double _h,
           double _phi);
27         double CalcWei(double _ct, double _WI, double _Pi
           );
28
29         ~CAdimensional(){};
30
31     };
32
33 #endif

```

Apresenta-se na listagem 6.23 o arquivo de implementação da classe CAdimensional.

Listing 6.23: Arquivo de implementação da classe CAdimensional.

```

1 #include "CAdimensional.h"
2
3 #define _USE_MATH_DEFINES
4
5 #include <cmath>
6
7 using namespace std;
8

```

```

9 vector <double> CAdimensional::DeltaT(vector <double> _t)
10 {
11
12     double delt;
13
14     for (int i = 0; i < _t.size(); i++)
15     {
16         if (i==0)
17             delt = 0;
18         else
19             delt = _t[i] - _t[i-1];
20
21         delT.push_back(delt);
22     }
23
24     return delT;
25
26 }
27
28 std::vector <double> CAdimensional::DeltaPbarra(std::vector <
    double> _Pbarra)
29 {
30     double pbarra;
31
32     for (int i = 0; i < _Pbarra.size(); i++)
33     {
34
35         if (i==0)
36             pbarra = 0;
37         else
38             pbarra = (_Pbarra[i-1] + _Pbarra[i])/2.0;
39
40         deltPbarra.push_back(pbarra);
41     }
42
43     return deltPbarra;
44
45 }
46
47 vector <double> CAdimensional::CalcTD(double _k, double _phi,
    double _mi, double _ct, double _Ro, vector <double> _t)
48 {

```



```

49
50     double td;
51
52     for (int i=0; i < _t.size(); i++)
53     {
54         td = (_k*_t[i])/(_phi*_mi*_ct*(_Ro*_Ro));
55         TD.push_back(0.008362*td);
56     }
57
58     return TD;
59
60 }
61
62 vector <double> CAdimensional::CalcdelP(vector <double> _P)
63 {
64
65     double prsI = _P[0], A;
66
67     for (int i = 0; i < _P.size(); i++)
68     {
69         A = prsI - _P[i];
70         delP.push_back(A);
71     }
72
73     return delP;
74
75 }
76
77 vector <double> CAdimensional::CalcTDA(double _Ro, double _Re,
78     vector <double> _TD)
79 {
80
81     double tda;
82
83     for (int i = 0; i < _TD.size(); i++)
84     {
85         tda = (_TD[i])/(M_PI*((( _Re/_Ro)*(_Re/_Ro)) - 1))
86         ;
87         TDA.push_back(tda);
88     }
89
90     return TDA;

```

```

89
90}
91
92vector <double> CAdimensional::CalcPD(vector <double> _TDA,
    vector <double> _TD, double _Re, double _Ro)
93{
94    double pd;
95
96    for (int i = 0; i<_TD.size();i++)
97    {
98
99        if (_TDA[i]<.1)
100            pd = (1.0/2.0)*(log(_TD[i])+.80907);
101        else
102            pd = ((2.0*_TD[i])/((_Re/_Ro)*(_Re/_Ro)))
                + log(_Re/_Ro) -(3.0/4.0);
103
104        PD.push_back(pd);
105    }
106
107    return PD;
108
109}
110
111vector <double> CAdimensional::CalcPDbarra(vector <double> _TDA,
    vector <double> _TD, double _Re, double _Ro)
112{
113    double pdbarra;
114
115    for (int i = 0; i<_TD.size();i++)
116    {
117
118        if (_TDA[i]<.1)
119            pdbarra = (1.0/(2.0*_TD[i]));
120        else
121            pdbarra = 2.0/((_Re/_Ro)*(_Re/_Ro));
122
123        PDbarra.push_back(pdbarra);
124    }
125
126    return PDbarra;
127

```

```

128 }
129
130 double CAdimensional::CalcU(double _phi, double _ct, double _h,
    double _Ro)
131 {
132
133     U=2.0*M_PI*_phi*_ct*_h*_Ro*_Ro;
134
135     return U;
136
137 }
138
139 double CAdimensional::CalcJ(double _k, double _h, double _Re,
    double _Ro, double _mi, bool _regime)
140 {
141     double A, B;
142
143     if (_regime)
144     {
145         A = 0.05255*_k*_h;
146         B = _mi*(log(_Re/_Ro));
147         J = A/B;
148     }
149     else
150     {
151         A = 0.05255*_k*_h;
152         B = _mi*(log(_Re/_Ro) - (3.0/4.0));
153         J = A/B;
154     }
155
156     return J;
157
158 }
159
160 double CAdimensional::CalcWe(double _Re, double _Ro, double _h,
    double _phi)
161 {
162
163     We = M_PI*_h*_phi*((_Re*_Re) - (_Ro*_Ro));
164
165     return We;
166

```

```

167 }
168
169 double CAdimensional::CalcWei(double _ct, double _We, double _Pi)
170 {
171
172     Wei = _ct*_We*_Pi;
173     return Wei;
174 }

```

Apresenta-se na listagem 6.24 o arquivo com código da classe CCarterTracy.

Listing 6.24: Arquivo de cabeçalho da classe CCarterTracy.

```

1 #ifndef CCARTERTRACY_H_
2 #define CCARTERTRACY_H_
3
4 #include <vector>
5
6
7 class CCarterTracy {
8
9     protected:
10
11         std::vector <double> Wecarter;
12         double AJ;
13
14     public:
15
16         CCarterTracy(){};
17
18         double CalcAJ(double _U, double _delp, double _we
19             , double _pdbarra, double _TD, double _td,
20             double _PD);
21         std::vector <double> CarterTracy(std::vector <
22             double> _TD, double _U, std::vector <double>
23             _delp, std::vector <double> _pdbarra, std:::
24             vector <double> _PD);
25
26         ~CCarterTracy(){};
27
28 };
29 #endif

```

Apresenta-se na listagem 6.25 o arquivo de implementação da classe CCarterTracy.

Listing 6.25: Arquivo de implementação da classe CCarterTracy.

---

```

1 #include "CCarterTracy.h"
2
3 using namespace std;
4
5 double CCarterTracy::CalcAJ(double _U, double _delp, double _we,
6     double _pdbarra, double _TD, double _td, double _PD)
7 {
8     double A,B,C;
9
10    A = (_U*_delp) - (_we*_pdbarra);
11    B = _PD - (_td*_pdbarra);
12    C = _TD-_td;
13    AJ = (A/B)*C;
14
15    return AJ;
16 }
17
18 vector <double> CCarterTracy::CarterTracy(vector <double> _TD,
19     double _U, vector <double> _delp, vector <double> _pdbarra,
20     vector <double> _PD)
21 {
22     double we;
23     for (int i=0; i < _TD.size(); i++)
24     {
25         if (i==0)
26             we = 0.0;
27         else
28             we = Wecarter[i-1] + CalcAJ(_U, _delp[i],
29                 Wecarter[i-1], _pdbarra[i], _TD[i],
30                 _TD[i-1], _PD[i]);
31
32     Wecarter.push_back(we);
33
34 }
35
36 return Wecarter;
37
38 }

```

---

Apresenta-se na listagem 6.26 o arquivo com código da classe CFetkovich.

Listing 6.26: Arquivo de cabeçalho da classe CFetkovich.

---

```

1 #ifndef CFETKOVICH_H_
2 #define CFETKOVICH_H_
3
4 #include "CAdimensional.h"
5
6 #include <vector>
7
8 class CFetkovich : CAdimensional{
9
10     protected:
11
12         std::vector <double>   WeFet;
13         double Pan, deltaWen;
14
15     public:
16
17         CFetkovich(){};
18
19         double CalcPan(double _Pi, double _We, double
20             _Wei);
21         double CalcdeltaWen(double _Wei, double _Pi,
22             double _Pan, double _Pbarra, double _J, double
23             _DeltaT);
24         std::vector <double> CalcFetkovic(double _We,
25             double _Wei, std::vector <double> _Pbarra,
26             double _J, std::vector <double> _DeltaT, std::
27             vector <double> _P);
28
29         ~CFetkovich(){};
30
31 };
32
33 #endif

```

---

Apresenta-se na listagem 6.27 o arquivo de implementação da classe CFetkovich.

Listing 6.27: Arquivo de implementação da classe CFetkovich.

---

```

1 #include "CFetkovich.h"
2
3 #include <cmath>
4
5 double CFetkovich::CalcPan(double _Pi, double _We, double _Wei)

```

---

```

6{
7
8    Pan = _Pi*(1.0 - (_We/_Wei));
9
10    return Pan;
11
12}
13
14double CFetkovich::CalcdeltaWen(double _Wei, double _Pi, double
    _Pan, double _Pbarra, double _J, double _DeltaT)
15{
16    double A, B, C;
17
18    A = (_Wei/_Pi)*(_Pan - _Pbarra);
19    B = -1*((_J*_Pi)/_Wei)*_DeltaT;
20    C = exp(B);
21    deltaWen = A*(1 - C);
22
23    return deltaWen;
24}
25
26std::vector <double> CFetkovich::CalcFetkovic(double _We, double
    _Wei, std::vector <double> _Pbarra, double _J, std::vector <
    double> _DeltaT, std::vector <double> _P)
27{
28    double prsI = _P[0];
29    double _paN, DelWen, wef;
30
31    for (int i=0; i< _P.size();i++)
32    {
33        if (i==0)
34        {
35            _paN = prsI;
36            wef = 0.0;
37        }
38        else
39        {
40
41            DelWen = CalcdeltaWen( _Wei, prsI, _paN,
                _Pbarra[i], _J, _DeltaT[i]);
42            wef = WeFet[i-1] + DelWen;
43            _paN = CalcPan(prsI, wef, _Wei);

```

---

```

44         }
45
46         WeFet.push_back(wef);
47     }
48
49     return WeFet;
50
51 }

```

---

Apresenta-se na listagem 6.28 o arquivo com código da classe CFluido.

Listing 6.28: Arquivo de cabeçalho da classe CFluido.

---

```

1 #ifndef CFLUIDO_H_
2 #define CFLUIDO_H_
3
4 class CFluido{
5
6     protected:
7
8         double mi;
9
10    public:
11
12        CFluido(){};
13
14        void SetMi(double _mi);
15        double GetMi();
16
17        ~CFluido(){};
18
19 };
20
21 #endif

```

---

Apresenta-se na listagem 6.29 o arquivo de implementação da classe CFluido.

Listing 6.29: Arquivo de implementação da classe CFluido.

---

```

1 #include "CFluido.h"
2
3 void CFluido::SetMi(double _mi)
4 {
5     mi=_mi;
6 }
7

```



```

8 double CFluido::GetMi()
9 {
10     return mi;
11 }

```

Apresenta-se na listagem 6.30 o arquivo com código da classe CGnuplot.

Listing 6.30: Arquivo de cabeçalho da classe CGnuplot.

```

1 //
  ///////////////////////////////////////////////////////////////////

2 //          Classe de Interface em C++ para o programa
  gnuplot.          //
3 //
  ///////////////////////////////////////////////////////////////////

4 // Esta interface usa pipes e nao ira funcionar em sistemas que
  nao suportam
5 // o padrao POSIX pipe.
6 // O mesmo foi testado em sistemas Windows (MinGW e Visual C++) e
  Linux(GCC/G++)
7 // Este programa foi originalmente escrito por:
8 // Historico de versoes:
9 // 0. Interface para linguagem C
10 //   por N. Devillard (27/01/03)
11 // 1. Interface para C++: tradução direta da versao em C
12 //   por Rajarshi Guha (07/03/03)
13 // 2. Correcoes para compatibilidadde com Win32
14 //   por V. Chyzhdenka (20/05/03)
15 // 3. Novos métodos membros, correcoes para compatibilidade com
  Win32 e Linux
16 //   por M. Burgis (10/03/08)
17 // 4. Traducaao para Portugues, documentacao - javadoc/doxygen,
18 //   e modificacoes na interface (adicao de interface
  alternativa)
19 //   por Bueno.A.D. (30/07/08)
20 //   Tarefas:
21 //   (v1)
22 //   Documentar toda classe
23 //   Adicionar novos métodos, criando atributos adicionais se
  necessario.
24 //   Adotar padrao C++, isto e, usar sobrecarga nas chamadas.
25 //   (v2)

```

```

26// Criar classe herdeira CGnuplot, que inclui somente a nova
    interface.
27// como e herdeira, o usuario vai poder usar nome antigos.
28// Vantagem: preserva classe original, cria nova interface,
    fica a critério do usuário
29// qual interface utilizar.
30//
    //////////////////////////////////////
31// Requisitos:
32// - O programa gnuplot deve estar instalado (veja http://www.gnuplot.info/download.html)
33// - No Windows: setar a Path do Gnuplot (i.e. C:/program files/
    gnuplot/bin)
34// ou setar a path usando: Gnuplot::set_GNUPlotPath(
    const std::string &path);
35// Gnuplot::set_GNUPlotPath("C:/program files/gnuplot
    /bin");
36// - Para um melhor uso, consulte o manual do gnuplot,
37// no GNU/Linux digite: man gnuplot ou info gnuplot.
38//
39// - Veja aula em http://www.lenep.uenf.br/~bueno/DisciplinaSL/
40//
41//
    //////////////////////////////////////
42
43
44#ifdef CGnuplot_h
45#define CGnuplot_h
46#include <iostream> // Para teste
47#include <string>
48#include <vector>
49#include <stdexcept> // Heranca da classe std::
    runtime_error em GnuplotException
50#include <cstdio> // Para acesso a arquivos FILE
51
52/**
53@brief Erros em tempo de execucao
54@class GnuplotException
55@file GnuplotException.h
56*/

```

```

57 class GnuplotException : public std::runtime_error
58 {
59 public:
60     /// Construtor
61     GnuplotException (const std::string & msg):std::runtime_error (
        msg) {}
62 };
63
64 /**
65 @brief Classe de interface para acesso ao programa gnuplot.
66 @class Gnuplot
67 @file  gnuplot_i.hpp
68 */
69 class Gnuplot
70 {
71 private:
72     //
    -----
    Atributos
73     FILE *   gnuclmd;          ///< Ponteiro para stream que escreve no
        pipe.
74     bool     valid;           ///< Flag que indica se a sessao do
        gnuplot esta valida.
75     bool     two_dim;         ///< true = verdadeiro = 2d, false =
        falso = 3d.
76     int      nplots;          ///< Numero de graficos (plots) na sessao
        .
77     std::string pstyle;       ///< Estilo utilizado para visualizacao
        das funcoes e dados.
78     std::string smooth;       ///< interpolate and approximate data in
        defined styles (e.g. spline).
79     std::vector<std::string> tmpfile_list; ///< Lista com nome dos
        arquivos temporarios.
80
81     //
    -----
    flags
82     bool fgrid;               ///< 0 sem grid,          1 com grid
83     bool fhidden3d;           ///< 0 nao oculta,        1 oculta
84     bool fcontour;            ///< 0 sem contorno,    1 com contorno
85     bool fsurface;           ///< 0 sem superficie,  1 com superficie
86     bool flegend;            ///< 0 sem legendad,    1 com legenda

```

```

87  bool ftitle;           ///< 0 sem titulo,      1 com titulo
88  bool fxlogscale;       ///< 0 desativa escala log, 1 ativa
                             escala log
89  bool fylogscale;       ///< 0 desativa escala log, 1 ativa
                             escala log
90  bool fzlogscale;       ///< 0 desativa escala log, 1 ativa
                             escala log
91  bool fsmooth;          ///< 0 desativa,          1 ativa
92
93  //
-----

94  // Atributos estaticos (compartilhados por todos os objetos)
95  static int tmpfile_num;           ///< Numero total de
                             arquivos temporarios (numero restrito).
96  static std::string m_sGNUPlotFileName; ///< Nome do arquivo
                             executavel do gnuplot.
97  static std::string m_sGNUPlotPath;   ///< Caminho para
                             executavel do gnuplot.
98  static std::string terminal_std;      ///< Terminal padrao (
                             standart), usado para visualizacoes.
99
100 //
-----

    Metodos
101 // Funcoes membro (métodos membro) (funcoes auxiliares)
102 /// @brief Cria arquivo temporario e retorna seu nome.
103 /// Usa get_program_path(); e popen();
104 void  init ();
105
106 /// @brief Cria arquivo temporario e retorna seu nome.
107 /// Usa get_program_path(); e popen();
108 void Init() { init(); }
109
110 /// @brief Cria arquivo temporario.
111 std::string create_tmpfile (std::ofstream & tmp);
112
113 /// @brief Cria arquivo temporario.
114 std::string CreateTmpFile (std::ofstream & tmp) { return
                             create_tmpfile(tmp); }
115
116 //

```

```

-----
117 // Funcoes estaticas (static functions)
118 /// @brief Retorna verdadeiro se a path esta presente.
119 static bool get_program_path ();
120
121 /// @brief Retorna verdadeiro se a path esta presente.
122 static bool Path() { return get_program_path(); }
123
124 /// @brief Checa se o arquivo existe.
125 static bool file_exists (const std::string & filename, int
    mode = 0);
126
127 /// @brief Checa se o arquivo existe.
128 static bool FileExists (const std::string & filename, int
    mode = 0)
129
    { return file_exists( filename, mode
    ); }
130
131 //
-----

132 public:
133 // Opcional: Seta path do gnuplot manualmente
134 // No windows: a path (caminho) deve ser dada usando '/' e nao
    backslash '\'
135 /// @brief Seta caminho para path do gnuplot.
136 //ex: CGnuplot::set_GNUPlotPath ("\"C:/program files/gnuplot/
    bin/\"");
137
138 static bool set_GNUPlotPath (const std::string & path);
139
140 /// @brief Seta caminho para path do gnuplot.
141 static bool Path(const std::string & path) { return
    set_GNUPlotPath(path); }
142 //
143 /// @brief Opcional: Seta terminal padrao (standart), usado
    para visualizacao dos graficos.
144 /// Valores padroes (default): Windows - win, Linux - x11, Mac
    - aqua
145 static void set_terminal_std (const std::string & type);
146

```

```

147  /// @brief Opcional: Seta terminal padrao (standart), usado
      para visualizacao dos graficos.
148  /// Para retornar para terminal janela precisa chamar
      ShowOnScreen().
149  /// Valores padroes (default): Windows - win, Linux - x11 ou
      wxt (fedora9), Mac - aqua
150  static void Terminal (const std::string & type) {
      set_terminal_std(type); }
151
152  //
      -----
      Construtores
153  /// @brief Construtor, seta o estilo do grafico na construcao.
154  Gnuplot (const std::string & style = "points");
155
156  /// @brief Construtor, plota um grafico a partir de um vector,
      diretamente na construcao.
157  Gnuplot (const std::vector < double >&x,
158           const std::string & title = "",
159           const std::string & style = "points",
160           const std::string & labelx = "x",
161           const std::string & labely = "y");
162
163  /// @brief Construtor, plota um grafico do tipo x_y a partir de
      vetores, diretamente na construcao.
164  Gnuplot (const std::vector < double >&x,
165           const std::vector < double >&y,
166           const std::string & title = "",
167           const std::string & style = "points",
168           const std::string & labelx = "x",
169           const std::string & labely = "y");
170
171  /// @brief Construtor, plota um grafico de x_y_z a partir de
      vetores, diretamente na construcao.
172  Gnuplot (const std::vector < double >&x,
173           const std::vector < double >&y,
174           const std::vector < double >&z,
175           const std::string & title = "",
176           const std::string & style = "points",
177           const std::string & labelx = "x",
178           const std::string & labely = "y",
179           const std::string & labelz = "z");

```

```

180
181  /// @brief Destrutor, necessario para deletar arquivos
      temporarios.
182  ~Gnuplot ();
183
184  //
      -----

185  /// @brief Envia comando para o gnuplot.
186  Gnuplot &  cmd (const std::string & cmdstr);
187
188  /// @brief Envia comando para o gnuplot.
189  Gnuplot &  Cmd (const std::string & cmdstr)      { return cmd(
      cmdstr); }

190
191  /// @brief Envia comando para o gnuplot.
192  Gnuplot &  Command (const std::string & cmdstr) { return cmd(
      cmdstr); }

193
194  /// @brief Sobrecarga operador <<, funciona como Comando.
195  Gnuplot &  operator<< (const std::string & cmdstr);
196
197  //
      -----

198  /// @brief Mostrar na tela ou escrever no arquivo, seta o tipo
      de terminal para terminal_std.
199  Gnuplot &  showonscreen ();                // Janela de saida e
      setada como default (win/x11/aqua)

200
201  /// @brief Mostrar na tela ou escrever no arquivo, seta o tipo
      de terminal para terminal_std.
202  Gnuplot &  ShowOnScreen ()                  { return
      showonscreen(); };

203
204  /// @brief Salva sessao do gnuplot para um arquivo postscript,
      informe o nome do arquivo sem extensao.
205  /// Depois retorna para modo terminal
206  Gnuplot &  savetops (const std::string & filename = "
      gnuplot_output");

207
208  /// @brief Salva sessao do gnuplot para um arquivo postscript,

```

```

        informe o nome do arquivo sem extensao
209  /// Depois retorna para modo terminal
210  Gnuplot & SaveTops (const std::string & filename = "
        gnuplot_output")
211
                                { return
                                    savetops(
                                        filename); }
212
213  /// @brief Salva sessao do gnuplot para um arquivo png, nome do
        arquivo sem extensao
214  /// Depois retorna para modo terminal
215  Gnuplot & savetopng (const std::string & filename = "
        gnuplot_output");
216
217  /// @brief Salva sessao do gnuplot para um arquivo png, nome do
        arquivo sem extensao
218  /// Depois retorna para modo terminal
219  Gnuplot & SaveTopng (const std::string & filename = "
        gnuplot_output")
220
                                { return
                                    savetopng(
                                        filename); }
221
222  /// @brief Salva sessao do gnuplot para um arquivo jpg, nome do
        arquivo sem extensao
223  /// Depois retorna para modo terminal
224  Gnuplot & savetojpeg (const std::string & filename = "
        gnuplot_output");
225
226  /// @brief Salva sessao do gnuplot para um arquivo jpg, nome do
        arquivo sem extensao
227  /// Depois retorna para modo terminal
228  Gnuplot & SaveTojpeg (const std::string & filename = "
        gnuplot_output")
229
                                { return
                                    savetojpeg(
                                        filename); }
230
231  /// @brief Salva sessao do gnuplot para um arquivo filename,
        usando o terminal_type e algum flag adicional
232  /// Ex:
233  /// grafico.SaveTo("pressao_X_temperatura","png", "enhanced

```



```

        size 1280,960");
234  /// Para melhor uso dos flags adicionais consulte o manual do
        gnuplot (help term)
235  Gnuplot& SaveTo(const std::string &filename,const std::string &
        terminal_type, std::string flags="");
236
237  //
        -----
        set e unset
238  /// @brief Seta estilos de linhas (em alguns casos sao
        necessarias informacoes adicionais).
239  /// lines, points, linespoints, impulses, dots, steps, fsteps,
        histeps,
240  /// boxes, histograms, filledcurves
241  Gnuplot & set_style (const std::string & stylestr = "points");
242
243  /// @brief Seta estilos de linhas (em alguns casos sao
        necessarias informacoes adicionais).
244  /// lines, points, linespoints, impulses, dots, steps, fsteps,
        histeps,
245  /// boxes, histograms, filledcurves
246  Gnuplot & Style (const std::string & stylestr = "points")
247                                     { return
                                                set_style(
                                                stylestr); }
248
249  /// @brief Ativa suavizacao.
250  /// Argumentos para interpolacoes e aproximacoes.
251  /// csplines, bezier, acsplines (para dados com valor > 0),
        sbezier, unique,
252  /// frequency (funciona somente com plot_x, plot_xy, plotfile_x
        ,
253  /// plotfile_xy (se a suavizacao esta ativa, set_style nao tem
        efeito na plotagem dos graficos)
254  Gnuplot & set_smooth (const std::string & stylestr = "csplines
        ");
255
256  /// @brief Desativa suavizacao.
257  Gnuplot & unset_smooth ();          // A suavizacao nao e
        setada por padrao (default)
258
259  /// @brief Ativa suavizacao.

```

```

260  /// Argumentos para interpolacoes e aproximacoes.
261  /// csplines, bezier, acsplines (para dados com valor > 0),
      sbezier, unique,
262  /// frequency (funciona somente com plot_x, plot_xy, plotfile_x
      ,
263  /// plotfile_xy (se a suavizacao esta ativa, set_style nao tem
      efeito na plotagem dos graficos)
264  Gnuplot & Smooth(const std::string & stylestr = "csplines")
265                                     { return
                                          set_smooth(
                                          stylestr); }

266
267  Gnuplot & Smooth( int _fsmooth )
268                                     { if( fsmooth =
                                          _fsmooth )
269                                         return
                                          set_contour
                                          ();
270                                     else
271                                         return
                                          unset_contour
                                          ();
272                                     }
273  /// @brief Desativa suavizacao.
274  Gnuplot & UnsetSmooth() { return
      unset_smooth (); }
275
276  /// @brief Escala o tamanho do ponto usado na plotagem.
277  Gnuplot & set_pointsize (const double pointsize = 1.0);
278
279  /// @brief Escala o tamanho do ponto usado na plotagem.
280  Gnuplot & PointSize (const double pointsize = 1.0)
281                                     { return
                                          set_pointsize(
                                          pointsize); }
282
283  /// @brief Ativa o grid (padrao = desativado).
284  Gnuplot & set_grid ();
285
286  /// @brief Desativa o grid (padrao = desativado).
287  Gnuplot & unset_grid ();
288

```

```

289  /// @brief Ativa/Desativa o grid (padrao = desativado).
290  Gnuplot &  Grid(bool _fgrid = 1)
291
292                                     { if(fgrid =
293                                         _fgrid)
294                                         return
295                                         set_grid();
296                                     else
297                                         return
298                                         unset_grid
299                                         (); }
300
301  /// @brief Seta taxa de amostragem das funcoes, ou dos dados de
302  interpolacao.
303  Gnuplot &  set_samples (const int samples = 100);
304
305  /// @brief Seta taxa de amostragem das funcoes, ou dos dados de
306  interpolacao.
307  Gnuplot &  Samples(const int samples = 100)  { return
308      set_samples(samples); }
309
310  /// @brief Seta densidade de isolinhas para plotagem de funcoes
311  como superficies (para plotagen 3d).
312  Gnuplot &  set_isosamples (const int isolines = 10);
313
314  /// @brief Seta densidade de isolinhas para plotagem de funcoes
315  como superficies (para plotagen 3d).
316  Gnuplot &  IsoSamples (const int isolines = 10){ return
317      set_isosamples(isolines); }
318
319  /// @brief Ativa remocao de linhas ocultas na plotagem de
320  superficies (para plotagen 3d).
321  Gnuplot &  set_hidden3d ();
322
323  /// @brief Desativa remocao de linhas ocultas na plotagem de
324  superficies (para plotagen 3d).
325  Gnuplot &  unset_hidden3d ();          // hidden3d nao e setado
326  por padrao (default)
327
328  /// @brief Ativa/Desativa remocao de linhas ocultas na plotagem
329  de superficies (para plotagen 3d).
330  Gnuplot &  Hidden3d(bool _fhidden3d = 1)
331
332                                     { if(fhidden3d =

```

```

317         _fhidden3d)
318         return
319         set_hidden3d
320         ();
321
322     else
323     return
324     unset_hidden3d
325     ();
326
327 }
328
329 /// @brief Ativa desenho do contorno em superficies (para
330 plotagen 3d).
331 /// @param base, surface, both.
332 Gnuplot & set_contour (const std::string & position = "base");
333
334 /// @brief Desativa desenho do contorno em superficies (para
335 plotagen 3d).
336 Gnuplot & unset_contour ();          // contour nao e setado
337                                     por default
338
339 /// @brief Ativa/Desativa desenho do contorno em superficies (
340 para plotagen 3d).
341 /// @param base, surface, both.
342 Gnuplot & Contour(const std::string & position = "base")
343 { return
344     set_contour(
345     position); }
346
347 Gnuplot & Contour( int _fcontour )
348 { if( fcontour =
349     _fcontour )
350     return
351     set_contour
352     ();
353     else
354     return
355     unset_contour
356     ();
357 }
358
359 /// @brief Ativa a visualizacao da superficie (para plotagen 3
360 d).
361 Gnuplot & set_surface ();          // surface e setado por

```

```

        padrao (default)
342
343 /// @brief Desativa a visualizacao da superficie (para
        plotagen 3d).
344 Gnuplot & unset_surface ();
345
346 /// @brief Ativa/Desativa a visualizacao da superficie (para
        plotagen 3d).
347 Gnuplot & Surface( int _fsurface = 1 )
348                                     { if(fsurface =
                                                _fsurface)
349                                         return
                                                set_surface
                                                ();
350                                         else
351                                         return
                                                unset_surface
                                                ();
352                                     }
353 /// @brief Ativa a legenda (a legenda é setada por padrao).
354 /// Posicao: inside/outside, left/center/right, top/center/
        bottom, nobox/box
355 Gnuplot & set_legend (const std::string & position = "default"
        );
356
357 /// @brief Desativa a legenda (a legenda é setada por padrao).
358 Gnuplot & unset_legend ();
359
360 /// @brief Ativa/Desativa a legenda (a legenda é setada por
        padrao).
361 Gnuplot & Legend(const std::string & position = "default")
362                                     { return
                                                set_legend(
                                                position); }
363
364 /// @brief Ativa/Desativa a legenda (a legenda é setada por
        padrao).
365 Gnuplot & Legend(int _flegend)
366                                     { if(flegend =
                                                _flegend)
367                                         return
                                                set_legend

```

```

368                                     ();
369                                     else
370                                     return
371                                     unset_legend
372                                     ();
373                                     }
374
375     /// @brief Ativa o titulo da secao do gnuplot.
376     Gnuplot & set_title (const std::string & title = "");
377
378     /// @brief Desativa o titulo da secao do gnuplot.
379     Gnuplot & unset_title ();          // 0 title nao e setado
380                                     por padrao (default)
381
382     /// @brief Ativa/Desativa o titulo da secao do gnuplot.
383     Gnuplot & Title(const std::string & title = "")
384     {
385         return set_title
386             (title);
387     }
388
389     Gnuplot & Title(int _ftitle)
390     {
391         if(ftitle =
392             _ftitle)
393             return
394                 set_title();
395         else
396             return
397                 unset_title
398                 ();
399     }
400
401     /// @brief Seta o rotulo (nome) do eixo y.
402     Gnuplot & set_ylabel (const std::string & label = "y");
403
404     /// @brief Seta o rotulo (nome) do eixo y.
405     /// Ex: set ylabel "{/Symbol s}[MPa]" font "Times Italic, 10"
406     Gnuplot & YLabel(const std::string & label = "y")
407     { return
408         set_ylabel(
409             label); }
410
411

```

```
399  /// @brief Seta o rotulo (nome) do eixo x.
400  Gnuplot & set_xlabel (const std::string & label = "x");
401
402  /// @brief Seta o rotulo (nome) do eixo x.
403  Gnuplot & XLabel(const std::string & label = "x")
404                      { return
                        set_xlabel(
                          label); }
405
406  /// @brief Seta o rotulo (nome) do eixo z.
407  Gnuplot & set_zlabel (const std::string & label = "z");
408
409  /// @brief Seta o rotulo (nome) do eixo z.
410  Gnuplot & ZLabel(const std::string & label = "z")
411                      { return
                        set_zlabel(
                          label); }
412
413  /// @brief Seta intervalo do eixo x.
414  Gnuplot & set_xrange (const int iFrom, const int iTo);
415
416  /// @brief Seta intervalo do eixo x.
417  Gnuplot & XRange (const int iFrom, const int iTo)
418                      { return
                        set_xrange(
                          iFrom,iTo); }
419
420  /// @brief Seta intervalo do eixo y.
421  Gnuplot & set_yrange (const int iFrom, const int iTo);
422
423  /// @brief Seta intervalo do eixo y.
424  Gnuplot & YRange (const int iFrom, const int iTo)
425                      { return
                        set_yrange(
                          iFrom,iTo); }
426
427  /// @brief Seta intervalo do eixo z.
428  Gnuplot & set_zrange (const int iFrom, const int iTo);
429
430  /// @brief Seta intervalo do eixo z.
431  Gnuplot & ZRange (const int iFrom, const int iTo)
432                      { return
```

```

set_zrange(
    iFrom,iTo); }

433
434 /// @brief Seta escalonamento automatico do eixo x (default).
435 Gnuplot & set_xautoscale ();
436
437 /// @brief Seta escalonamento automatico do eixo x (default).
438 Gnuplot & XAutoscale() { return
    set_xautoscale (); }
439
440 /// @brief Seta escalonamento automatico do eixo y (default).
441 Gnuplot & set_yautoscale ();
442
443 /// @brief Seta escalonamento automatico do eixo y (default).
444 Gnuplot & YAutoscale() { return
    set_yautoscale (); }
445
446 /// @brief Seta escalonamento automatico do eixo z (default).
447 Gnuplot & set_zautoscale ();
448
449 /// @brief Seta escalonamento automatico do eixo z (default).
450 Gnuplot & ZAutoscale() { return
    set_zautoscale (); }
451
452 /// @brief Ativa escala logaritma do eixo x (logscale nao e
    setado por default).
453 Gnuplot & set_xlogscale (const double base = 10);
454
455 /// @brief Desativa escala logaritma do eixo x (logscale nao e
    setado por default).
456 Gnuplot & unset_xlogscale ();
457
458 /// @brief Ativa escala logaritma do eixo x (logscale nao e
    setado por default).
459 Gnuplot & XLogscale (const double base = 10) { //if(base)
460
    return
        set_xlogscale
            (base);
461
    //else
462
    //return
        unset_xlogscale
            ();

```



```
463                                     }
464
465 /// @brief Ativa/Desativa escala logaritma do eixo x (logscale
      nao e setado por default).
466 Gnuplot & XLogscale(bool _fxlogscale)
467                                     { if(fxlogscale =
      _fxlogscale)
468                                     return
      set_xlogscale
      ();
469                                     else
470                                     return
      unset_xlogscale
      ();
471                                     }
472
473 /// @brief Ativa escala logaritma do eixo y (logscale nao e
      setado por default).
474 Gnuplot & set_ylogscale (const double base = 10);
475
476 /// @brief Ativa escala logaritma do eixo y (logscale nao e
      setado por default).
477 Gnuplot & YLogscale (const double base = 10) { return
      set_ylogscale (base); }
478
479 /// @brief Desativa escala logaritma do eixo y (logscale nao e
      setado por default).
480 Gnuplot & unset_ylogscale ();
481
482 /// @brief Ativa/Desativa escala logaritma do eixo y (logscale
      nao e setado por default).
483 Gnuplot & YLogscale(bool _fylogscale)
484                                     { if(fylogscale =
      _fylogscale)
485                                     return
      set_ylogscale
      ();
486                                     else
487                                     return
      unset_ylogscale
      ();
488                                     }
```

```

489
490 /// @brief Ativa escala logaritma do eixo y (logscale nao e
      setado por default).
491 Gnuplot & set_zlogscale (const double base = 10);
492
493 /// @brief Ativa escala logaritma do eixo y (logscale nao e
      setado por default).
494 Gnuplot & ZLogscale (const double base = 10) { return
      set_zlogscale (base); }
495
496 /// @brief Desativa escala logaritma do eixo z (logscale nao e
      setado por default).
497 Gnuplot & unset_zlogscale ();
498
499 /// @brief Ativa/Desativa escala logaritma do eixo y (logscale
      nao e setado por default).
500 Gnuplot & ZLogscale(bool _fzlogscale)
501                                     { if(fzlogscale =
                                         _fzlogscale)
502                                         return
                                         set_zlogscale
                                         ();
503                                         else
504                                         return
                                         unset_zlogscale
                                         ();
505                                     }
506
507
508 /// @brief Seta intervalo da palette (autoscale por padrao).
509 Gnuplot & set_cbrange (const int iFrom, const int iTo);
510
511 /// @brief Seta intervalo da palette (autoscale por padrao).
512 Gnuplot & CBRange(const int iFrom, const int iTo)
513                                     { return
                                         set_cbrange(
                                         iFrom, iTo); }
514
515 //
      -----
516 /// @brief Plota dados de um arquivo de disco.

```

```
517 Gnuplot & plotfile_x (const std::string & filename,
518                       const int column = 1, const std::string & title = "
                       ");
519
520 /// @brief Plota dados de um arquivo de disco.
521 Gnuplot & PlotFile (const std::string & filename,
522                   const int column = 1, const std::string & title = "
                       ")
523
524                                     { return
525                                     plotfile_x(
526                                     filename,
527                                     column, title);
528                                     }
529
530 /// @brief Plota dados de um vector.
531 Gnuplot & plot_x (const std::vector < double >&x, const std::
532                  string & title = "");
533
534 /// @brief Plota dados de um vector.
535 Gnuplot & PlotVector (const std::vector < double >&x, const
536                      std::string & title = "")
537
538                                     { return plot_x(
539                                     x, title ); }
540
541 /// @brief Plota pares x,y a partir de um arquivo de disco.
542 Gnuplot & plotfile_xy (const std::string & filename,
543                       const int column_x = 1,
544                       const int column_y = 2, const std::string & title
545                       = "");
546
547 /// @brief Plota pares x,y a partir de um arquivo de disco.
548 Gnuplot & PlotFile (const std::string & filename,
549                   const int column_x = 1,
550                   const int column_y = 2, const std::string & title
551                   = "")
552
553                                     {
554                                     return
555                                     plotfile_xy(
556                                     filename,
557                                     column_x,
558                                     column_y, title
559                                     );
560                                     }
```

```

543
544 /// @brief Plota pares x,y a partir de vetores.
545 Gnuplot & plot_xy (const std::vector < double >&x,
546                  const std::vector < double >&y, const std::string &
                    title = "");
547
548 /// @brief Plota pares x,y a partir de vetores.
549 Gnuplot & PlotVector (const std::vector < double >&x,
550                    const std::vector < double >&y, const std::string &
                    title = "")
551
552
553
554
555
556
557
558
559 /// @brief Plota pares x,y com barra de erro dy a partir de um
    arquivo.
560 Gnuplot & plotfile_xy_err (const std::string & filename,
561                          const int column_x = 1,
562                          const int column_y = 2,
563                          const int column_dy = 3, const std::string &
                    title = "");
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

571         const std::string & title = "");
572
573     /// @brief Plota pares x,y com barra de erro dy a partir de
        vetores.
574     Gnuplot & PlotVectorXYErrorBar(const std::vector < double >&x,
575         const std::vector < double >&y,
576         const std::vector < double >&dy,
577         const std::string & title = "")
578
579                                     { return
580                                         plot_xy_err
581                                             (x, y, dy,
582                                             title); }
579
580     /// @brief Plota valores de x,y,z a partir de um arquivo de
        disco.
581     Gnuplot & plotfile_xyz (const std::string & filename,
582         const int column_x = 1,
583         const int column_y = 2,
584         const int column_z = 3, const std::string & title
585             = "");
586
587     /// @brief Plota valores de x,y,z a partir de um arquivo de
        disco.
588     Gnuplot & PlotFile (const std::string & filename,
589         const int column_x = 1,
590         const int column_y = 2,
591         const int column_z = 3, const std::string & title
592             = "")
593
594                                     { return
595                                         plotfile_xyz
596                                             (filename,
597                                             column_x,
598                                             column_y,
599                                             column_z)
600                                         ; }
592
593     /// @brief Plota valores de x,y,z a partir de vetores.
594     Gnuplot & plot_xyz (const std::vector < double >&x,
595         const std::vector < double >&y,
596         const std::vector < double >&z, const std::string &
597             title = "");
598
599     /// @brief Plota valores de x,y,z a partir de vetores.

```

```

599 Gnuplot & PlotVector(const std::vector< double >&x,
600                      const std::vector< double >&y,
601                      const std::vector< double >&z, const std::string &
                        title = "")
602
                                                { return
                                                    plot_xyz(x,
                                                        y, z,
                                                        title); }
603
604 /// @brief Plota uma equacao da forma y = ax + b, voce fornece
        os coeficientes a e b.
605 Gnuplot & plot_slope (const double a, const double b, const
        std::string & title = "");
606
607 /// @brief Plota uma equacao da forma y = ax + b, voce fornece
        os coeficientes a e b.
608 Gnuplot & PlotSlope (const double a, const double b, const
        std::string & title = "")
609
                                                { return
                                                    plot_slope(
                                                        a,b,title);
                                                    }
610
611 /// @brief Plota uma equacao fornecida como uma std::string y=
        f(x).
612 /// Escrever somente a funcao f(x) e nao y=
613 /// A variavel independente deve ser x
614 /// Os operadores binarios aceitos sao:
615 /// ** exponenciacao,
616 /// * multiplicacao,
617 /// / divisao,
618 /// + adicao,
619 /// - subtracao,
620 /// % modulo
621 /// Os operadores unarios aceitos sao:
622 /// - menos,
623 /// ! fatorial
624 /// Funcoes elementares:
625 /// rand(x), abs(x), sgn(x), ceil(x), floor(x), int(x), imag(x)
        , real(x), arg(x),
626 /// sqrt(x), exp(x), log(x), log10(x), sin(x), cos(x), tan(x),
        asin(x), acos(x),

```

```

627  /// atan(x), atan2(y,x), sinh(x), cosh(x), tanh(x), asinh(x),
        acosh(x), atanh(x)
628  /// Funcoes especiais:
629  /// erf(x), erfc(x), inverf(x), gamma(x), igamma(a,x), lgamma(x)
        ), ibeta(p,q,x),
630  /// besj0(x), besj1(x), besy0(x), besy1(x), lambertw(x)
631  /// Funcoes estatisticas:
632  /// norm(x), invnorm(x)
633  Gnuplot & plot_equation (const std::string & equation,
634                          const std::string & title = "");
635
636  /// @brief Plota uma equacao fornecida como uma std::string y=
        f(x).
637  /// Escrever somente a funcao f(x) e nao y=
638  /// A variavel independente deve ser x.
639  /// Exemplo: gnuplot->PlotEquation(CFuncao& obj);
640  /// Deve receber um CFuncao, que tem cast para string.
641  Gnuplot & PlotEquation(const std::string & equation,
642                        const std::string & title = "")
643  { return
        plot_equation(
            equation, title )
        ; }
644
645  /// @brief Plota uma equacao fornecida na forma de uma std::
        string z=f(x,y).
646  /// Escrever somente a funcao f(x,y) e nao z=, as variaveis
        independentes sao x e y.
647  Gnuplot & plot_equation3d (const std::string & equation, const
        std::string & title = "");
648
649  /// @brief Plota uma equacao fornecida na forma de uma std::
        string z=f(x,y).
650  /// Escrever somente a funcao f(x,y) e nao z=, as vaiaveis
        independentes sao x e y.
651  // gnuplot->PlotEquation3d(CPolinomio());
652  Gnuplot & PlotEquation3d (const std::string & equation,
653                          const std::string & title = "")
654  { return
        plot_equation3d(
            equation, title )
        ; }

```

```
655
656 /// @brief Plota uma imagem.
657 Gnuplot & plot_image (const unsigned char *ucPicBuf,
658                       const int iWidth, const int iHeight, const std::
659                           string & title = "");
660
661 /// @brief Plota uma imagem.
662 Gnuplot & PlotImage (const unsigned char *ucPicBuf,
663                     const int iWidth, const int iHeight,
664                     const std::string & title = "")
665 { return plot_image
666     (ucPicBuf,
667      iWidth, iHeight,
668      title); }
669
670
671 -----
672
673 // Repete o ultimo comando de plotagem, seja plot (2D) ou splot
674 // (3D)
675 // Usado para visualizar plotagens, após mudar algumas opcoes
676 // de plotagem
677 // ou quando gerando o mesmo grafico para diferentes
678 // dispositivos (showonscreen, savetops)
679 Gnuplot & replot ();
680
681
682 // Repete o ultimo comando de plotagem, seja plot (2D) ou splot
683 // (3D)
684 // Usado para visualizar plotagens, após mudar algumas opcoes
685 // de plotagem
686 // ou quando gerando o mesmo grafico para diferentes
687 // dispositivos (showonscreen, savetops)
688 Gnuplot & Replot() { return replot()
689 ; }
690
691
692 // Reseta uma sessao do gnuplot (próxima plotagem apaga
693 // definicoes previas)
694 Gnuplot & reset_plot ();
695
696
697 // Reseta uma sessao do gnuplot (próxima plotagem apaga
698 // definicoes previas)
699 Gnuplot & ResetPlot() { return
```



```

        reset_plot(); }
681
682 // Chama função reset do gnuplot
683 Gnuplot & Reset()                      { this->cmd("reset");
        return *this; }
684
685 // Reseta uma sessao do gnuplot e seta todas as variaveis para
        o default
686 Gnuplot & reset_all ();
687
688 // Reseta uma sessao do gnuplot e seta todas as variaveis para
        o default
689 Gnuplot & ResetAll ()                    { return
        reset_all(); }
690
691 // Verifica se a sessao esta valida
692 bool is_valid ();
693
694 // Verifica se a sessao esta valida
695 bool IsValid ()                        { return is_valid
        (); };
696
697};
698typedef Gnuplot CGnuplot;
699#endif

```

Apresenta-se na listagem 6.31 o arquivo de implementação da classe CGnuplot.

Listing 6.31: Arquivo de implementação da classe CGnuplot.

```

1 //////////////////////////////////////
2 //
3 // A C++ interface to gnuplot.
4 //
5 // This is a direct translation from the C interface
6 // written by Nicolas Devillard (ndevilla@free.fr) which
7 // is available from http://ndevilla.free.fr/gnuplot/.
8 //
9 // As in the C interface this uses pipes and so wont
10 // run on a system that doesn't have POSIX pipe support
11 //
12 // Rajarshi Guha
13 // e-mail: rguha@indiana.edu, rajarshi@presidency.com
14 // http://cheminfo.informatics.indiana.edu/~rguha/code/cc++/

```

```

15 //
16 // 07/03/03
17 //
18 ///////////////////////////////////////////////////////////////////
19 //
20 // A little correction for Win32 compatibility
21 // and MS VC 6.0 done by V.Chyzhdenka
22 //
23 // Notes:
24 // 1. Added private method Gnuplot::init().
25 // 2. Temporary file is created in the current
26 //     folder but not in /tmp.
27 // 3. Added #ifdef WIN32 e.t.c. where is needed.
28 // 4. Added private member m_sGnuplotFileName is
29 //     a name of executed Gnuplot file.
30 //
31 // Viktor Chyzhdenka
32 // e-mail: chyzhdenka@mail.ru
33 //
34 // 20/05/03
35 //
36 ///////////////////////////////////////////////////////////////////
37 //
38 // corrections for Win32 and Linux compatibility
39 //
40 // some member functions added:
41 //   set_GnuplotPath, set_terminal_std,
42 //   create_tmpfile, get_program_path, file_exists,
43 //   operator<<, replot, reset_all, savetops, showonscreen,
44 //   plotfile_*, plot_xy_err, plot_equation3d
45 // set, unset: pointsize, grid, *logscale, *autoscale,
46 // smooth, title, legend, samples, isosamples,
47 // hidden3d, cbrange, contour
48 //
49 // Markus Burgis
50 // e-mail: mail@burgis.info
51 //
52 // 10/03/08
53 //
54 ///////////////////////////////////////////////////////////////////
55 //
56 // Modificacoes:

```

```

57// Tradução para o português
58// Adição de novos nomes para os métodos(funções)
59// Uso de documentação no formato javadoc/doxygen
60// Bueno A.D.
61// e-mail: bueno@lenep.uenf.br
62// 20/07/08
63//
64////////////////////////////////////
65#include <fstream>           // for std::ifstream
66#include <sstream>           // for std::ostringstream
67#include <list>              // for std::list
68#include <cstdio>            // for FILE, fputs(), fflush(),
    popen()
69#include <cstdlib>           // for getenv()
70#include "CGnuplot.h"
71
72// Se estamos no windows // defined for 32 and 64-bit
    environments
73#if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)
74 #include <io.h>             // for _access(), _mktemp()
75 #define GP_MAX_TMP_FILES 27 // 27 temporary files it's
    Microsoft restriction
76// Se estamos no unix, GNU/Linux, Mac Os X
77#elif defined(unix) || defined(__unix) || defined(__unix__) ||
    defined(__APPLE__) //all UNIX-like OSs (Linux, *BSD, MacOSX,
    Solaris, ...)
78 #include <unistd.h>         // for access(), mkstemp()
79 #define GP_MAX_TMP_FILES 64
80#else
81 #error unsupported or unknown operating system
82#endif
83
84//
    -----
85//
86// initialize static data
87//
88int Gnuplot::tmpfile_num = 0;
89
90// Se estamos no windows

```

```

91 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)
92 std::string Gnuplot::m_sGnUPlotFileName = "gnuplot.exe";
93 std::string Gnuplot::m_sGnUPlotPath = "C:/gnuplot/bin/";
94 // Se estamos no unix, GNU/Linux, Mac Os X
95 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
    defined(__APPLE__)
96 std::string Gnuplot::m_sGnUPlotFileName = "gnuplot";
97 std::string Gnuplot::m_sGnUPlotPath = "/usr/bin/";
98 #endif
99
100 // Se estamos no windows
101 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)
102 std::string Gnuplot::terminal_std = "windows";
103 // Se estamos no unix, GNU/Linux
104 #elif ( defined(unix) || defined(__unix) || defined(__unix__) )
    && !defined(__APPLE__)
105 std::string Gnuplot::terminal_std = "x11";
106 // Se estamos Mac Os X
107 #elif defined(__APPLE__)
108 std::string Gnuplot::terminal_std = "aqua";
109 #endif
110
111 //
    -----
112 //
113 // define static member function: set Gnuplot path manual
114 //   for windows: path with slash '/' not backslash '\'
115 //
116 bool Gnuplot::set_GNUPlotPath(const std::string &path)
117 {
118     std::string tmp = path + "/" + Gnuplot::m_sGnUPlotFileName;
119     #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
        defined(__TOS_WIN__)
120         if ( Gnuplot::file_exists(tmp,0) ) // check existence
121     #elif defined(unix) || defined(__unix) || defined(__unix__) ||
        defined(__APPLE__)
122         if ( Gnuplot::file_exists(tmp,1) ) // check existence and
            execution permission
123     #endif

```

```

124     {
125         Gnuplot::m_sGnUPlotPath = path;
126         return true;
127     }
128     else
129     {
130         Gnuplot::m_sGnUPlotPath.clear();
131         return false;
132     }
133 }
134
135 //
-----

136 // define static member function: set standart terminal, used by
    showonscreen
137 // defaults: Windows - win, Linux - x11, Mac - aqua
138 void Gnuplot::set_terminal_std(const std::string &type)
139 {
140     #if defined(unix) || defined(__unix) || defined(__unix__) ||
        defined(__APPLE__)
141         if (type.find("x11") != std::string::npos && getenv("DISPLAY"
            ) == NULL)
142         {
143             throw GnuplotException("Can't find DISPLAY variable");
144         }
145     #endif
146
147
148     Gnuplot::terminal_std = type;
149     return;
150 }
151
152
153 //
-----

154 // A string tokenizer taken from http://www.sunsite.ualberta.ca/
    Documentation/
155 // /Gnu/libstdc++-2.90.8/html/21_strings/stringtok_std_h.txt
156 template <typename Container>
157 void stringtok (Container &container,

```

```

158         std::string const &in,
159         const char * const delimiters = "\t\n")
160 {
161     const std::string::size_type len = in.length();
162     std::string::size_type i = 0;
163
164     while ( i < len )
165     {
166         // eat leading whitespace
167         i = in.find_first_not_of (delimiters, i);
168
169         if (i == std::string::npos)
170             return;    // nothing left but white space
171
172         // find the end of the token
173         std::string::size_type j = in.find_first_of (delimiters,
174             i);
175
176         // push token
177         if (j == std::string::npos)
178         {
179             container.push_back (in.substr(i));
180             return;
181         }
182         else
183             container.push_back (in.substr(i, j-i));
184
185         // set up for next loop
186         i = j + 1;
187     }
188
189     return;
190 }
191 //
192 //
193 // constructor: set a style during construction
194 //
195 Gnuplot::Gnuplot(const std::string &style)
196 {

```

```
197     this->init();
198     this->set_style(style);
199 }
200
201 //
-----

202 // constructor: open a new session, plot a signal (x)
203 Gnuplot::Gnuplot(const std::vector<double> &x,
204                 const std::string &title,
205                 const std::string &style,
206                 const std::string &labelx,
207                 const std::string &labely)
208 {
209     this->init();
210
211     this->set_style(style);
212     this->set_xlabel(labelx);
213     this->set_ylabel(labely);
214
215     this->plot_x(x,title);
216 }
217
218 //
-----

219 // constructor: open a new session, plot a signal (x,y)
220 Gnuplot::Gnuplot(const std::vector<double> &x,
221                 const std::vector<double> &y,
222                 const std::string &title,
223                 const std::string &style,
224                 const std::string &labelx,
225                 const std::string &labely)
226 {
227     this->init();
228
229     this->set_style(style);
230     this->set_xlabel(labelx);
231     this->set_ylabel(labely);
232
233     this->plot_xy(x,y,title);
234 }
```

```

235
236 //
-----

237 // constructor: open a new session, plot a signal (x,y,z)
238 Gnuplot::Gnuplot(const std::vector<double> &x,
239                 const std::vector<double> &y,
240                 const std::vector<double> &z,
241                 const std::string &title,
242                 const std::string &style,
243                 const std::string &labelx,
244                 const std::string &labely,
245                 const std::string &labelz)
246 {
247     this->init();
248
249     this->set_style(style);
250     this->set_xlabel(labelx);
251     this->set_ylabel(labely);
252     this->set_zlabel(labelz);
253
254     this->plot_xyz(x,y,z,title);
255 }
256
257 //
-----

258 // Destructor: needed to delete temporary files
259 Gnuplot::~Gnuplot()
260 {
261     if ((this->tmpfile_list).size() > 0)
262     {
263         for (unsigned int i = 0; i < this->tmpfile_list.size(); i
                ++))
264             remove( this->tmpfile_list[i].c_str() );
265
266         Gnuplot::tmpfile_num -= this->tmpfile_list.size();
267     }
268
269     // A stream opened by popen() should be closed by pclose()
270 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)

```



```

271     if (_pclose(this->gnucmd) == -1)
272 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
        defined(__APPLE__)
273     if (pclose(this->gnucmd) == -1)
274 #endif
275         true; //throw GnuplotException("Problem closing
                communication to gnuplot");
276 }
277
278 //
-----

279 // Resets a gnuplot session (next plot will erase previous ones)
280 Gnuplot& Gnuplot::reset_plot()
281 {
282     if (this->tmpfile_list.size() > 0)
283     {
284         for (unsigned int i = 0; i < this->tmpfile_list.size(); i
                ++))
285             remove(this->tmpfile_list[i].c_str());
286
287         Gnuplot::tmpfile_num -= this->tmpfile_list.size();
288         this->tmpfile_list.clear();
289     }
290
291     this->nplots = 0;
292
293     return *this;
294 }
295
296 //
-----

297 // resets a gnuplot session and sets all variables to default
298 Gnuplot& Gnuplot::reset_all()
299 {
300     if (this->tmpfile_list.size() > 0)
301     {
302         for (unsigned int i = 0; i < this->tmpfile_list.size(); i
                ++))
303             remove(this->tmpfile_list[i].c_str());
304

```

```

305         Gnuplot::tmpfile_num -= this->tmpfile_list.size();
306         this->tmpfile_list.clear();
307     }
308
309     this->nplots = 0;
310     this->cmd("reset");
311     this->cmd("clear");
312     this->pstyle = "points";
313     this->smooth = "";
314     this->showonscreen();
315
316     return *this;
317 }
318
319 //
-----
320 // Find out if valid is true
321 bool Gnuplot::is_valid()
322 {
323     return (this->valid);
324 }
325
326 //
-----
327 // replot repeats the last plot or splot command
328 Gnuplot& Gnuplot::replot()
329 {
330     if (this->nplots > 0)
331     {
332         this->cmd("replot");
333     }
334
335     return *this;
336 }
337
338
339 //
-----
340 // Change the plotting style of a gnuplot session

```

```

341 Gnuplot& Gnuplot::set_style(const std::string &stylestr)
342 {
343     if (stylestr.find("lines") == std::string::npos &&
344         stylestr.find("points") == std::string::npos &&
345         stylestr.find("linespoints") == std::string::npos &&
346         stylestr.find("impulses") == std::string::npos &&
347         stylestr.find("dots") == std::string::npos &&
348         stylestr.find("steps") == std::string::npos &&
349         stylestr.find("fsteps") == std::string::npos &&
350         stylestr.find("histeps") == std::string::npos &&
351         stylestr.find("boxes") == std::string::npos &&
352         // 1-4 columns of data are required
353         stylestr.find("filledcurves") == std::string::npos &&
354         stylestr.find("histograms") == std::string::npos )
355         //only for one data column
356         stylestr.find("labels") == std::string::npos
357         && // 3 columns of data are required
358         stylestr.find("xerrorbars") == std::string::npos
359         && // 3-4 columns of data are required
360         stylestr.find("xerrorlines") == std::string::npos
361         && // 3-4 columns of data are required
362         stylestr.find("errorbars") == std::string::npos
363         && // 3-4 columns of data are required
364         stylestr.find("errorlines") == std::string::npos
365         && // 3-4 columns of data are required
366         stylestr.find("yerrorbars") == std::string::npos
367         && // 3-4 columns of data are required
368         stylestr.find("yerrorlines") == std::string::npos
369         && // 3-4 columns of data are required
370         stylestr.find("boxerrorbars") == std::string::npos
371         && // 3-5 columns of data are required
372         stylestr.find("xyerrorbars") == std::string::npos
373         && // 4,6,7 columns of data are required
374         stylestr.find("xyerrorlines") == std::string::npos
375         && // 4,6,7 columns of data are required
376         stylestr.find("boxxyerrorbars") == std::string::npos
377         && // 4,6,7 columns of data are required
378         stylestr.find("financebars") == std::string::npos
379         && // 5 columns of data are required
380         stylestr.find("candlesticks") == std::string::npos
381         && // 5 columns of data are required
382         stylestr.find("vectors") == std::string::npos

```

```

    &&
368 //          stylestr.find("image")          == std::string::npos
    &&
369 //          stylestr.find("rgbimage")        == std::string::npos
    &&
370 //          stylestr.find("pm3d")            == std::string::npos    )
371 {
372     this->pstyle = std::string("points");
373 }
374 else
375 {
376     this->pstyle = stylestr;
377 }
378
379 return *this;
380 }
381
382 //
-----

383 // smooth: interpolation and approximation of data
384 Gnuplot& Gnuplot::set_smooth(const std::string &stylestr)
385 {
386     if (stylestr.find("unique")      == std::string::npos &&
387         stylestr.find("frequency")   == std::string::npos &&
388         stylestr.find("csplines")    == std::string::npos &&
389         stylestr.find("acsplines")   == std::string::npos &&
390         stylestr.find("bezier")      == std::string::npos &&
391         stylestr.find("sbezier")     == std::string::npos )
392     {
393         this->smooth = "";
394     }
395     else
396     {
397         this->smooth = stylestr;
398     }
399
400     return *this;
401 }
402
403 //
-----

```

```

404 // unset smooth
405 Gnuplot& Gnuplot::unset_smooth()
406 {
407     this->smooth = "";
408
409     return *this;
410 }
411
412 //
-----

413 // sets terminal type to windows / x11
414 Gnuplot& Gnuplot::showonscreen()
415 {
416     this->cmd("set output");
417     this->cmd("set terminal " + Gnuplot::terminal_std);
418
419     return *this;
420 }
421
422 //
-----

423 // saves a gnuplot session to a postscript file
424 Gnuplot& Gnuplot::savetops(const std::string &filename)
425 {
426 //     this->cmd("set terminal postscript color");           //
    Tipo de terminal (tipo de arquivo)
427 //
428 //     std::ostream cmdstr;                                //
    Muda o nome do arquivo
429 //     cmdstr << "set output \"" << filename << ".ps\"";    //
    Nome do arquivo
430 //     this->cmd(cmdstr.str());
431 //     this->replot();                                       //
    Replota o gráfico, agora salvando o arquivo ps
432 //
433 //     ShowOnScreen ();                                    //
    Volta para terminal modo janela
434
435     this->cmd("set term png size 1800,1200");

```

```

436
437     std::ostringstream cmdstr;
438     cmdstr << "set output \"./images/\" << filename << ".png\"";
439     this->cmd(cmdstr.str());
440     this->Replot();
441
442     return *this;
443 }
444 //
-----

445 // saves a gnuplot session to a png file and return do on screen
    terminal
446 Gnuplot& Gnuplot::savetopng(const std::string &filename)
447 {
448 //                                     //
    Muda o terminal
449 //     this->cmd("set term png enhanced size 1280,960"); //
    Tipo de terminal (tipo de arquivo)
450 //
451 //     std::ostringstream cmdstr; //
    Muda o nome do arquivo
452 //     cmdstr << "set output \"" << filename << ".png\""; //
    Nome do arquivo
453 //     this->cmd(cmdstr.str());
454 //     this->replot(); //
    Replota o gráfico, agora salvando o arquivo ps
455 //                                     //
    Retorna o terminal para o padrão janela
456 //     ShowOnScreen (); //
    Volta para terminal modo janela
457 //     this->replot(); //
    Replota o gráfico, agora na tela
458     SaveTo(filename, "png", "enhanced_size_1280,960");
459
460     return *this;
461 }
462
463 //
-----

464 // saves a gnuplot session to a jpeg file and return do on screen

```

```

terminal
465 Gnuplot& Gnuplot::savetojpeg(const std::string &filename)
466 {
467                                     // Muda o
                                     terminal

468 //      this->cmd("set term jpeg enhanced size 1280,960"); //
      Tipo de terminal (tipo de arquivo)
469 //
470 //      std::ostream cmdstr; //
      Muda o nome do arquivo
471 //      cmdstr << "set output \"" << filename << ".jpeg\""; //
      Nome do arquivo
472 //      this->cmd(cmdstr.str());
473 //      this->replot(); //
      Replota o gráfico, agora salvando o arquivo ps
474 // //
      Retorna o terminal para o padrão janela
475 //      ShowOnScreen (); //
      Volta para terminal modo janela
476 //      this->replot(); //
      Replota o gráfico, agora na tela
477      SaveTo(filename,"jpeg", "enhanced_size_1280,960");
478
479      return *this;
480 }
481
482
483 //
-----

484 // saves a gnuplot session to specific terminal and output file
485 // @filename: name of disc file
486 // @terminal_type: type of terminal
487 // @flags: additional information specific to terminal type
488 // Ex:
489 // grafico.SaveTo("pressao_X_temperatura","png", "enhanced size
      1280,960");
490 // grafico.TerminalType("png").SaveFile(pressao_X_temperatura);
      pense nisso?
491 Gnuplot& Gnuplot::SaveTo(const std::string &filename,const std::

```

```

    string &terminal_type, std::string flags)
492{                                                                    // Muda o
    terminal
493    this->cmd("set_term_" + terminal_type + "_" + flags);           //
    Tipo de terminal (tipo de arquivo) e flags adicionais
494    std::ostringstream cmdstr;                                       // Muda o
    nome do arquivo
495    cmdstr << "set_output_" << filename << "." << terminal_type
    << "_";
496    this->cmd(cmdstr.str());
497    this->replot();                                                    //
    Replota o gráfico, agora salvando o arquivo ps
498                                                                    //
                                                                    Retorna
                                                                    o
                                                                    terminal
                                                                    para o
                                                                    padrão
                                                                    janela
499    ShowOnScreen ();
500    this->replot();                                                    //
    Replota o gráfico, agora na tela
501
502    return *this;
503}
504
505//
-----

506// Switches legend on
507Gnuplot& Gnuplot::set_legend(const std::string &position)
508{
509    std::ostringstream cmdstr;
510    cmdstr << "set_key_" << position;
511
512    this->cmd(cmdstr.str());
513
514    return *this;
515}
516
517//
-----

```



```
518 // Switches legend off
519 Gnuplot& Gnuplot::unset_legend()
520 {
521     this->cmd("unset_key");
522
523     return *this;
524 }
525
526 //
-----

527 // Turns grid on
528 Gnuplot& Gnuplot::set_grid()
529 {
530     this->cmd("set_grid");
531
532     return *this;
533 }
534
535 //
-----

536 // Turns grid off
537 Gnuplot& Gnuplot::unset_grid()
538 {
539     this->cmd("unset_grid");
540
541     return *this;
542 }
543
544 //
-----

545 // turns on log scaling for the x axis
546 Gnuplot& Gnuplot::set_xlogscale(const double base)
547 {
548     std::ostringstream cmdstr;
549
550     cmdstr << "set_logscale_x" << base;
551     this->cmd(cmdstr.str());
552
```

```
553     return *this;
554 }
555
556 //
-----
557 // turns on log scaling for the y axis
558 Gnuplot& Gnuplot::set_ylogscale(const double base)
559 {
560     std::ostringstream cmdstr;
561
562     cmdstr << "set_ylogscale_y" << base;
563     this->cmd(cmdstr.str());
564
565     return *this;
566 }
567
568 //
-----
569 // turns on log scaling for the z axis
570 Gnuplot& Gnuplot::set_zlogscale(const double base)
571 {
572     std::ostringstream cmdstr;
573
574     cmdstr << "set_zlogscale_z" << base;
575     this->cmd(cmdstr.str());
576
577     return *this;
578 }
579
580 //
-----
581 // turns off log scaling for the x axis
582 Gnuplot& Gnuplot::unset_xlogscale()
583 {
584     this->cmd("unset_xlogscale_x");
585     return *this;
586 }
587
588 //
```

```
-----

589 // turns off log scaling for the y axis
590 Gnuplot& Gnuplot::unset_ylogscale()
591 {
592     this->cmd("unset_logscale_y");
593     return *this;
594 }
595
596 //
-----

597 // turns off log scaling for the z axis
598 Gnuplot& Gnuplot::unset_zlogscale()
599 {
600     this->cmd("unset_logscale_z");
601     return *this;
602 }
603
604
605 //
-----

606 // scales the size of the points used in plots
607 Gnuplot& Gnuplot::set_pointsize(const double pointsize)
608 {
609     std::ostringstream cmdstr;
610     cmdstr << "set_pointsize_" << pointsize;
611     this->cmd(cmdstr.str());
612
613     return *this;
614 }
615
616 //
-----

617 // set isoline density (grid) for plotting functions as surfaces
618 Gnuplot& Gnuplot::set_samples(const int samples)
619 {
620     std::ostringstream cmdstr;
621     cmdstr << "set_samples_" << samples;
622     this->cmd(cmdstr.str());
```

```
623
624     return *this;
625 }
626
627 //
-----

628 // set isoline density (grid) for plotting functions as surfaces
629 Gnuplot& Gnuplot::set_isosamples(const int isolines)
630 {
631     std::ostringstream cmdstr;
632     cmdstr << "set_isosamples_" << isolines;
633     this->cmd(cmdstr.str());
634
635     return *this;
636 }
637
638 //
-----

639 // enables hidden line removal for surface plotting
640 Gnuplot& Gnuplot::set_hidden3d()
641 {
642     this->cmd("set_hidden3d");
643
644     return *this;
645 }
646
647 //
-----

648 // disables hidden line removal for surface plotting
649 Gnuplot& Gnuplot::unset_hidden3d()
650 {
651     this->cmd("unset_hidden3d");
652
653     return *this;
654 }
655
656 //
```

```
657// enables contour drawing for surfaces set contour {base |
    surface | both}
658Gnuplot& Gnuplot::set_contour(const std::string &position)
659{
660    if (position.find("base")      == std::string::npos    &&
661        position.find("surface") == std::string::npos    &&
662        position.find("both")     == std::string::npos    )
663    {
664        this->cmd("set_␣contour_␣base");
665    }
666    else
667    {
668        this->cmd("set_␣contour_␣" + position);
669    }
670
671    return *this;
672}
673
674//
-----
675// disables contour drawing for surfaces
676Gnuplot& Gnuplot::unset_contour()
677{
678    this->cmd("unset_␣contour");
679
680    return *this;
681}
682
683//
-----
684// enables the display of surfaces (for 3d plot)
685Gnuplot& Gnuplot::set_surface()
686{
687    this->cmd("set_␣surface");
688
689    return *this;
690}
691
692//
-----
```

```
693 // disables the display of surfaces (for 3d plot)
694 Gnuplot& Gnuplot::unset_surface()
695 {
696     this->cmd("unset_surface");
697
698     return *this;
699 }
700
701 //
-----

702 // Sets the title of a gnuplot session
703 Gnuplot& Gnuplot::set_title(const std::string &title)
704 {
705     std::ostringstream cmdstr;
706
707     cmdstr << "set_title \" " << title << "\"";
708     this->cmd(cmdstr.str());
709
710     return *this;
711 }
712
713 //
-----

714 // Clears the title of a gnuplot session
715 Gnuplot& Gnuplot::unset_title()
716 {
717     this->set_title("");
718
719     return *this;
720 }
721
722 //
-----

723 // set labels
724 // set the xlabel
725 Gnuplot& Gnuplot::set_xlabel(const std::string &label)
726 {
727     std::ostringstream cmdstr;
```

```
728
729     cmdstr << "set_xlabel\" << label << "\"";
730     this->cmd(cmdstr.str());
731
732     return *this;
733 }
734
735 //
-----

736 // set the ylabel
737 Gnuplot& Gnuplot::set_ylabel(const std::string &label)
738 {
739     std::ostringstream cmdstr;
740
741     cmdstr << "set_ylabel\" << label << "\"";
742     this->cmd(cmdstr.str());
743
744     return *this;
745 }
746
747 //
-----

748 // set the zlabel
749 Gnuplot& Gnuplot::set_zlabel(const std::string &label)
750 {
751     std::ostringstream cmdstr;
752
753     cmdstr << "set_zlabel\" << label << "\"";
754     this->cmd(cmdstr.str());
755
756     return *this;
757 }
758
759 //
-----

760 // set range
761 // set the xrange
762 Gnuplot& Gnuplot::set_xrange(const int iFrom,
763                             const int iTo)
```

```
764{
765    std::ostringstream cmdstr;
766
767    cmdstr << "set_xrange[" << iFrom << ":" << iTo << "];";
768    this->cmd(cmdstr.str());
769
770    return *this;
771}
772
773//
-----
774// set autoscale x
775Gnuplot& Gnuplot::set_xautoscale()
776{
777    this->cmd("set_xrange_restore");
778    this->cmd("set_autoscale_x");
779
780    return *this;
781}
782
783//
-----
784// set the yrange
785Gnuplot& Gnuplot::set_yrange(const int iFrom, const int iTo)
786{
787    std::ostringstream cmdstr;
788
789    cmdstr << "set_yrange[" << iFrom << ":" << iTo << "];";
790    this->cmd(cmdstr.str());
791
792    return *this;
793}
794
795//
-----
796// set autoscale y
797Gnuplot& Gnuplot::set_yautoscale()
798{
799    this->cmd("set_yrange_restore");
```



```

800     this->cmd("set_autoscale_y");
801
802     return *this;
803 }
804
805 //
-----

806 // set the zrange
807 Gnuplot& Gnuplot::set_zrange(const int iFrom,
808                               const int iTo)
809 {
810     std::ostringstream cmdstr;
811
812     cmdstr << "set_zrange[" << iFrom << ":" << iTo << "];";
813     this->cmd(cmdstr.str());
814
815     return *this;
816 }
817
818 //
-----

819 // set autoscale z
820 Gnuplot& Gnuplot::set_zautoscale()
821 {
822     this->cmd("set_zrange_restore");
823     this->cmd("set_autoscale_z");
824
825     return *this;
826 }
827
828 //
-----

829 // set the palette range
830 Gnuplot& Gnuplot::set_cbrange(const int iFrom,
831                                const int iTo)
832 {
833     std::ostringstream cmdstr;
834
835     cmdstr << "set_cbrange[" << iFrom << ":" << iTo << "];";

```

```

836     this->cmd(cmdstr.str());
837
838     return *this;
839 }
840
841 //
-----

842 // Plots a linear equation y=ax+b (where you supply the
843 // slope a and intercept b)
844 Gnuplot& Gnuplot::plot_slope(const double a,
845                               const double b,
846                               const std::string &title)
847 {
848     std::ostringstream cmdstr;
849
850     // command to be sent to gnuplot
851     if (this->nplots > 0 && this->two_dim == true)
852         cmdstr << "replot_";
853     else
854         cmdstr << "plot_";
855
856     cmdstr << a << "_*_x+_ " << b << "_title_\n";
857
858     if (title == "")
859         cmdstr << "f(x)_=_ " << a << "_*_x+_ " << b;
860     else
861         cmdstr << title;
862
863     cmdstr << "\"_with_ " << this->pstyle;
864
865     // Do the actual plot
866     this->cmd(cmdstr.str());
867
868     return *this;
869 }
870
871 //
-----

872 // Plot an equation which is supplied as a std::string y=f(x) (
      only f(x) expected)

```

```

873 Gnuplot& Gnuplot::plot_equation(const std::string &equation,
874                                const std::string &title)
875 {
876     std::ostringstream cmdstr;
877
878     // command to be sent to gnuplot
879     if (this->nplots > 0 && this->two_dim == true)
880         cmdstr << "replot_";
881     else
882         cmdstr << "plot_";
883
884     cmdstr << equation << "_title_\"";
885
886     if (title == "")
887         cmdstr << "f(x)_=" << equation;
888     else
889         cmdstr << title;
890
891     cmdstr << "\"_with_" << this->pstyle;
892
893     // Do the actual plot
894     this->cmd(cmdstr.str());
895
896     return *this;
897 }
898
899 //
-----
900 // plot an equation supplied as a std::string y=(x)
901 Gnuplot& Gnuplot::plot_equation3d(const std::string &equation,
902                                  const std::string &title)
903 {
904     std::ostringstream cmdstr;
905
906     // command to be sent to gnuplot
907     if (this->nplots > 0 && this->two_dim == false)
908         cmdstr << "replot_";
909     else
910         cmdstr << "splot_";
911
912     cmdstr << equation << "_title_\"";

```

```

913
914     if (title == "")
915         cmdstr << "f(x,y)_" << equation;
916     else
917         cmdstr << title;
918
919     cmdstr << "\"_with_" << this->pstyle;
920
921     // Do the actual plot
922     this->cmd(cmdstr.str());
923
924     return *this;
925 }
926
927 //
-----
928 // Plots a 2d graph from a list of doubles (x) saved in a file
929 Gnuplot& Gnuplot::plotfile_x(const std::string &filename,
930                             const int column,
931                             const std::string &title)
932 {
933     // check if file exists
934     if( !(Gnuplot::file_exists(filename,4)) ) // check existence
        and read permission
935     {
936         std::ostringstream except;
937         if( !(Gnuplot::file_exists(filename,0)) ) // check
            existence
938             except << "File_" << filename << "\"_does_not_exist
                ";
939         else
940             except << "No_read_permission_for_File_" <<
                filename << "\"";
941         throw GnuplotException( except.str() );
942         return *this;
943     }
944
945     std::ostringstream cmdstr;
946
947     // command to be sent to gnuplot
948     if (this->nplots > 0 && this->two_dim == true)

```

```

949         cmdstr << "replot_";
950     else
951         cmdstr << "plot_";
952
953     cmdstr << "\" " << filename << "\"_using" << column;
954
955     if (title == "")
956         cmdstr << "_notitle_";
957     else
958         cmdstr << "_title_" << title << "\"_";
959
960     if(smooth == "")
961         cmdstr << "with_" << this->pstyle;
962     else
963         cmdstr << "smooth_" << this->smooth;
964
965     // Do the actual plot
966     this->cmd(cmdstr.str()); //nplots++; two_dim = true; already
        in this->cmd();
967
968     return *this;
969 }
970
971 //
-----
972 // Plots a 2d graph from a list of doubles: x
973 Gnuplot& Gnuplot::plot_x(const std::vector<double> &x,
974                          const std::string &title)
975 {
976     if (x.size() == 0)
977     {
978         throw GnuplotException("std::vector too small");
979         return *this;
980     }
981
982     std::ofstream tmp;
983     std::string name = create_tmpfile(tmp);
984     if (name == "")
985         return *this;
986
987     // write the data to file

```

```

988     for (unsigned int i = 0; i < x.size(); i++)
989         tmp << x[i] << std::endl;
990
991     tmp.flush();
992     tmp.close();
993
994     this->plotfile_x(name, 1, title);
995
996     return *this;
997 }
998
999 //
-----
1000 // Plots a 2d graph from a list of doubles (x y) saved in a file
1001 Gnuplot& Gnuplot::plotfile_xy(const std::string &filename,
1002                               const int column_x,
1003                               const int column_y,
1004                               const std::string &title)
1005 {
1006     // check if file exists
1007     if( !(Gnuplot::file_exists(filename,4)) ) // check existence
        and read permission
1008     {
1009         std::ostringstream except;
1010         if( !(Gnuplot::file_exists(filename,0)) ) // check
            existence
1011             except << "File_\n" << filename << "\n_does_not_exist
                ";
1012         else
1013             except << "No_read_permission_for_File_\n" <<
                filename << "\n";
1014         throw GnuplotException( except.str() );
1015         return *this;
1016     }
1017
1018     std::ostringstream cmdstr;
1019
1020     // command to be sent to gnuplot
1021     if (this->nplots > 0 && this->two_dim == true)
1022         cmdstr << "replot_";
1023     else

```

```

1024         cmdstr << "plot_";
1025
1026         cmdstr << "\" " << filename << "\"_using_" << column_x << ":"
            << column_y;
1027
1028         if (title == "")
1029             cmdstr << "_notitle_";
1030         else
1031             cmdstr << "_title_" << title << "\"_";
1032
1033         if(smooth == "")
1034             cmdstr << "with_" << this->pstyle;
1035         else
1036             cmdstr << "smooth_" << this->smooth;
1037
1038         // Do the actual plot
1039         this->cmd(cmdstr.str());
1040
1041         return *this;
1042     }
1043
1044 //
-----
1045 // Plots a 2d graph from a list of doubles: x y
1046 Gnuplot& Gnuplot::plot_xy(const std::vector<double> &x,
1047                           const std::vector<double> &y,
1048                           const std::string &title)
1049 {
1050     if (x.size() == 0 || y.size() == 0)
1051     {
1052         throw GnuplotException("std::vectors_too_small");
1053         return *this;
1054     }
1055
1056     if (x.size() != y.size())
1057     {
1058         throw GnuplotException("Length_of_the_std::vectors_
            differs");
1059         return *this;
1060     }
1061

```

```

1062     std::ofstream tmp;
1063     std::string name = create_tmpfile(tmp);
1064     if (name == "")
1065         return *this;
1066
1067     // write the data to file
1068     for (unsigned int i = 0; i < x.size(); i++)
1069         tmp << x[i] << " " << y[i] << std::endl;
1070
1071     tmp.flush();
1072     tmp.close();
1073
1074     this->plotfile_xy(name, 1, 2, title);
1075
1076     return *this;
1077 }
1078
1079 //
-----
1080 // Plots a 2d graph with errorbars from a list of doubles (x y dy
    ) saved in a file
1081 Gnuplot& Gnuplot::plotfile_xy_err(const std::string &filename,
1082                                   const int column_x,
1083                                   const int column_y,
1084                                   const int column_dy,
1085                                   const std::string &title)
1086 {
1087     // check if file exists
1088     if( !(Gnuplot::file_exists(filename,4)) ) // check existence
        and read permission
1089     {
1090         std::ostringstream except;
1091         if( !(Gnuplot::file_exists(filename,0)) ) // check
            existence
1092             except << "File_" << filename << "\"_does_not_exist
                ";
1093         else
1094             except << "No_read_permission_for_File_" <<
                filename << "\"";
1095         throw GnuplotException( except.str() );
1096         return *this;

```



```

1097     }
1098
1099     std::ostringstream cmdstr;
1100
1101     // command to be sent to gnuplot
1102     if (this->nplots > 0 && this->two_dim == true)
1103         cmdstr << "replot_";
1104     else
1105         cmdstr << "plot_";
1106
1107     cmdstr << "\" " << filename << "\"_using_" << column_x << ":"
1108         << column_y;
1109
1110     if (title == "")
1111         cmdstr << "_notitle_";
1112     else
1113         cmdstr << "_title_" << title << "\"_";
1114
1115     cmdstr << "with_" << this->pstyle << ",_" << filename << "
1116         << "_using_"
1117         << column_x << ":" << column_y << ":" << column_dy <<
1118         "_notitle_with_errorbars";
1119
1120     // Do the actual plot
1121     this->cmd(cmdstr.str());
1122
1123     return *this;
1124 }
1125
1126 // plot x,y pairs with dy errorbars
1127 Gnuplot& Gnuplot::plot_xy_err(const std::vector<double> &x,
1128                                const std::vector<double> &y,
1129                                const std::vector<double> &dy,
1130                                const std::string &title)
1131 {
1132     if (x.size() == 0 || y.size() == 0 || dy.size() == 0)
1133     {
1134         throw GnuplotException("std::vectors too small");
1135         return *this;
1136     }
1137 }

```

---

```

1124 // plot x,y pairs with dy errorbars
1125 Gnuplot& Gnuplot::plot_xy_err(const std::vector<double> &x,
1126                                const std::vector<double> &y,
1127                                const std::vector<double> &dy,
1128                                const std::string &title)
1129 {
1130     if (x.size() == 0 || y.size() == 0 || dy.size() == 0)
1131     {
1132         throw GnuplotException("std::vectors too small");
1133         return *this;
1134     }
1135 }

```

```

1134     }
1135
1136     if (x.size() != y.size() || y.size() != dy.size())
1137     {
1138         throw GnuplotException("Length_of_the_std::vectors_
            differs");
1139         return *this;
1140     }
1141
1142     std::ofstream tmp;
1143     std::string name = create_tmpfile(tmp);
1144     if (name == "")
1145         return *this;
1146
1147     // write the data to file
1148     for (unsigned int i = 0; i < x.size(); i++)
1149         tmp << x[i] << " " << y[i] << " " << dy[i] << std::endl;
1150
1151     tmp.flush();
1152     tmp.close();
1153
1154     // Do the actual plot
1155     this->plotfile_xy_err(name, 1, 2, 3, title);
1156
1157     return *this;
1158 }
1159
1160 //
-----
1161 // Plots a 3d graph from a list of doubles (x y z) saved in a
    file
1162 Gnuplot& Gnuplot::plotfile_xyz(const std::string &filename,
1163                                const int column_x,
1164                                const int column_y,
1165                                const int column_z,
1166                                const std::string &title)
1167 {
1168
1169     // check if file exists
1170     if( !(Gnuplot::file_exists(filename,4)) ) // check existence
        and read permission

```

```

1171     {
1172         std::ostringstream except;
1173         if( !(Gnuplot::file_exists(filename,0)) ) // check
            existence
1174             except << "File_\\"" << filename << "\"_does_not_exist
                ";
1175         else
1176             except << "No_read_permission_for_File_\\"" <<
                filename << "\"";
1177         throw GnuplotException( except.str() );
1178         return *this;
1179     }
1180
1181     std::ostringstream cmdstr;
1182
1183     // command to be sent to gnuplot
1184     if (this->nplots > 0 && this->two_dim == false)
1185         cmdstr << "replot_";
1186     else
1187         cmdstr << "splot_";
1188
1189     cmdstr << "\"\" << filename << "\"_using_" << column_x << ":"
        << column_y << ":" << column_z;
1190
1191     if (title == "")
1192         cmdstr << "_notitle_with_" << this->pstyle;
1193     else
1194         cmdstr << "_title_\\"" << title << "\"_with_" << this->
            pstyle;
1195
1196     // Do the actual plot
1197     this->cmd(cmdstr.str());
1198
1199     return *this;
1200 }
1201
1202 //
-----
1203 // Plots a 3d graph from a list of doubles: x y z
1204 Gnuplot& Gnuplot::plot_xyz(const std::vector<double> &x,
1205                             const std::vector<double> &y,

```

```

1206         const std::vector<double> &z,
1207         const std::string &title)
1208 {
1209     if (x.size() == 0 || y.size() == 0 || z.size() == 0)
1210     {
1211         throw GnuplotException("std::vectors_too_small");
1212         return *this;
1213     }
1214
1215     if (x.size() != y.size() || x.size() != z.size())
1216     {
1217         throw GnuplotException("Length_of_the_std::vectors_
1218                                 differs");
1219         return *this;
1220     }
1221
1222     std::ofstream tmp;
1223     std::string name = create_tmpfile(tmp);
1224     if (name == "")
1225         return *this;
1226
1227     // write the data to file
1228     for (unsigned int i = 0; i < x.size(); i++)
1229     {
1230         tmp << x[i] << " " << y[i] << " " << z[i] <<std::endl;
1231     }
1232
1233     tmp.flush();
1234     tmp.close();
1235
1236
1237     this->plotfile_xyz(name, 1, 2, 3, title);
1238
1239     return *this;
1240 }
1241
1242
1243
1244 //

```

---

```

1245 /// * note that this function is not valid for versions of
      GNUPlot below 4.2
1246 Gnuplot& Gnuplot::plot_image(const unsigned char * ucPicBuf,
1247                               const int iWidth,
1248                               const int iHeight,
1249                               const std::string &title)
1250 {
1251     std::ofstream tmp;
1252     std::string name = create_tmpfile(tmp);
1253     if (name == "")
1254         return *this;
1255
1256     // write the data to file
1257     int iIndex = 0;
1258     for(int iRow = 0; iRow < iHeight; iRow++)
1259     {
1260         for(int iColumn = 0; iColumn < iWidth; iColumn++)
1261         {
1262             tmp << iColumn << "_" << iRow << "_" << static_cast<
                float>(ucPicBuf[iIndex++]) << std::endl;
1263         }
1264     }
1265
1266     tmp.flush();
1267     tmp.close();
1268
1269
1270     std::ostringstream cmdstr;
1271
1272     // command to be sent to gnuplot
1273     if (this->nplots > 0 && this->two_dim == true)
1274         cmdstr << "replot_";
1275     else
1276         cmdstr << "plot_";
1277
1278     if (title == "")
1279         cmdstr << "\"" << name << "\"_with_image";
1280     else
1281         cmdstr << "\"" << name << "\"_title_" << title << "\"_
            with_image";
1282
1283     // Do the actual plot

```

```

1284     this->cmd(cmdstr.str());
1285
1286     return *this;
1287 }
1288
1289 //
-----
1290 // Sends a command to an active gnuplot session
1291 Gnuplot& Gnuplot::cmd(const std::string &cmdstr)
1292 {
1293     if( !(this->valid) )
1294     {
1295         return *this;
1296     }
1297
1298     // int fputs ( const char * str, FILE * stream );
1299     // writes the string str to the stream.
1300     // The function begins copying from the address specified (
1301     // str) until it reaches the
1302     // terminating null character ('\0'). This final null-
1303     // character is not copied to the stream.
1304     fputs( (cmdstr+"\n").c_str(), this->gnucmd );
1305
1306     // int fflush ( FILE * stream );
1307     // If the given stream was open for writing and the last i/o
1308     // operation was an output operation,
1309     // any unwritten data in the output buffer is written to the
1310     // file.
1311     // If the argument is a null pointer, all open files are
1312     // flushed.
1313     // The stream remains open after this call.
1314     fflush(this->gnucmd);
1315
1316     if( cmdstr.find("replot") != std::string::npos )
1317     {
1318         return *this;
1319     }
1320     else if( cmdstr.find("splot") != std::string::npos )
1321     {
1322         this->two_dim = false;
1323     }

```

```

1319         this->nplots++;
1320     }
1321     else if( cmdstr.find("plot") != std::string::npos )
1322     {
1323         this->two_dim = true;
1324         this->nplots++;
1325     }
1326     return *this;
1327 }
1328
1329 //
-----

1330 // Sends a command to an active gnuplot session, identical to cmd
1331      ()
1332 Gnuplot& Gnuplot::operator<<(const std::string &cmdstr)
1333 {
1334     this->cmd(cmdstr);
1335     return *this;
1336 }
1337 //
-----

1338 // Opens up a gnuplot session, ready to receive commands
1339 void Gnuplot::init()
1340 {
1341     // char * getenv ( const char * name );  get value of an
1342     // environment variable
1343     // Retrieves a C string containing the value of the
1344     // environment variable whose
1345     // name is specified as argument.
1346     // If the requested variable is not part of the environment
1347     // list, the function returns a NULL pointer.
1348 #if ( defined(unix) || defined(__unix) || defined(__unix__) ) &&
1349     !defined(__APPLE__)
1350     if (getenv("DISPLAY") == NULL)
1351     {
1352         this->valid = false;
1353         throw GnuplotException("Can't find DISPLAY variable");
1354     }
1355 #endif

```

```

1352
1353 // if gnuplot not available
1354 if (!Gnuplot::get_program_path())
1355 {
1356     this->valid = false;
1357     throw GnuplotException("Can't find gnuplot");
1358 }
1359
1360 // open pipe
1361 std::string tmp = Gnuplot::m_sGnuplotPath + "/" + Gnuplot::
    m_sGnuplotFileName;
1362
1363 // FILE *popen(const char *command, const char *mode);
1364 // The popen() function shall execute the command specified
    by the string command,
1365 // create a pipe between the calling program and the executed
    command, and
1366 // return a pointer to a stream that can be used to either
    read from or write to the pipe.
1367 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)
1368     this->gnucmd = _popen(tmp.c_str(), "w");
1369 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
    defined(__APPLE__)
1370     this->gnucmd = popen(tmp.c_str(), "w");
1371 #endif
1372
1373 // popen() shall return a pointer to an open stream that can
    be used to read or write to the pipe.
1374 // Otherwise, it shall return a null pointer and may set
    errno to indicate the error.
1375 if (!this->gnucmd)
1376 {
1377     this->valid = false;
1378     throw GnuplotException("Couldn't open connection to
        gnuplot");
1379 }
1380
1381 this->nplots = 0;
1382 this->valid = true;
1383 this->smooth = "";
1384

```



```

1385     //set terminal type
1386     this->showonscreen();
1387
1388     return;
1389 }
1390
1391 //
-----

1392 // Find out if a command lives in m_sGnuPlotPath or in PATH
1393 bool Gnuplot::get_program_path()
1394 {
1395     // first look in m_sGnuPlotPath for Gnuplot
1396     std::string tmp = Gnuplot::m_sGnuPlotPath + "/" + Gnuplot::
        m_sGnuPlotFileName;
1397
1398 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
        defined(__TOS_WIN__)
1399     if ( Gnuplot::file_exists(tmp,0) ) // check existence
1400 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
        defined(__APPLE__)
1401     if ( Gnuplot::file_exists(tmp,1) ) // check existence and
        execution permission
1402 #endif
1403     {
1404         return true;
1405     }
1406
1407     // second look in PATH for Gnuplot
1408     char *path;
1409     // Retrieves a C string containing the value of the
        environment variable PATH
1410     path = getenv("PATH");
1411
1412     if (path == NULL)
1413     {
1414         throw GnuplotException("Path is not set");
1415         return false;
1416     }
1417     else
1418     {
1419         std::list<std::string> ls;

```

```

1420         //split path (one long string) into list ls of strings
1421 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
        defined(__TOS_WIN__)
1422         stringtok(ls,path,";");
1423 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
        defined(__APPLE__)
1424         stringtok(ls,path,":");
1425 #endif
1426         // scan list for Gnuplot program files
1427         for (std::list<std::string>::const_iterator i = ls.begin
                (); i != ls.end(); ++i)
1428         {
1429             tmp = (*i) + "/" + Gnuplot::m_sGnUPlotFileName;
1430 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
                defined(__TOS_WIN__)
1431                 if ( Gnuplot::file_exists(tmp,0) ) // check existence
1432 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
                defined(__APPLE__)
1433                 if ( Gnuplot::file_exists(tmp,1) ) // check existence
                    and execution permission
1434 #endif
1435                 {
1436                     Gnuplot::m_sGnUPlotPath = *i; // set
                        m_sGnUPlotPath
1437                     return true;
1438                 }
1439         }
1440
1441         tmp = "Can't find gnuplot neither in PATH nor in \" +
                Gnuplot::m_sGnUPlotPath + "\"";
1442         throw GnuplotException(tmp);
1443
1444         Gnuplot::m_sGnUPlotPath = "";
1445         return false;
1446     }
1447 }
1448
1449 //
-----
1450 // check if file exists
1451 bool Gnuplot::file_exists(const std::string &filename, int mode)

```

```

1452 {
1453     if ( mode < 0 || mode > 7)
1454     {
1455         throw std::runtime_error("In function \"Gnuplot::
            file_exists\": mode has to be an integer between 0 and
            7");
1456         return false;
1457     }
1458
1459     // int _access(const char *path, int mode);
1460     // returns 0 if the file has the given mode,
1461     // it returns -1 if the named file does not exist or is not
        accessible in the given mode
1462     // mode = 0 (F_OK) (default): checks file for existence only
1463     // mode = 1 (X_OK): execution permission
1464     // mode = 2 (W_OK): write permission
1465     // mode = 4 (R_OK): read permission
1466     // mode = 6      : read and write permission
1467     // mode = 7      : read, write and execution permission
1468 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
        defined(__TOS_WIN__)
1469     if (_access(filename.c_str(), mode) == 0)
1470 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
        defined(__APPLE__)
1471     if (access(filename.c_str(), mode) == 0)
1472 #endif
1473     {
1474         return true;
1475     }
1476     else
1477     {
1478         return false;
1479     }
1480
1481 }
1482
1483 //
        -----
1484 // Opens a temporary file
1485 std::string Gnuplot::create_tmpfile(std::ofstream &tmp)
1486 {

```

```

1487 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)
1488     char name[] = "gnuplotiXXXXXX"; //tmp file in working
        directory
1489 #elif defined(unix) || defined(__unix) || defined(__unix__) ||
    defined(__APPLE__)
1490     char name[] = "/tmp/gnuplotiXXXXXX"; // tmp file in /tmp
1491 #endif
1492
1493 // check if maximum number of temporary files reached
1494 if (Gnuplot::tmpfile_num == GP_MAX_TMP_FILES - 1)
1495 {
1496     std::ostringstream except;
1497     except << "Maximum number of temporary files reached ("
        << GP_MAX_TMP_FILES
1498         << "): cannot open more files" << std::endl;
1499
1500     throw GnuplotException( except.str() );
1501     return "";
1502 }
1503
1504 //int mkstemp(char *name);
1505 // shall replace the contents of the string pointed to by "
    name" by a unique filename,
1506 // and return a file descriptor for the file open for reading
    and writing.
1507 // Otherwise, -1 shall be returned if no suitable file could
    be created.
1508 // The string in template should look like a filename with
    six trailing 'X' s;
1509 // mkstemp() replaces each 'X' with a character from the
    portable filename character set.
1510 // The characters are chosen such that the resulting name
    does not duplicate the name of an existing file at the time
    of a call to mkstemp()
1511
1512
1513 // open temporary files for output
1514 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) ||
    defined(__TOS_WIN__)
1515     if (_mktemp(name) == NULL)
1516 #elif defined(unix) || defined(__unix) || defined(__unix__) ||

```

```

        defined(__APPLE__)
1517     if (mkstemp(name) == -1)
1518 #endif
1519     {
1520         std::ostringstream except;
1521         except << "Cannot_create_temporary_file_" << name << "
            \"";
1522         throw GnuplotException(except.str());
1523         return "";
1524     }
1525
1526     tmp.open(name);
1527     if (tmp.bad())
1528     {
1529         std::ostringstream except;
1530         except << "Cannot_create_temporary_file_" << name << "
            \"";
1531         throw GnuplotException(except.str());
1532         return "";
1533     }
1534
1535     // Save the temporary filename
1536     this->tmpfile_list.push_back(name);
1537     Gnuplot::tmpfile_num++;
1538
1539     return name;
1540 }

```

Apresenta-se na listagem 6.32 o arquivo com código da classe CPoco.

Listing 6.32: Arquivo de cabeçalho da classe CPoco.

```

1 #ifndef CPOCO_H_
2 #define CPOCO_H_
3
4
5 class CPoco{
6
7     protected:
8
9         double Rw;
10
11     public:
12

```

```

13         CPoco (){};
14
15         void SetRw ( double _Rw);
16         double GetRw();
17
18         ~CPoco (){};
19
20
21
22
23 };
24
25 #endif

```

Apresenta-se na listagem 6.33 o arquivo de implementação da classe CPoco.

Listing 6.33: Arquivo de implementação da classe CPoco.

```

1 #include "CPoco.h"
2
3 void CPoco::SetRw(double _Rw)
4 {
5     Rw = _Rw;
6 }
7
8 double CPoco::GetRw()
9 {
10     return Rw;
11 }

```

Apresenta-se na listagem 6.34 o arquivo com código da classe CReservatorio.

Listing 6.34: Arquivo de cabeçalho da classe CReservatorio.

```

1 #ifndef CRESERVATORIO_H_
2 #define CRESERVATORIO_H_
3
4 class CReservatorio{
5
6     protected:
7
8         double ct;
9
10    public:
11
12        CReservatorio(){};

```

```

13
14         void SetCt(double _ct);
15         double GetCt();
16
17         ~CReservatorio(){};
18
19 };
20
21 #endif

```

Apresenta-se na listagem 6.35 o arquivo de implementação da classe CReservatorio.

Listing 6.35: Arquivo de implementação da classe CReservatorio.

```

1 #include "CReservatorio.h"
2
3 void CReservatorio::SetCt(double _ct)
4 {
5
6     ct = _ct;
7
8 }
9
10 double CReservatorio::GetCt()
11 {
12
13     return ct;
14
15 }

```

Apresenta-se na listagem 6.36 o arquivo com código da classe CRocha.

Listing 6.36: Arquivo de cabeçalho da classe CRocha.

```

1 #ifndef CROCHA_H_
2 #define CROCHA_H_
3
4 class CRocha{
5
6     protected:
7
8         double k, phi, Re, h;
9
10    public:
11
12        CRocha(){};

```

```
13
14         void SetK(double _k);
15         double GetK();
16         void SetPhi(double _phi);
17         double GetPhi();
18         void SetRe(double _Re);
19         double GetRe();
20         void Seth(double _h);
21         double Geth();
22
23         ~CRocha(){};
24
25 };
26
27 #endif
```

---

Apresenta-se na listagem 6.37 o arquivo de implementação da classe CRocha.

Listing 6.37: Arquivo de implementação da classe CRocha.

---

```
1 #include "CRocha.h"
2
3 void CRocha::SetK(double _k)
4 {
5     k = _k;
6 }
7
8 void CRocha::SetPhi(double _phi)
9 {
10     phi = _phi;
11 }
12
13 void CRocha::SetRe(double _Re)
14 {
15     Re = _Re;
16 }
17
18 void CRocha::Seth(double _h)
19 {
20     h = _h;
21 }
22
23 double CRocha::GetK()
24 {
```



---

```

25         return k;
26     }
27
28     double CRocha::GetPhi()
29     {
30         return phi;
31     }
32
33     double CRocha::GetRe()
34     {
35         return Re;
36     }
37
38     double CRocha::Geth()
39     {
40         return h;
41     }

```

---

Apresenta-se na listagem 6.38 o arquivo com código da classe CSolverInfluxo.

Listing 6.38: Arquivo de cabeçalho da classe CSolverInfluxo.

---

```

1 #ifndef CEEXECUTA_H_
2 #define CEEXECUTA_H_
3
4 #include <vector>
5
6 #include "CRocha.h"
7 #include "CPoco.h"
8 #include "CFluido.h"
9 #include "CFetkovich.h"
10 #include "CCarterTracy.h"
11 #include "CAdimensional.h"
12 #include "CGnuplot.h"
13 #include "CReservatorio.h"
14 #include "CCarterTracy.h"
15
16 class CSolverInfluxo{
17
18     protected:
19
20         std::vector <double> P, T;
21
22         int fluxo;

```

```

23
24         CGnuplot plot_carter, plot_fetkovich;
25         CFluido fluido;
26         CPoco poco;
27         CRocha rocha;
28         CReservatorio reservatorio;
29         CCarterTracy carter;
30         CFetkovich fetkovich;
31         CAdimensional adimensional;
32
33
34     public:
35
36         CSolverInfluxo(){};
37
38         void EntradaDados();
39         void Executa();
40
41         ~CSolverInfluxo(){};
42
43 };
44
45 #endif

```

Apresenta-se na listagem 6.39 o arquivo de implementação da classe CSolverInfluxo.

Listing 6.39: Arquivo de implementação da classe CSolverInfluxo.

```

1 #include "CSolverInfluxo.h"
2
3 #include <iostream>
4 #include <fstream>
5 #include <filesystem>
6 #include <string>
7
8 using namespace std;
9
10 void CSolverInfluxo::EntradaDados()
11 {
12
13     cout << "
14
15     -----
16     " << endl;
17     cout << "|

```

```

        " " << endl;
15    cout << " | " << endl;
        " " << endl;
16    cout << " |
        -----
        | \n" << endl;

17
18    bool errado = true;
19
20    string path = "./dados/";
21
22    cout << "\nArquivos Disponíveis\n" << endl;
23
24    for (const auto & file : filesystem::directory_iterator(path)
        )
25        cout << file.path() << endl;
26
27    cout << endl;
28
29    cout << "Entre com nome dos arquivos de dados de pressão\n" << endl;
30
31    string nomeArquivo;
32
33    getline(cin, nomeArquivo);
34
35    nomeArquivo = "dados/" + nomeArquivo;
36
37    ifstream in;
38
39    in.open(nomeArquivo, ifstream::in);
40
41    double tmp;
42
43    while (!in.eof())
44    {
45        in >> tmp;
46        P.push_back(tmp);
47    }
48
49    in.close();
50

```

```

51      cout << "
           -----
           " << endl;
52      cout << "|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           XXXXXXXXXXXX|" << endl;
53      cout << "|XXXXXXXXXXXXXXXXXXXXDados_de_pressao_carregadosXXXXXXXXXX
           XXXXXXXXXXXX|" << endl;
54      cout << "|
           -----
           |\n" << endl;
55
56      cout << "Entre com nome dos arquivos de dados de tempo\n"
           << endl;
57
58      getline (cin, nomeArquivo);
59
60      nomeArquivo="dados/"+nomeArquivo;
61
62      in.open(nomeArquivo, fstream::in);
63
64
65      while (!in.eof())
66      {
67          in >> tmp;
68          T.push_back(tmp);
69      }
70
71      in.close();
72
73      cout << "
           -----
           " << endl;
74      cout << "|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           XXXXXXXXXXXX|" << endl;
75      cout << "|XXXXXXXXXXXXXXXXXXXXDados_de_tempo_carregadosXXXXXXXXXX
           XXXXXXXXXXXX|" << endl;
76      cout << "|
           -----
           |\n" << endl;
77
78      cout << "Entre com arquivos de dados do reservatorio\n"
           << endl;

```

```

79         getline (cin, nomeArquivo);
80
81
82         nomeArquivo="dados/"+nomeArquivo;
83
84         in.open(nomeArquivo , fstream::in);
85
86         in >> tmp;
87         rocha.SetRe(tmp);
88         in >> tmp;
89         poco.SetRw(tmp);
90         in >> tmp;
91         rocha.Seth(tmp);
92         in >> tmp;
93         rocha.SetPhi(tmp);
94         in >> tmp;
95         rocha.SetK(tmp);
96         in >> tmp;
97         fluido.SetMi(tmp);
98         in >> tmp;
99         reservatorio.SetCt(tmp);
100
101         cout << endl;
102
103     }
104
105 void CSolverInfluxo::Executa()
106 {
107
108     EntradaDados();
109
110     bool rodar = true;
111
112     cout << "Qual metodo para resolucao de influxo deseja utilizar?" << endl;
113
114     cout << "
115
116     -----
117     " << endl;
118     cout << "|oooooooooooooooooooooooooooooooooooooooooooooooooooo
119     ooooooooo|" << endl;
120     cout << "|uuuu1u-uuCarter&Tracyuu|uu2u-uuFetkovichuu|u0u-
```

```

        _sair_|||||" << endl;
117     cout << "|

        -----
        |\n" << endl;

118
119     int escolha, j=0;
120     cin >> escolha;
121
122     bool time = false;
123
124     vector <double> TD;

125
126         TD = adimensional.CalcTD(rocha.GetK(),
                                   rocha.GetPhi(), fluido.GetMi(),
                                   reservatorio.GetCt(), poco.GetRw(), T);

127
128     double _U;

129
130     _U = adimensional.CalcU(rocha.GetPhi(),
                              reservatorio.GetCt(), rocha.Geth(),
                              poco.GetRw());

131
132     vector <double> _delp;

133
134     _delp = adimensional.CalcdelP(P);

135
136     vector <double> _tda;

137
138     _tda = adimensional.CalcTDA(poco.GetRw(),
                                   rocha.GetRe(), TD);

139
140     vector <double> _pd;

141
142     _pd = adimensional.CalcPD(_tda, TD, rocha
                               .GetRe(), poco.GetRw());

143
144     vector <double> _pdbarra;

145
146     _pdbarra = adimensional.CalcPDbarra(_tda,
                                           TD, rocha.GetRe(), poco.GetRw());

147
148     while (rodar)

```



```

181         plot_carter.Title("Carter-_-Tracy
182         ");
183         plot_carter.Legend("inside_-left_-
184         top_-nobox");
185         plot_carter.set_style();
186         plot_carter.set_xlabel("t(Tempo)"
187         );
188         plot_carter.set_ylabel("We(
189         influxo_acumulado)");
190
191         plot_carter.plot_xy(T, _wecarter,
192         "Carter-_-Tracy");
193         plot_carter.savetops("CarterTracy
194         ");
195         cout << "\nAperte_ENTER_para_
196         continuar" << endl;
197         cin.get();
198         cin.get();
199
200         plot_carter.set_style("lines");
201         plot_carter.replot();
202         plot_carter.showonscreen();
203         plot_carter.plot_xy(T, _wecarter)
204         ;
205         plot_carter.savetops("CarterTracy
206         (linhas)");
207         cout << "Aperte_ENTER_para_
208         continuar" << endl;
209         cin.get();
210
211     }
212
213     else
214     if (escolha == 2)
215     {
216
217         cout << "Qual_regime_de_fluxo?_
218         1_-Permanente_|_2_-_
219         Pseudopermanente_" << endl;
220
221         bool teste = true;
222         double escolha2;

```



```
211
212         cin >> escolha2;
213
214         while (teste)
215             {
216                 if (escolha2 ==
217                     1)
218                 {
219                     fluxo =
220                         true;
221                     teste =
222                         false;
223                 }
224                 else
225                 if (escolha2 ==
226                     2)
227                 {
228                     fluxo =
229                         false;
230                     teste =
231                         false;
232                 }
233                 else
234                 {
235                     cout << "opcao_
236                         invalida!!!!"
237                         << endl;
238                     cout << "Qual_
239                         regime_de_fluxo
240                         ?_1-_
241                         Permanente_|_2_
242                         -_
243                         Pseudopermanente
244                         _" << endl;
245
246                     cin >> escolha2;
247                 }
248             }
```

```
239
240         double _We;
241
242         _We = adimensional.CalcWe
            (rocha.GetRe(), poco.
            GetRw(), rocha.Geth(),
            rocha.GetPhi());
243
244         double _Wei;
245
246         _Wei = adimensional.
            CalcWei(reservatorio.
            GetCt(), _We, P[0]);
247
248         vector <double>
            _delpbarra;
249
250         _delpbarra = adimensional.
            .DeltaPbarra(P);
251
252         double _J;
253
254         _J = adimensional.CalcJ(
            rocha.GetK(), rocha.
            Geth(), rocha.GetRe(),
            poco.GetRw(), fluido.
            GetMi(), fluxo);
255
256         vector <double> deltat;
257
258         deltat = adimensional.
            DeltaT(T);
259
260         vector <double> Fet;
261
262         Fet = fetkovich.
            CalcFetkovic(_We, _Wei,
            _delpbarra, _J, deltat
            , P);
263
264         cout << endl;
265
```

```

266         for (double i=0; i < Fet.
                size();i++)
267             cout << "We_=" << Fet[i]
                << endl;

268
269         Gnuplot::Terminal("qt");

270
271         plot_fetkovich.Grid();
272         plot_fetkovich.showonscreen();
273         plot_fetkovich.Title("Fetkovich")
                ;
274         plot_fetkovich.Legend("inside_
                left_top_nobox");
275         plot_fetkovich.set_style();
276         plot_fetkovich.set_xlabel("t(
                Tempo)");
277         plot_fetkovich.set_ylabel("We(
                influxo_acumulado)");

278
279         plot_fetkovich.plot_xy(T, Fet, "
                Fetkovich");
280         plot_fetkovich.savetops("
                Fetkovich");
281         cout << "\nAperte ENTER para
                continuar" << endl;
282         cin.get();
283         cin.get();

284
285         plot_fetkovich.set_style("lines")
                ;
286         plot_fetkovich.replot();
287         plot_fetkovich.showonscreen();
288         plot_fetkovich.plot_xy(T, Fet);
289         plot_fetkovich.savetops("
                Fetkovich(linhas)");
290         cout << "Aperte ENTER para
                continuar" << endl;
291         cin.get();

292
293     }
294     else

```

```

295         if (escolha == 0)
296             rodar = false;
297         else
298         {
299             cout << "Opcao invalida" << endl;
300             cout << "\nQual metodo para resolucao de
                fluxo deseja utilizar?" << endl;
301
302             cout << "
                -----
                " << endl;
303             cout << "|oooooooooooooooooooooooooooooooooooo
                ooooooooooooooooooooooooooooooooooooo|" << endl;
304             cout << "|uuuu1-uuCarter&Tracyuu|uu2-uu
                Fetkovichh uu|0-ussairuu|" << endl;
305             cout << "|
                -----
                |\n" << endl;
306
307             cin >> escolha;
308         }
309         j++;
310         time = true;
311
312         if (j==2)
313             break;
314
315     }
316
317 }

```

Apresenta-se na listagem 6.40 o programa que usa a classe main.

Listing 6.40: Arquivo de implementação da função main().

```

1 #include "CSolverInfluxo.h"
2
3 int main () {
4
5     CSolverInfluxo executa;
6
7     executa.Executa();
8
9 }

```

# Capítulo 7

## Teste

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

### 7.1 Teste 1: Escolha do Software

Na Figura 7.1, o usuário deverá escolher a pasta (não há ordem pré-estabelecida) com o respectivo modelo (“Van Everdingen” ou “Carter-Tracy” e “Fetkovich”) de aquíferos analíticos a serem utilizados para os cálculos e geração de resultados (*plots*).

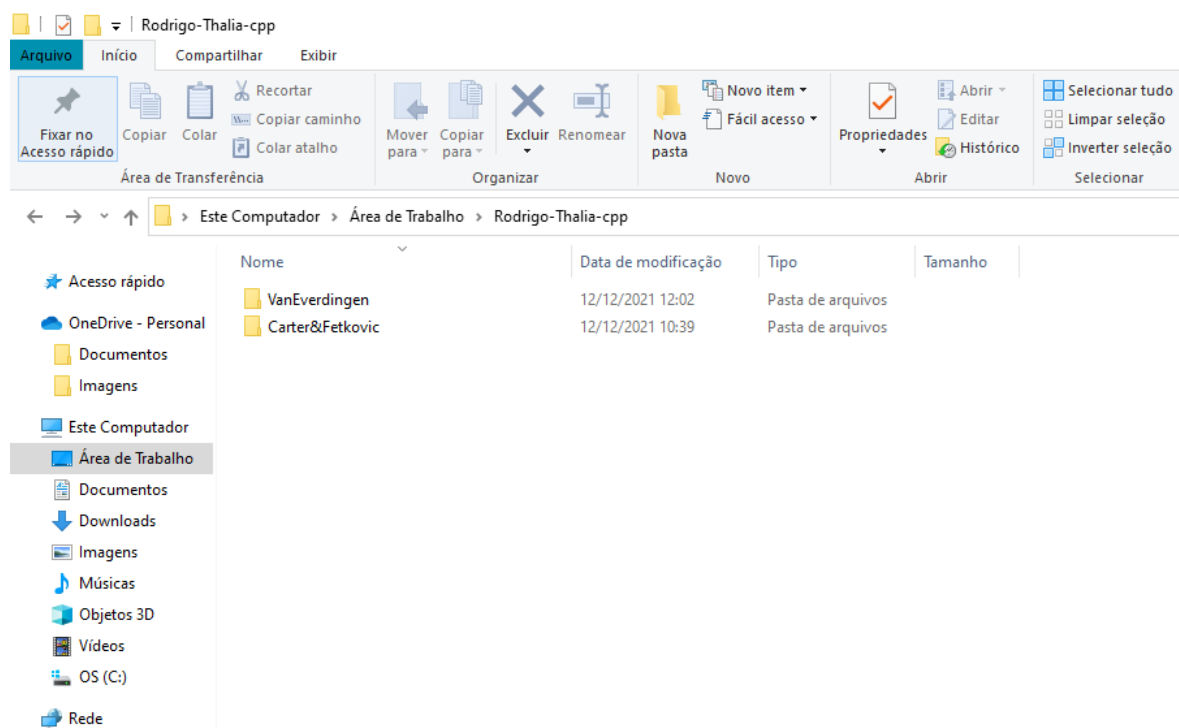


Figura 7.1: Tela que exhibe as duas pastas que armazenam os *softwares* utilizados neste trabalho

## 7.2 Teste 2: *Software* - Van Everdingen & Hurst

### 7.2.1 Passo inicial

Este *software* se apresenta em modo texto. A Figura 7.2 retrata a configuração inicial do mesmo.

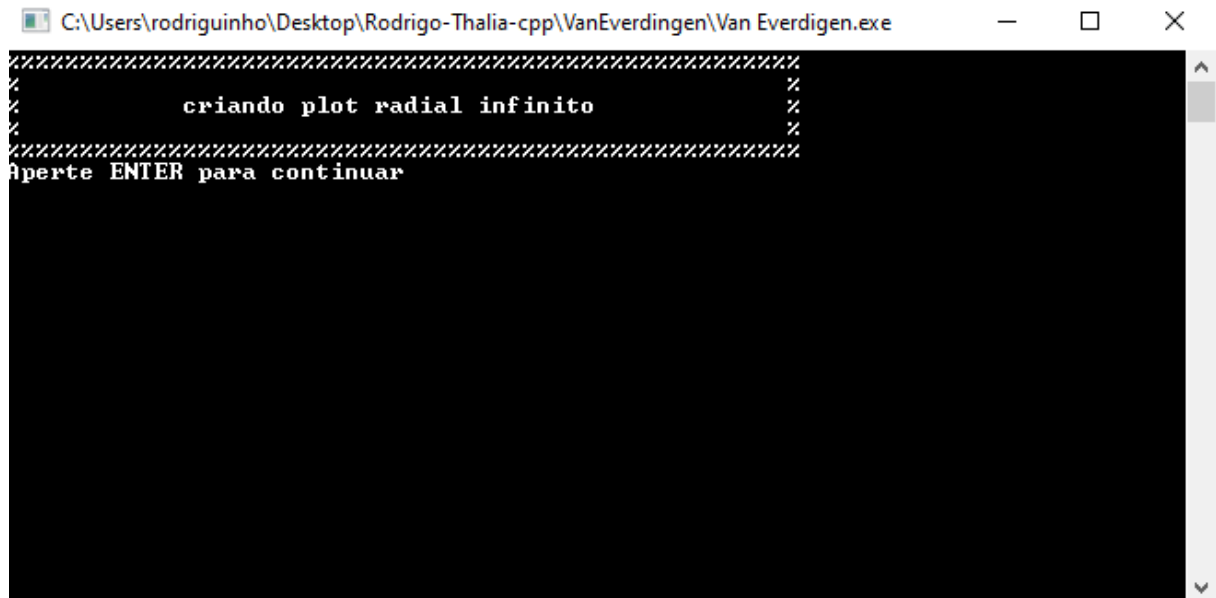


Figura 7.2: Tela do software mostrando a inicialização para os cálculos do modelo de Van Everdingen & Hurst

### 7.2.2 Cálculo do influxo de água adimensional para o aquífero radial infinito x aquífero radial finito selado com seus respectivos *plots*

A Figura 7.3 mostra a tela com o cálculo do influxo de água adimensional para um aquífero radial infinito x aquífero radial finito selado, assim como o gráfico dos parâmetros calculados.

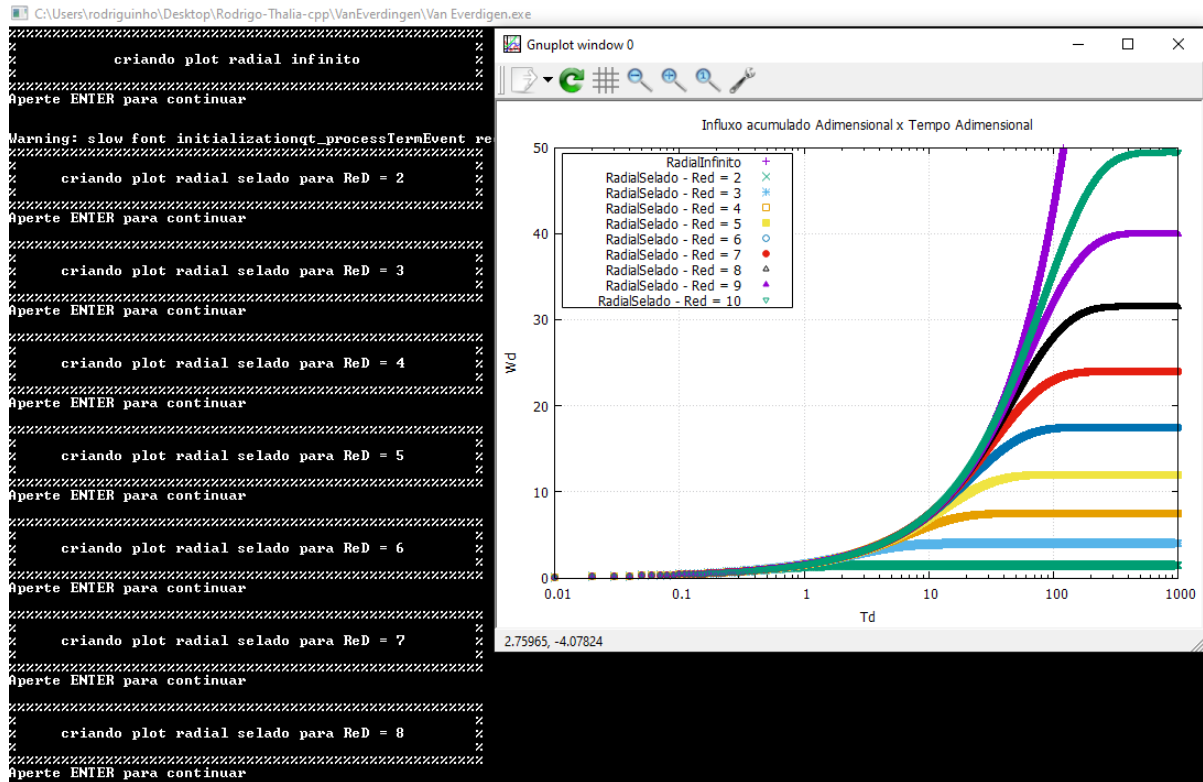


Figura 7.3: Influxo acumulado adimensional  $W_D$  para aquífero radial infinito x aquífero radial finito selado em função do tempo adimensional  $t_D$  e do tamanho do aquífero dado pela razão  $r_{eD} = r_e/r_0$

### 7.2.3 Cálculo do influxo de água adimensional para o aquífero radial infinito x aquífero radial finito com manutenção de pressão com seus respectivos *plots*

A Figura 7.4 mostra a tela com o cálculo do influxo de água adimensional para um aquífero radial infinito x aquífero radial finito com manutenção de pressão (realimentado), assim como o gráfico dos parâmetros calculados.

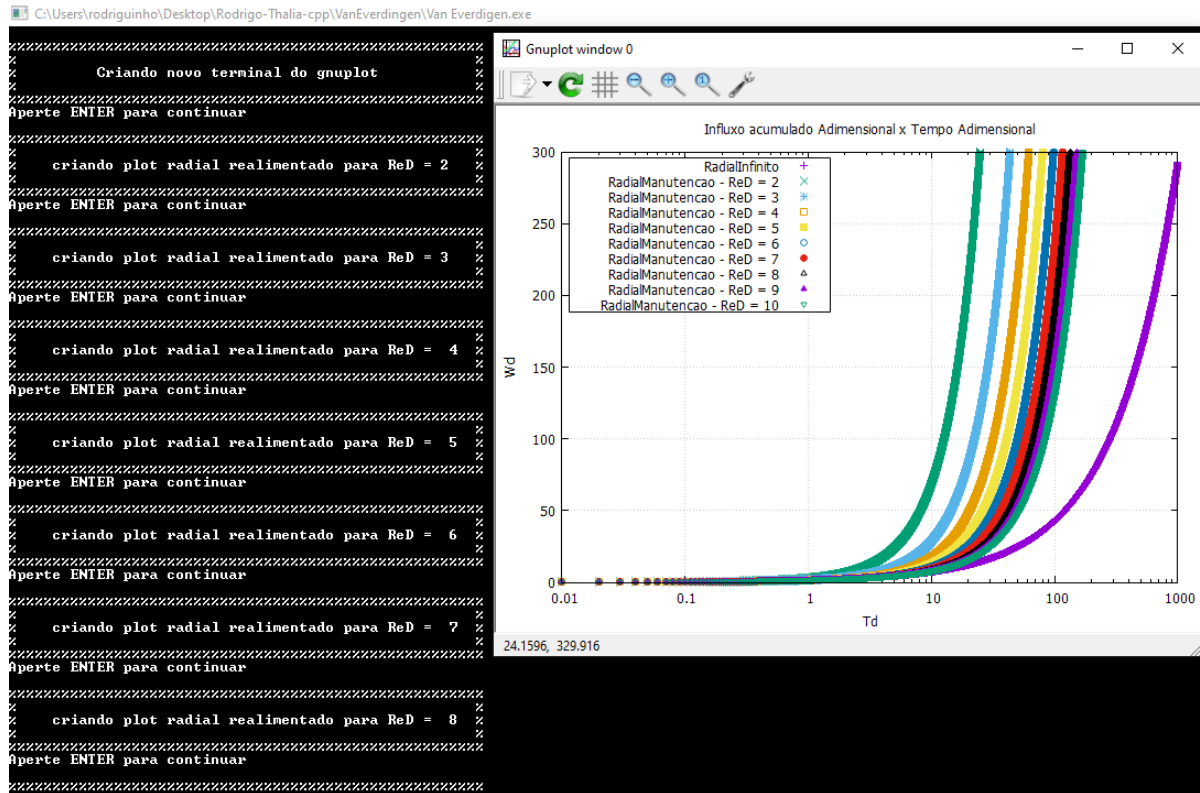


Figura 7.4: Gráfico do influxo acumulado adimensional  $W_D$  para aquífero radial infinito x aquífero radial finito realimentado em função do tempo adimensional  $t_D$  e do tamanho do aquífero dado pela razão  $r_{eD} = r_e/r_0$

#### 7.2.4 Cálculo o influxo acumulado de água adimensional para o aquífero linear infinito, selado e com manutenção de pressão

A Figura 7.5 expõe o comportamento do influxo acumulado adimensional em função do tipo de condição de contorno externa (C.C.E) do modelo infinito, selado e com manutenção de pressão (realimentado):



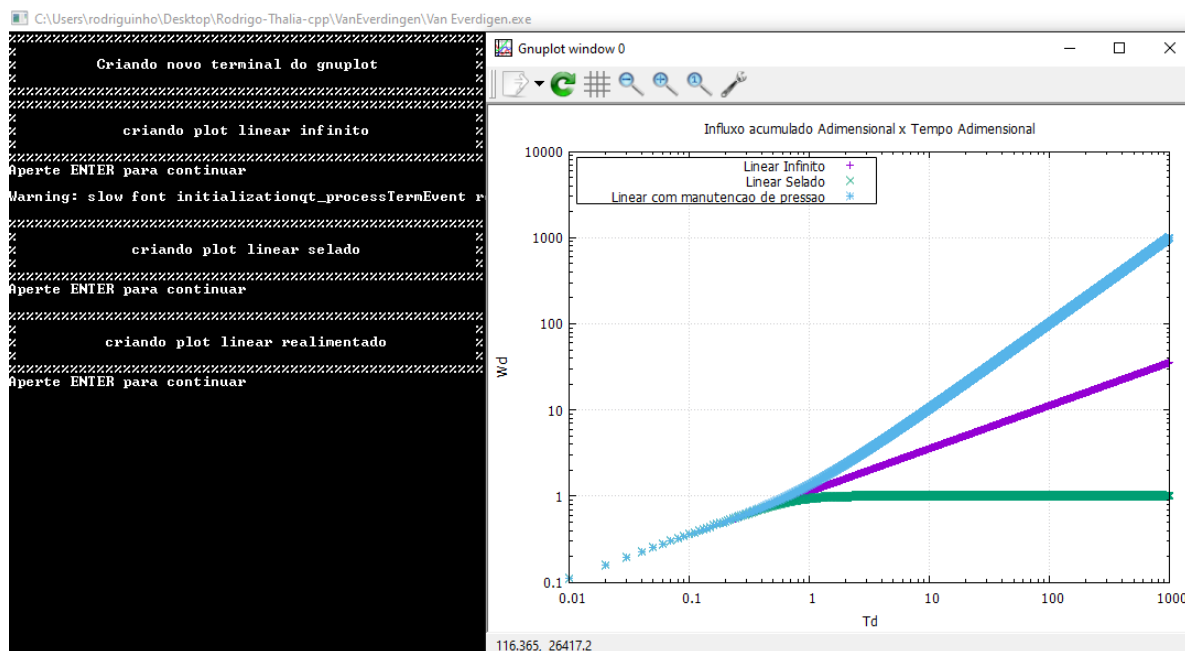


Figura 7.5: Gráfico do influxo adimensional  $W_D$  para aquífero linear em função do tempo adimensional  $t_D$

### 7.2.5 Salvando *plots* da simulação, destruindo arquivos temporários e saída do programa

A Figura 7.6 mostra o processo simples de exportar os resultados (*plots*) da simulação em um arquivo de disco. Os mesmos ficarão salvos na pasta `./Src` que está localizado na pasta `VanEverdingen`. Em seguida, os arquivos serão destruídos e o programa será encerrado.

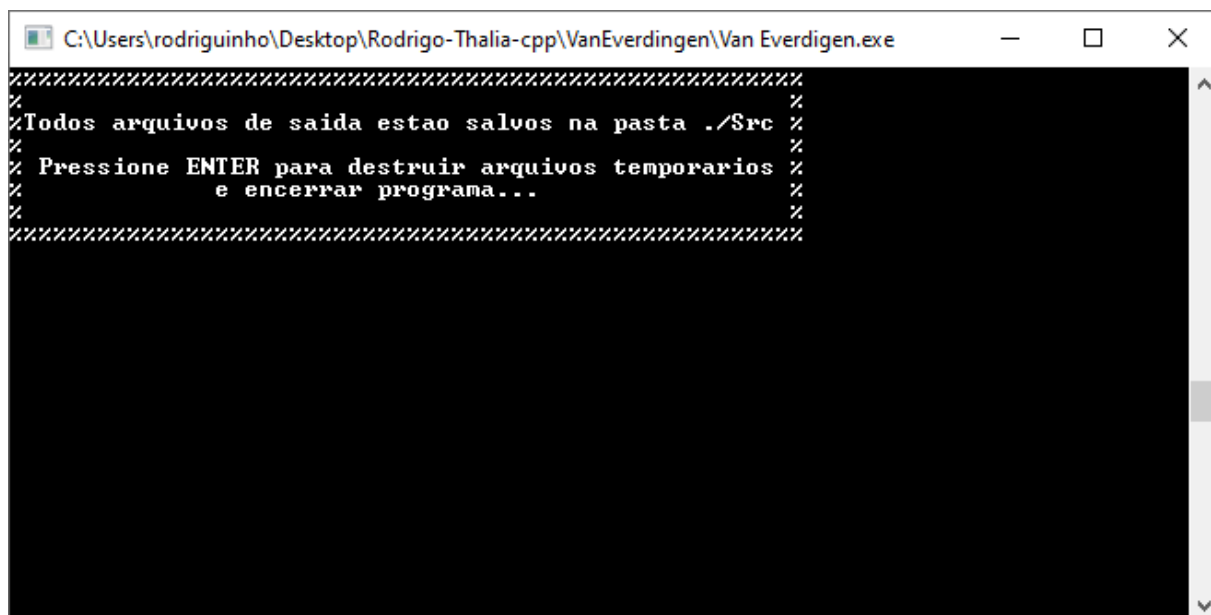


Figura 7.6: Tela que reproduz o encerramento, destruição dos arquivos temporários e o armazenamento dos *plots* realizados pelo *software*.

### 7.3 Teste 3: *Software* - Carter-Tracy e Fetkovich

O problema dado para a realização dos cálculos utilizados para gerar os resultados (*plots*) por esses dois modelos são fornecidos pelo Capítulo 6 do livro de Engenharia de Reservatórios de autoria do [Rosa, 2011].

**Exemplo 6.5 e Exemplo 6.6** — Um reservatório de petróleo com  $762m$  de raio é circundado por um **aqüífero radial selado** com as seguintes características:

Raio .....	$r_e = 6.096\ m$
Espessura .....	$h = 18,3m$
Porosidade .....	$f = 0,22$
Permeabilidade .....	$k = 100md$
Viscosidade da água .....	$\mu = 0.30cp$
Compressibilidade da formação .....	$c_f = 56,9 \times 10^{-6} (kgf/cm^2)$
Compressibilidade da água .....	$c_w = 42,7 \times 10^{-6} (kgf/cm^2)$

Calcular, usando o modelo Fetkovich e Carter-Tracy, o influxo acumulado de água após 500 dias, baseando-se no histórico de pressões médias no contato óleo/água mostrado na tabela a seguir:

Tabela 7.1: Dados de tempo  $t$  (d) e pressão  $p$  ( $kgf/cm^2$ )

$t$ (d)	0	100	200	300	400	500
$p$ ( $kgf/cm^2$ )	246,13	245,43	244,44	243,18	242,19	240,51

#### 7.3.1 Passo inicial

Este *software* se apresenta em modo texto. A Figura 7.7 retrata a configuração inicial do mesmo.

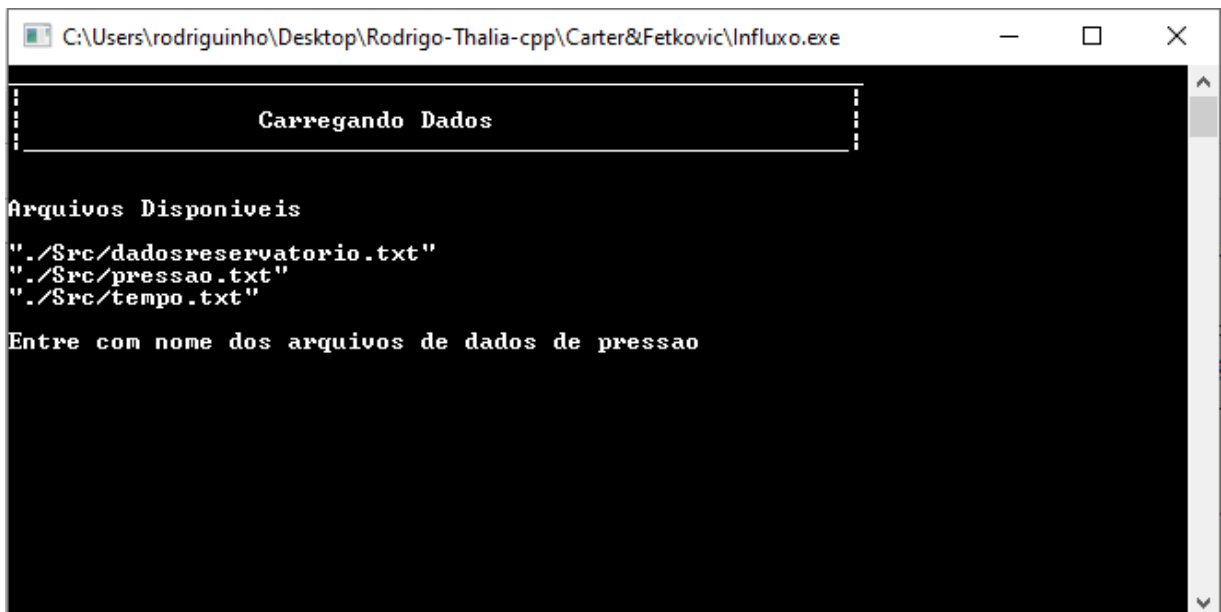


Figura 7.7: Tela do software mostrando a inicialização para os cálculos dos modelos de Carter-Tracy e Fetkovich

### 7.3.2 Carregando os dados do reservatório (dados iniciais) e dados da tabela (pressão e tempo)

A Figura 7.8 a seguir mostra o carregamento dos dados de reservatório (iniciais) e dos dados da tabela (pressão e tempo) visando a resolução do problema de influxo de água de um aquífero radial selado:

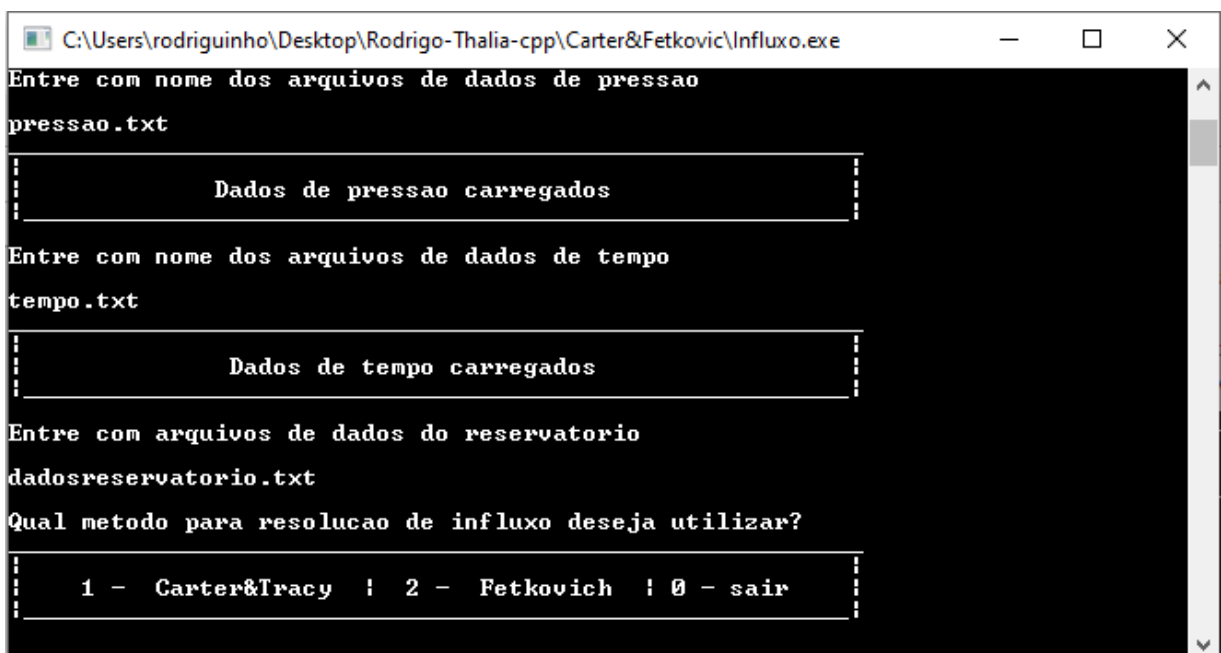


Figura 7.8: Carregamento dos dados de reservatório (iniciais) e dos dados da tabela (pressão e tempo)

### 7.3.3 Cálculo do influxo acumulado de água do modelo de Carter-Tracy

A Figura 7.9 mostra o cálculo do  $W_e$  (influxo acumulado de água) do modelo Carter-Tracy após 500 dias, baseando-se no histórico de pressões médias no contato óleo/água da tabela 7.1:

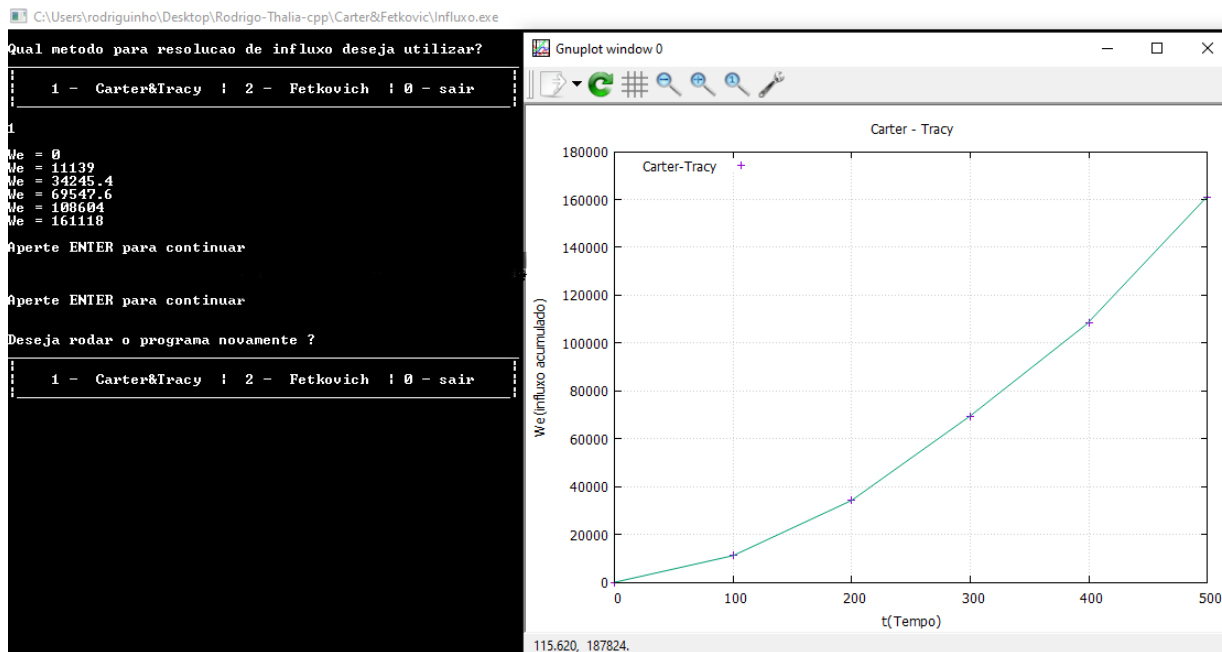


Figura 7.9: Cálculo do influxo de água  $W_e$  ao longo do tempo  $t$ .

Feito isso, os resultados (*plots*) ficarão salvos na pasta `"/Src"` que está localizada na pasta `"Carter&Fetkovic"`.

### 7.3.4 Cálculo do influxo acumulado de água do modelo de Fetkovich e o encerramento do *software*

Após calcular o influxo acumulado de água em Carter-Tracy (o usuário pode escolher qual método calcular inicialmente, seja ele Carter-Tracy ou Fetkovich), a Figura 7.10 mostra o cálculo do  $W_e$  (influxo acumulado de água) do modelo de Fetkovich após 500 dias, baseando-se no histórico de pressões médias no contato óleo/água da tabela 7.1:

Após a escolha do modelo, o usuário terá que escolher o regime de fluxo. Como o problema se apresenta como *radial finito selado* e Fetkovich admite o fluxo como pseudopermanente em sem modelo, logo a escolha deverá ser a opção "2".

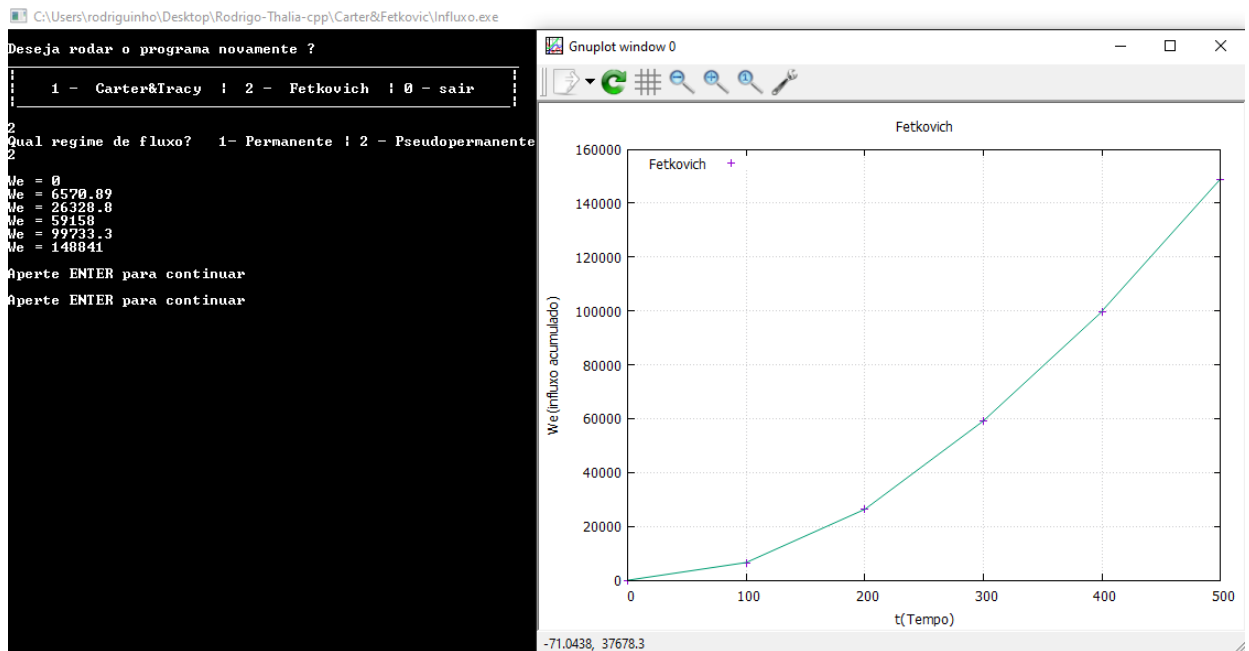


Figura 7.10: Cálculo do influxo de água  $W_e$  ao longo do tempo  $t$ .

Feito isso, os resultados (*plots*) ficarão salvos na pasta ".\Src" que está localizada na pasta "Carter&Fetkovic". E em seguida, o usuário terá que apertar a tecla "enter" por duas vezes consecutivas, para que o software se encerre.

# Capítulo 8

## Documentação

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

### 8.1 Documentação do usuário

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

#### 8.1.1 Como instalar o software

Para instalar os softwares, execute o seguinte passo a passo:

- Em Windows: Faça o download de um compilador, como por exemplo o g++ (por linhas de comando utilizando o MinGw); e o Dev C++, disponível em <https://dev-c.softonic.com.br/> . Compile o simulador e execute-o.
- Em Linux: Abra o terminal, vá para o diretório onde está o simulador, faça a compilação, e em seguida a execução.

#### 8.1.2 Como rodar o software

No capítulo 7 têm todas as informações necessárias para que se possa rodar os softwares.

### 8.2 Documentação para desenvolvedor

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

### 8.2.1 Dependências

Para compilar o software é necessário atender as seguintes dependências:

- Instalar o compilador g++ da GNU disponível em <http://gcc.gnu.org>. Para instalar no GNU/Linux use o comando `yum install gcc`.
- Biblioteca CGnuplot; os arquivos para acesso a biblioteca CGnuplot 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`.
- Os arquivos com dados de reservatório, podem ser mudados desde que haja coerência com os mesmos, ou seja, os parâmetros de um reservatório e suas unidades de medida precisam ser respeitadas. Se essas alterações forem realizadas, é preciso que as mesmas sejam feitas diretamente no código e nos arquivos de dados de reservatório com extensão `txt`.

### 8.2.2 Como gerar a documentação usando doxygen

A documentação do código do software deve ser feita seguindo o padrão JAVADOC, através do software `doxygen`, que lê os arquivos com os códigos (`*.h` e `*.cpp`) e gera uma documentação muito útil e de fácil navegação no formato `html`.

Apresenta-se a seguir algumas imagens com as telas das saídas geradas pelo software `doxygen`.

Modelos de Aquíferos Analíticos: Van Everdingen 1.0

Main Page

Classes ▾

Files ▾

Q Search

Class Index

C | G

C

CFormaReservatorio  
CReservatorioLinearInfinito

CReservatorioLinearManutencao  
CReservatorioLinearSelado  
CReservatorioRadialInfinito  
CReservatorioRadialManutencao

CReservatorioRadialSelado  
CSolverVanEverdingen  
CStehfest

G

Gnuplot  
GnuplotException


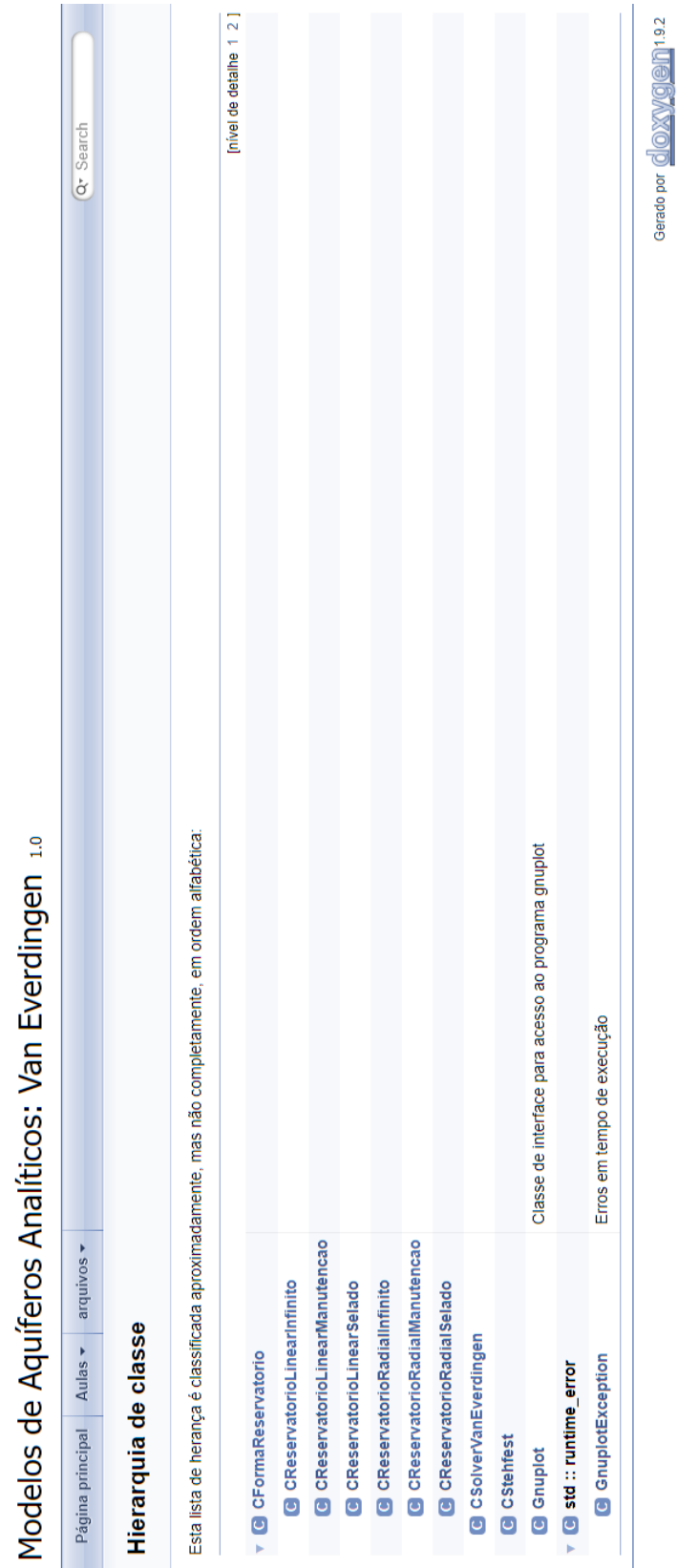
Generated by  1.9.2

Figura 8.1: Lista de classes de Van Everdingen - Doxygen





Modelos de Aquíferos Analíticos: Carter-Tracy e Fetkovich<sup>1.0</sup>

Main Page

Classes ▾

Files ▾

Q Search

Class Index

C | G

C

CAdimensional

CCarterTracy

G

CFetkovich

CFuido

CPoco

CReservatorio

CRocha

CSolverInfluo

Gnuplot

GnuplotException


Generated by  1.9.2

Figura 8.3: Lista de classes de Carter-Tracy e Fetkovich - Doxygen

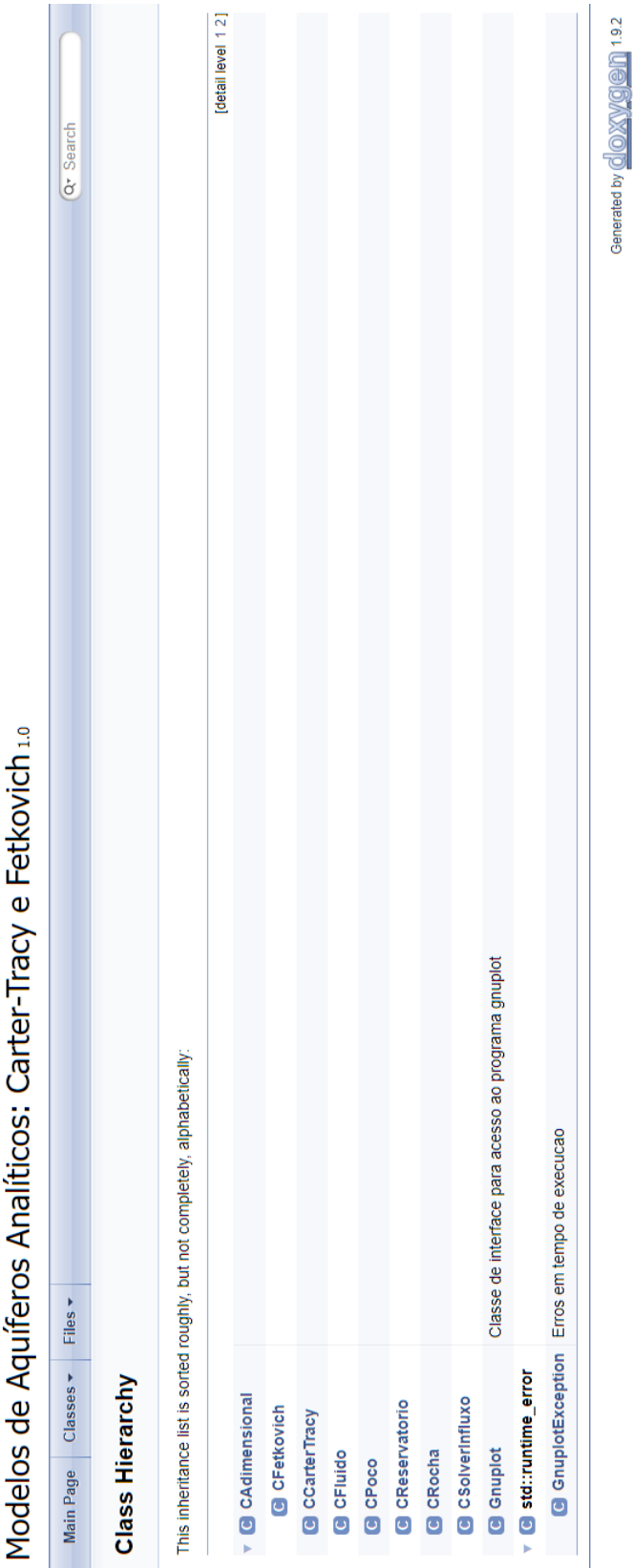


Figura 8.4: Hierarquia de classes de Carter-Tracy e Fetkovich - Doxygen



# Referências Bibliográficas

- [Ahmed, 2006] Ahmed, T. (2006). *reservoir engineering*. Elsevier Inc. 9, 32
- [Allard and Chen, 1988] Allard, D. and Chen, S. (1988). Calculation of water influx for bottomwater drive reservoirs. *SPE Reservoir Engineering*, 3(02):369–379. 14
- [Blaha and Rumbaugh, 2006] Blaha, M. and Rumbaugh, J. (2006). *Modelagem e Projetos Baseados em Objetos com UML 2*. Campus, Rio de Janeiro. 45
- [Carter and Tracy, 1960] Carter, R. and Tracy, G. (1960). An improved method for calculating water influx. *Transactions of the AIME*, 219(01):415–417. 24
- [Dake, 1983] Dake, L. (1983). *fundamentals of reservoir engineering*. ELSEVIER SCIENCE B.V. 10
- [Dietz, 1965] Dietz, D. (1965). Determination of average reservoir pressure from build-up surveys. *Journal of Petroleum Technology*, 17(08):955–959. 32
- [Ezekwe, 2011] Ezekwe, N. (2011). *Petroleum reservoir engineering practice*. Pearson Education, Inc. 11
- [Fetkovich, 1971] Fetkovich, M. (1971). A simplified approach to water influx calculations-finite aquifer systems. *Journal of Petroleum Technology*, 23(07):814–828. 27
- [Okotie and Ikporo, 2019] Okotie, S. and Ikporo, B. (2019). *reservoir engineering*. Springer International Publishing. 8, 9, 24, 27
- [Rosa, 2011] Rosa, A. J. (2011). *engenharia de reservatorios de petroleo*. Editora Inter-ciencia Ltda. 10, 24, 27, 32, 247
- [Rumbaugh et al., 1994] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorenzen, W. (1994). *Modelagem e Projetos Baseados em Objetos*. Edit. Campus, Rio de Janeiro. 45
- [Schilthuis, 1936] Schilthuis, R. J. (1936). Active oil and reservoir energy. *Transactions of the AIME*, 118(01):33–52. 32
- [Stehfest, 1970] Stehfest, H. (1970). Algorithm 368: Numerical inversion of laplace transforms [d5]. *Communications of the ACM*, 13(1):47–49. 14, 21