

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

PROJETO DE ENGENHARIA  
DESENVOLVIMENTO DO SOFTWARE  
PROGRAMA EM C++ PARA AVALIAÇÃO DE FORMAÇÕES POR  
DADOS DE TESTES DE PRESSÃO  
TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA

Versão 1:  
LEONAM DOS SANTOS BRAGA  
RENAN MARCOS DE LIMA FILHO  
Versão 2:  
ANDRÉIA DE PAULA MARTUSCELLI

Prof. André Duarte Bueno

MACAÉ - RJ  
Dezembro - 2023

# Sumário

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introdução</b>                                     | <b>1</b>  |
| 1.1      | Escopo do problema . . . . .                          | 1         |
| 1.2      | Objetivos . . . . .                                   | 2         |
| 1.3      | Metodologia utilizada . . . . .                       | 2         |
| <b>2</b> | <b>Concepção</b>                                      | <b>4</b>  |
| 2.1      | Nome do sistema/produto . . . . .                     | 4         |
| 2.2      | Especificação . . . . .                               | 4         |
| 2.3      | Requisitos . . . . .                                  | 5         |
| 2.3.1    | Requisitos funcionais . . . . .                       | 5         |
| 2.3.2    | Requisitos não funcionais . . . . .                   | 5         |
| 2.4      | Casos de uso . . . . .                                | 5         |
| 2.4.1    | Diagrama de caso de uso geral . . . . .               | 6         |
| 2.4.2    | Diagrama de caso de uso específico . . . . .          | 6         |
| <b>3</b> | <b>Elaboração</b>                                     | <b>9</b>  |
| 3.1      | Análise de domínio . . . . .                          | 9         |
| 3.2      | Formulação teórica . . . . .                          | 10        |
| 3.2.1    | Teste de Pressão . . . . .                            | 10        |
| 3.2.2    | Método de Horner . . . . .                            | 13        |
| 3.2.3    | Método de MDH . . . . .                               | 15        |
| 3.3      | Identificação de pacotes – assuntos . . . . .         | 18        |
| 3.4      | Diagrama de pacotes – assuntos . . . . .              | 18        |
| <b>4</b> | <b>AOO – Análise Orientada a Objeto</b>               | <b>20</b> |
| 4.1      | Diagramas de classes . . . . .                        | 20        |
| 4.1.1    | Dicionário de classes . . . . .                       | 22        |
| 4.2      | Diagrama de sequência – eventos e mensagens . . . . . | 27        |
| 4.2.1    | Diagrama de sequência geral . . . . .                 | 27        |
| 4.3      | Diagrama de comunicação . . . . .                     | 28        |
| 4.4      | Diagrama de máquina de estado . . . . .               | 29        |
| 4.5      | Diagrama de atividades . . . . .                      | 30        |

|   |           |
|---|-----------|
| <b>5 Projeto</b>  | <b>31</b> |
| 5.1 Projeto do sistema . . . . .  | 31        |
| 5.2 Projeto orientado a objeto – POO . . . . .                              | 32        |
| 5.3 Diagrama de componentes . . . . .                                       | 34        |
| 5.4 Diagrama de implantação . . . . .                                       | 34        |
| <b>6 Ciclos Construção - Implementação</b>                                  | <b>36</b> |
| 6.1 Código fonte . . . . .  | 36        |
| <b>7 Teste</b>  | <b>89</b> |
| 7.1 Teste 1: Teste no Windows . . . . .                                     | 89        |
| 7.2 Teste 2: Teste no GNU/Linux (Gerando o gráfico com o Gnuplot) . . . . . | 93        |
| <b>8 Documentação para o Desenvolvedor</b>                                  | <b>99</b> |
| 8.1 Documentação para desenvolvedor . . . . .                               | 99        |
| 8.1.1 Dependências . . . . .  | 99        |
| 8.1.2 Documentação usando doxygen . . . . .                                 | 99        |
| 8.2 Sugestões para trabalhos futuros . . . . .                              | 100       |
| 8.3 Como gerar a documentação usando doxygen . . . . .                      | 100       |

# Listas de Figuras

|     |  |    |
|-----|--|----|
| 1.1 | Etapas para o desenvolvimento do software - <i>projeto de engenharia</i> . . . . .   | 3  |
| 2.1 | Diagrama de caso de uso – Caso de uso geral . . . . .  | 8  |
| 2.2 | Diagrama de caso de uso específico – Diagrama de caso de uso específico<br>-Escolhendo o método e variando um parâmetro de entrada, sendo salvo<br>novamente os resultados com essa variação. . . . .  | 8  |
| 3.1 | Exemplo de teste de poço convencional, onde é possível observar o comportamento da pressão em função da variação da vazão de produção. As regiões 1 e 3 mostram etapas com o poço fechado ( $q=0$ ), onde é possível ver um aumento da pressão no reservatório. Já na região 2, o poço está produzindo com uma vazão constante, causando uma queda na pressão do reservatório. . . . . | 11 |
| 3.2 | Esquema de vazão e comportamento da pressão em um polo submetido a um teste de pressão . . . . .   | 12 |
| 3.3 | Gráfico semi-logarítmico obtido pelo Método de Horner, que fornece parâmetros referentes ao reservatório . . . . .   | 13 |
| 3.4 | Gráfico de MDH . . . . .   | 17 |
| 3.5 | Diagrama de Pacotes . . . . .  | 19 |
| 4.1 | Diagrama de classes . . . . .  | 21 |
| 4.2 | Diagrama de sequência . . . . .  | 28 |
| 4.3 | Diagrama de comunicação . . . . .  | 29 |
| 4.4 | Diagrama de máquina de estado. . . . .   | 30 |
| 4.5 | Diagrama de atividades da classe CCaracterizacaoReservatorio. . . . .  | 30 |
| 5.1 | Diagrama de componentes . . . . .  | 34 |
| 5.2 | Diagrama de implantação . . . . .  | 35 |
| 7.1 | Tela do programa mostrando a entrada de dados. . . . .   | 90 |
| 7.2 | Tela do programa mostrando seleção do arquivo de entrada, e posterior regressão linear e o ajuste da curva. . . . .  | 91 |
| 7.3 | Tela do programa mostrando a possibilidade de geração do gráfico. . . . .  | 91 |
| 7.4 | Gráfico de $Pws$ vs. $\log\Delta t$ gerado pelo Gnuplot. . . . .   | 92 |

|      |  |     |
|------|--|-----|
| 7.5  | Tela do programa mostrando a exportação dos resultados para um arquivo de saída .dat, a caracterização do reservatório, e a possibilidade de se variar um parâmetro. . . . . | 92  |
| 7.6  | Diretório onde se encontram as listagens e onde são salvos os arquivos de saída (em destaque). . . . .   | 93  |
| 7.7  | Tela do programa mostrando abertura do compilador Visual Code. . . . .   | 94  |
| 7.8  | Tela do programa mostrando o Visual Code compilando o make do programa, com as entradas dos dados. . . . .   | 94  |
| 7.9  | Tela do programa mostrando a leitura dos dados do Registrador de pressão. .  | 95  |
| 7.10 | Gráfico de $P_{ws}$ vs. $\log\Delta t$ gerado pelo Gnuplot. . . . .  | 95  |
| 7.11 | Tela do programa mostrando o nome do arquivo de saída a ser salvo, onde ele está e o seu resultado. . . . .  | 96  |
| 7.12 | Tela do programa mostrando escolha método Horner e entrada de dados. .   | 96  |
| 7.13 | Entrada de dados do Registrador de pressão para o caso de Horner e a escolha do gráfico. . . . .   | 97  |
| 7.14 | Gráfico de $P_{ws}$ vs. $\log[(t + \Delta t)/\Delta t]$ gerado pelo Gnuplot. . . . .   | 97  |
| 7.15 | Tela do programa mostrando o nome do arquivo de saída a ser salvo. . . . .   | 98  |
| 8.1  | Lista de classes pelo doxygen. . . . .   | 101 |

# **Lista de Tabelas**

|     |                                  |    |
|-----|----------------------------------|----|
| 2.1 | Caso de uso 1                    | 6  |
| 3.1 | Variáveis no Sistema de Unidades | 13 |

# Listagens

|      |   |    |
|------|---|----|
| 6.1  | Arquivo de cabeçalho da classe CPoco.h . . . . .                              | 36 |
| 6.2  | Arquivo de implementação da classe CPoco.cpp. . . . .                         | 37 |
| 6.3  | Arquivo de cabeçalho da classe CReservatorio.h. . . . .                       | 40 |
| 6.4  | Arquivo de implementação da classe CReservatorio.cpp. . . . .                 | 41 |
| 6.5  | Arquivo de cabeçalho da classe CFluido.h. . . . .                             | 44 |
| 6.6  | Arquivo de implementação da classe CFluido.cpp. . . . .                       | 45 |
| 6.7  | Arquivo de cabeçalho da classe CDadosRegistradorPressao.h. . . . .            | 47 |
| 6.8  | Arquivo de implementação da classe CDadosRegistradorPressao.cpp. . . . .      | 49 |
| 6.9  | Arquivo de cabeçalho da classe CEstatistica.h. . . . .                        | 51 |
| 6.10 | Arquivo de implementação da classe CEstatistica.cpp. . . . .                  | 52 |
| 6.11 | Arquivo de cabeçalho da classe CAjusteCurvaMinimosQuadrados.h. . . . .        | 53 |
| 6.12 | Arquivo de implementação da classe CAjusteCurvaMinimosQuadrados.cpp.          | 54 |
| 6.13 | Arquivo de cabeçalho da classe CAjusteCurva.h. . . . .                        | 56 |
| 6.14 | Arquivo de implementação da classe CAjusteCurva.cpp. . . . .                  | 57 |
| 6.15 | Arquivo de cabeçalho da classe CBuildUp.h. . . . .                            | 62 |
| 6.16 | Arquivo de implementação da classe CBuildUp.cpp. . . . .                      | 62 |
| 6.17 | Arquivo de cabeçalho da classe CMetodo.h. . . . .                             | 63 |
| 6.18 | Arquivo de cabeçalho da classe CMetodoMDH.h. . . . .                          | 63 |
| 6.19 | Arquivo de implementação da classe CMetodoMDH.cpp. . . . .                    | 64 |
| 6.20 | Arquivo de cabeçalho da classe CMetodoHorner.h. . . . .                       | 65 |
| 6.21 | Arquivo de implementação da classe CMetodoHorner.cpp. . . . .                 | 65 |
| 6.22 | Arquivo de cabeçalho da classe CCaracterizacaoReservatorio.h. . . . .         | 66 |
| 6.23 | Arquivo de implementação da classe CCaracterizacaoReservatorio.cpp. . . . .   | 67 |
| 6.24 | Arquivo de cabeçalho da classe CSimuladorAnaliseTestePressao.h. . . . .       | 68 |
| 6.25 | Arquivo de implementação da classe CSimuladorAnaliseTestePressao.cpp. . . . . | 70 |
| 6.26 | Arquivo de implementação da função main(). . . . .                            | 87 |

# Capítulo 1

## Introdução

Neste projeto de engenharia será desenvolvido um software que possibilita ao usuário calcular de forma eficaz e simplificada os parâmetros de um reservatório. Tal método consiste na análise do comportamento da pressão do reservatório ao longo do tempo sob efeito de diferentes vazões de produção. Todas essas informações são importantes para a construção de um modelo confiável e o mais próximo do real visando avaliar o potencial de um reservatório. Este projeto, frente a outros já desenvolvidos, possui dois diferenciais: a realização dos cálculos dos parâmetros de reservatório por meio dos métodos de Horner e método MDH e a comparação de seus resultados; e a possibilidade de variação de um determinado parâmetro, e a visualização de como tal variação influenciará no comportamento do reservatório.

### 1.1 Escopo do problema

O estudo do comportamento das pressões nas formações produtoras de petróleo é de fundamental importância para a engenharia de reservatórios. Por meio de análises destes testes, é possível determinar potencialidades e características dos reservatórios, avaliar as reservas disponíveis de hidrocarbonetos, bem como fornecer previsões de produção dos fluidos existentes.

O domínio dos conceitos básicos da teoria de fluxo de fluidos através de meios porosos é necessário para aplicação e desenvolvimento das técnicas atuais de análise de dados de pressão em poços, particularmente no que se refere à avaliação das potencialidades e determinação das características dos horizontes produtores.

Por ocasião da descoberta de novos campos petrolíferos, as decisões sobre os recursos a serem investidos, dependem, por exemplo, de avaliações realizadas por intermédio de testes em poços. Estes testes duram pouco tempo e geram resultados confiáveis para avaliar a exploração de tais jazidas.

## 1.2 Objetivos

Os objetivos deste trabalho são:

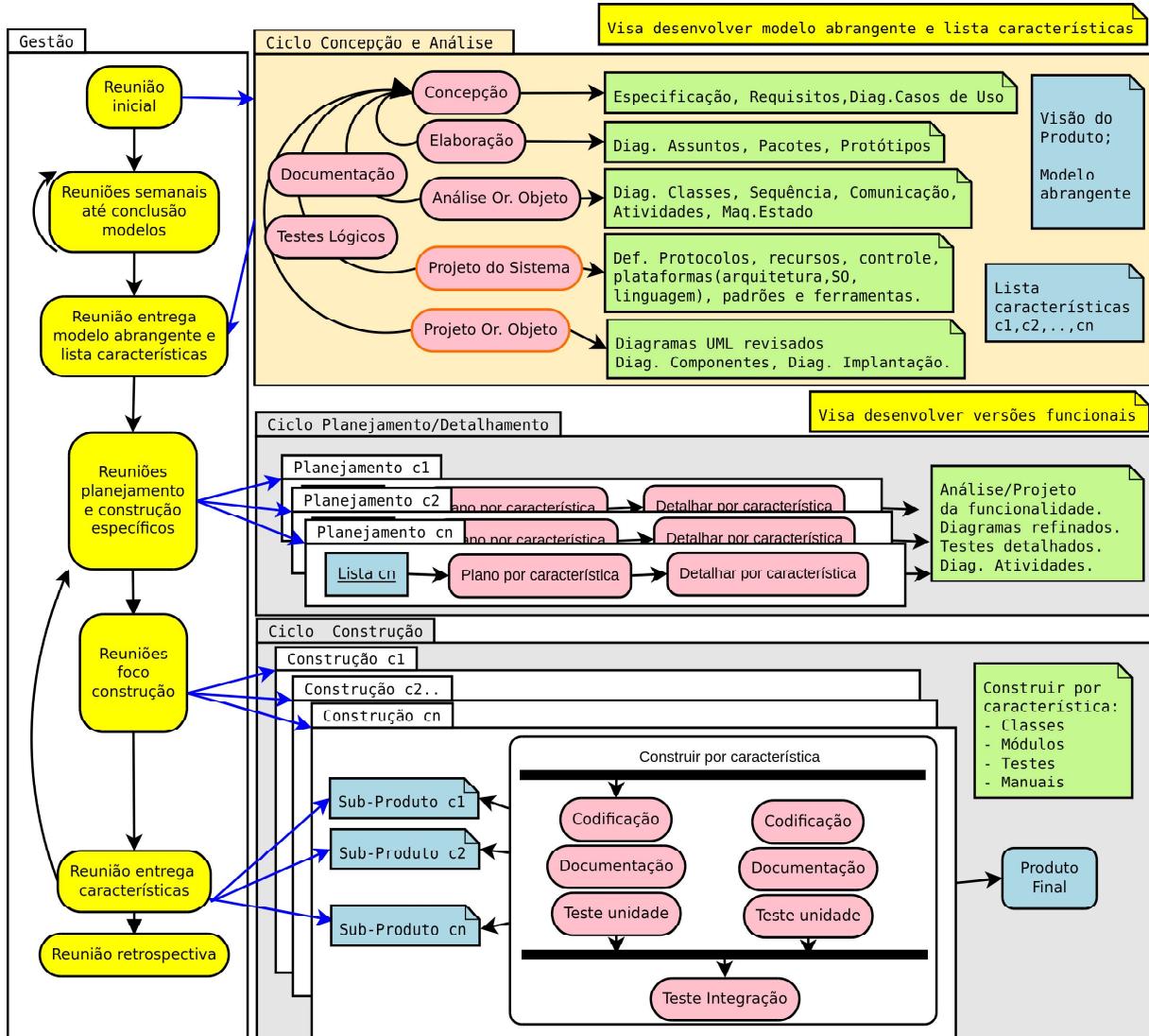
- Objetivo geral:
  - Criar um software capaz de fornecer ao usuário, através da análise dos dados obtidos em testes de pressão em poços utilizando método de Horner ou método MDH, possibilitando estimar as dimensões do campo e sua potencialidade econômica.
- Objetivos específicos:
  - Calcular a permeabilidade efetiva do meio poroso;
  - Calcular o fator de película do poço;
  - Calcular índice de produtividade;
  - Calcular a pressão inicial e média na região drenada pelo poço;
  - Estimar efeito de estocagem e sua duração;
  - Apresentar de forma gráfica o comportamento da pressão ao longo do tempo;
  - Criar um banco de dados que possibilite ao usuário comparar o reservatório que está sendo analisado com outros anteriormente analisados pelo programa;
  - Salvar os resultados em modo texto no disco;
  - Utilizar o método de Horner e o método de MDH, de acordo com a escolha do usuário;
  - Realizar a caracterização do reservatório de acordo com os resultados obtidos com aquele método escolhido.

## 1.3 Metodologia utilizada

O software a ser desenvolvido utiliza a metodologia de engenharia de software apresentada pelo Prof. André Bueno na disciplina de programação e ilustrado na Figura 1.1. Note que o “Ciclo de Concepção e Análise” é composto por diversas partes representadas neste trabalho em diferentes capítulos. Os ciclos de planejamento/detalhamento tem seu próprio capítulo, assim como o ciclo de construção - implementação.

Esta metodologia é utilizada nas disciplinas:

- LEP01447 : Programação Orientada a Objeto em C++.
- LEP01446 : Programação Prática.

Figura 1.1: Etapas para o desenvolvimento do software - *projeto de engenharia*

# Capítulo 2

## Concepção

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

### 2.1 Nome do sistema/produto

|                        |  |
|------------------------|--|
| Nome                   | Simulador de teste build-up                            |
| Componentes principais | Sistema para cálculo de parâmetros de reservatório     |
| Missão                 | Ferramenta de ensino na área de avaliação de formações |

### 2.2 Especificação

O software descrito nesse projeto tem como função simular os cálculos relacionados a parâmetros de reservatório, por meio de dados referentes a um teste de crescimento de pressão em um poço produtor, importados de um arquivo texto. O programa realizará uma regressão linear semi-logarítmica entre a pressão medida no fundo do poço versus o tempo, gerando gráficos com auxílio do software externo *gnuplot*. O usuário fornecerá através do arquivo .dat, valores iniciais sobre o sistema poço-reservatório: de porosidade, espessura do reservatório, viscosidade, compressibilidade total e fator volume de formação do fluido. Além disso, ele escolherá qual método será utilizado para a geração dos resultados. Através da visualização e interpretação dos gráficos gerados, o programa terá como saída os valores de permeabilidade efetiva, pressão inicial, raio externo do reservatório, índice de produtividade, fator película de formação e efeito de estocagem (caso estes ocorram), além dos gráficos da variação da pressão no poço versus tempo.

O desenvolvimento do programa será feito utilizando a linguagem de programação C++, por se tratar de uma linguagem eficiente e possibilitar o reaproveitamento de códigos já desenvolvidos. Por se tratar de um programa científico, será utilizada uma interface em modo texto, que permitirá a entrada e saída de dados de forma simplificada.

Licença: o presente software tem licença GPL 2.0, conforme consta no site <http://softwarelivre.org>.

## 2.3 Requisitos

Apresenta-se nesta seção os requisitos funcionais e não funcionais.

### 2.3.1 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

|              |  |
|--------------|--|
| <b>RF-01</b> | O sistema deverá conter uma base de dados referentes a um teste de crescimento de pressão.   |
| <b>RF-02</b> | O usuário deverá informar o nome do arquivo texto de onde serão lidas as informações do teste de pressão                             |
| <b>RF-03</b> | O usuário deverá entrar com os parâmetros do fluido, do reservatório e do poço.  |
| <b>RF-04</b> | O usuário deverá entrar com os parâmetros do fluido, do reservatório e do poço.  |
| <b>RF-05</b> | O usuário deverá ter liberdade para escolher o método de Horner ou MDH.  |
| <b>RF-06</b> | O usuário poderá plotar um gráfico semi-logarítmico. 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 fórmulas provenientes da disciplina de Avaliação de Formações.          |
| <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> . |

## 2.4 Casos de uso

Um caso de uso descreve um ou mais cenários de uso do software, exemplos de uso, como o sistema interage com usuários externos (atores). Ademais, ele deve representar uma seqüência típica de uso do programa (a execução de determinadas tarefas-padrão).

Também deve representar as exceções, casos em que o usuário comete algum erro, em que o sistema não consegue realizar as tarefas solicitadas. A Tabela 2.1 mostra os itens a serem incluídos na descrição do caso de uso.

Tabela 2.1: Caso de uso 1

| Nome do caso de uso:   | Cálculo de parâmetros do reservatório  |
|------------------------|--|
| Resumo/descrição:      | Determinação de parâmetros do reservatório através de uma regressão linear semi-logarítmica com dados de teste de crescimento de pressão   |
| Etapas:                | <ol style="list-style-type: none"> <li>1. Importar dados do teste de pressão de arquivo texto.</li> <li>2. Fornecer dados do sistema poço-reservatório.</li> <li><b>3. Selecionar o método de Horner ou método de MDH.</b></li> <li>4. Gerar gráficos, <b>conforme o método escolhido.</b></li> <li>5. Exibir resultados na tela exportar para um arquivo .dat.</li> <li><b>6. Caracterizar o reservatório também com a escolha do método MDH.</b></li> <li>7. Selecionar uma variável e definir um intervalo de variação para a mesma.</li> <li>8. Gerar novos resultados e analisar a interferência da variação nestes.</li> </ol> |
| Cenários alternativos: | Um cenário alternativo envolve uma entrada errada do usuário (por exemplo, valores de entrada dos dados do fluido como viscosidade, valores de pressão negativos, fator volume de formação menor que a unidade). O programa apresentará um bug quando valores que deveriam ser positivos forem menores que zero. Outro exemplo ocorre na entrada de dados para criação da função semi-logarítmica. O programa apresentará um bug quando for determinar o logaritmo de -1. O software apresentará uma mensagem de erro.   |

#### 2.4.1 Diagrama de caso de uso geral

O diagrama de caso de uso geral da Figura 2.1 mostra o usuário inserindo os dados do teste de pressão, fornecendo os dados do fluido e do reservatório, necessários para os cálculos dos parâmetros do reservatório a serem determinados e caracterizando o reservatório. Demonstra a interação do usuário com o programa.

#### 2.4.2 Diagrama de caso de uso específico

O diagrama de caso de uso específico da Figura 2.2 mostra o usuário inserindo os dados. O usuário escolhe qual método quer utilizar para calcular os parâmetros, assim o simulador realiza a regressão e calcula os parâmetros do reservatório de acordo com essa escolha. E também o gráfico de pressão pelo tempo é gerado de adequado para o método

escolhido, utilizando um sistema externo, o *gnuplot*. O usuário após obter os resultados, pode selecionar a variação de algum dado de entrada, como porosidade, viscosidade e com isso obter novos resultados que podem ser salvos, obtidos por essa variação.

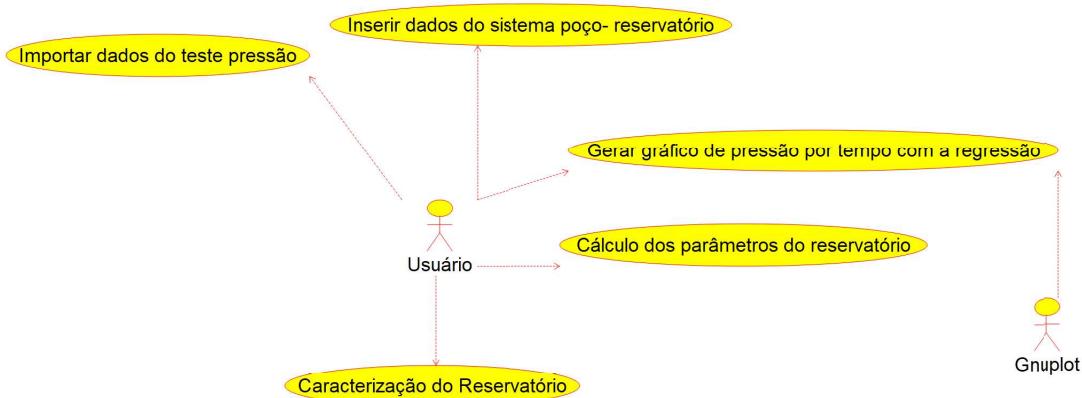


Figura 2.1: Diagrama de caso de uso – Caso de uso geral



Figura 2.2: Diagrama de caso de uso específico – Diagrama de caso de uso específico -Escolhendo o método e variando um parâmetro de entrada, sendo salvo novamente os resultados com essa variação.

# Capítulo 3

## Elaboração

Na elaboração serão apresentados o estudo de conceitos relacionados ao sistema a ser desenvolvido, a análise de domínio e a identificação de pacotes.

### 3.1 Análise de domínio

- O *software*, com o auxílio do corpo docente, será desenvolvido na Universidade Estadual do Norte Fluminense Darcy Ribeiro.
- O *software* envolve conceitos abordados nas disciplinas de Engenharia de Reservatórios, Avaliação de Formações, Cálculo Numérico e Programação Orientada a Objeto em C++.

A Engenharia de Reservatórios estuda o escoamento dos fluidos e permite descrever seu comportamento no interior dos meios porosos que compõem os reservatórios de petróleo. Com isso planeja o desenvolvimento de um campo de petróleo, prevê seu desempenho durante a vida produtiva e analisa seu comportamento propondo correções para otimizar seu desempenho, com o objetivo de maximizar a produção de hidrocarbonetos com o menor custo possível.

Avaliação de Formações são as atividades e estudos que visam definir em termos qualitativos e quantitativos a capacidade produtiva e a valorização das reservas de óleo e gás de uma jazida petrolífera. O monitoramento das vazões e pressões de fundo do poço, durante um teste, e o conhecimento das propriedades dos fluidos, permite que informações a respeito das características da rocha-reservatório e parâmetros de drenagem sejam obtidos.

Os métodos matemáticos utilizados, regressão semi-logarítmica, ajuste de curvas pelo método dos mínimos quadrados, são provenientes da disciplina de cálculo numérico. Esta tem como objetivo estudar e criar algoritmos numéricos para resolução de problemas que podem ser representados por um Modelo Matemático. O objetivo do cálculo numérico é encontrar uma solução aproximada para o problema,

mantendo sobre controle os erros associados com essa aproximação.

## 3.2 Formulação teórica

Para determinar os parâmetros de reservatórios é necessário compreender alguns conceitos e formulações matemáticas relacionados às disciplinas citadas. Estes serão explicados abaixo, com a finalidade de elucidar as idéias em que serão desenvolvidas para o funcionamento do *software*.

### 3.2.1 Teste de Pressão

Um dos métodos mais utilizados na Engenharia de Reservatórios são os testes de pressão. Estes consistem em acompanhar os dados de pressão no poço em função da vazão de produção utilizada. Através desse acompanhamento pode se determinar parâmetros de drenagem e outros fatores que interferem na produção do campo de petróleo.

A avaliação da formação consiste em um conjunto de atividades com vazão controlada, que têm como objetivos:

- obter o fluido contido na formação para análise do mesmo.
- avaliar a capacidade produtiva da formação.
- investigar a existência de danos de formação e efeito de estocagem.
- determinar a extensão do reservatório e sua pressão inicial.

Os testes de pressão seguem as seguintes etapas:

- completar o poço temporariamente para permitir a produção do fluido de forma segura.
- isolar o intervalo a ser testado.
- criar um diferencial de pressão entre poço e reservatório, afim de produzir o fluido.
- promover períodos intercalados de produção e fechamento do poço.
- registro contínuo de vazões em superfície e pressões no poço.

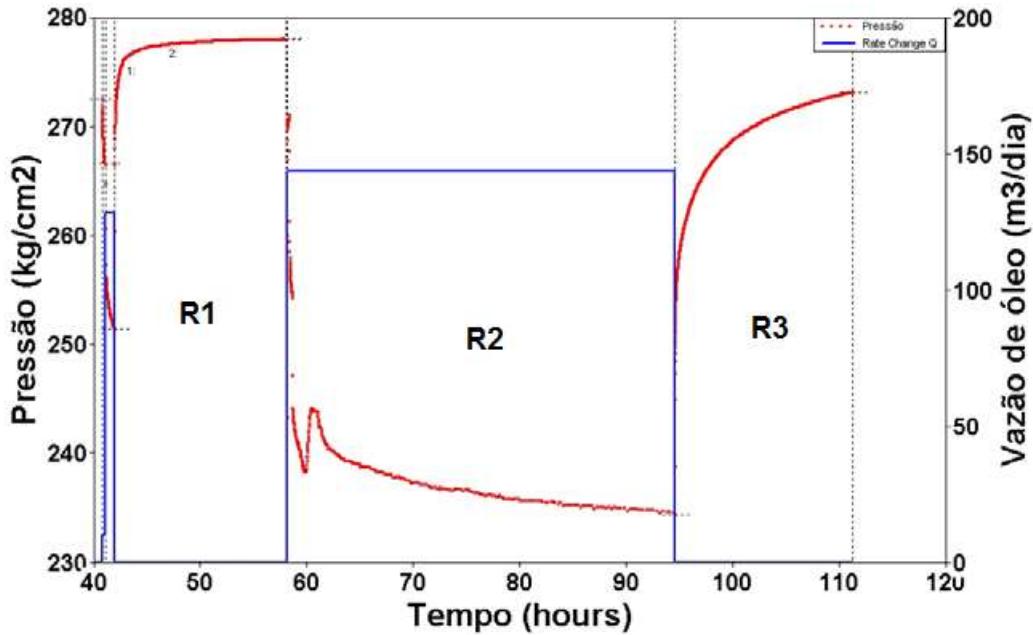


Figure 3.1: Exemplo de teste de poço convencional, onde é possível observar o comportamento da pressão em função da variação da vazão de produção. As regiões 1 e 3 mostram etapas com o poço fechado ( $q=0$ ), onde é possível ver um aumento da pressão no reservatório. Já na região 2, o poço está produzindo com uma vazão constante, causando uma queda na pressão do reservatório.

Os teste de pressão mais comuns são: testes de fluxo (*drawdown*), testes de crescimento de pressão (*buildup*) e testes de injeção e *falloff*.

O software desenvolvido tratará de um teste de crescimento de pressão, a metodologia mais difundida para avaliação de formações.

O teste de crescimento de pressão, baseia-se no registro contínuo das pressões de fundo, após o fechamento de um poço que tenha estado produzindo por um determinado período. A Figura (3.2) ilustra o esquema do teste. Os dados de pressão medidos durante o período de estática contém menos ruído, que os medidos durante o período de fluxo, devido à vazão ( $q$ ) igual a zero imposta ao poço.

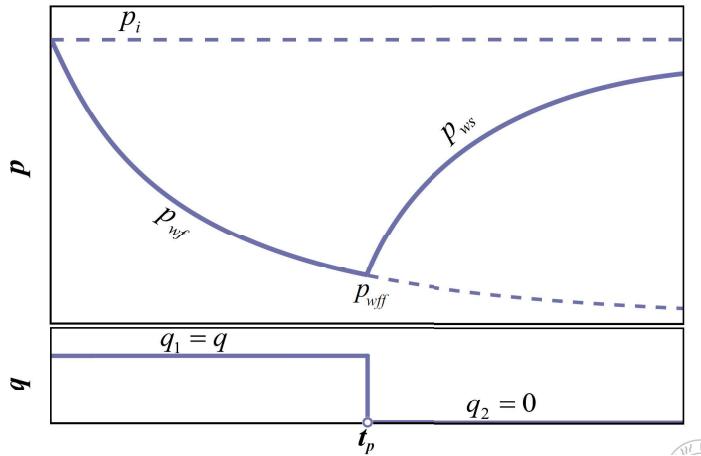


Figura 3.2: Esquema de vazão e comportamento da pressão em um poço submetido a um teste de pressão

Onde:

$p_{wf}$  = pressão no poço durante o período de fluxo.

$p_{ws}$  = pressão durante o período de fechamento.

$p_{wff}$  = última pressão de fluxo, ou seja, no instante em que o poço é fechado.

$t_p$  = tempo de produção (ou de fluxo).

$\Delta t$  = período de tempo medido a partir do momento em que o poço é fechado.

A base para toda análise e obtenção das equações que regem o comportamento do reservatório é a Equação da Difusividade 3.1, que para geometria radial e fluxo monofásico é dada por:

$$\frac{1}{r} \frac{\partial}{\partial r} \left( \frac{k(r)}{\mu} r \frac{\partial p}{\partial r} \right) + \frac{k(r)}{\mu} Cf \left( \frac{\partial p}{\partial r} \right)^2 = \phi C t \frac{\partial p}{\partial t} \quad (3.1)$$

Dentre as inúmeras soluções existentes para determinados tipos de reservatórios existentes e suas propriedades, os testes de pressão são fundamentados na utilização da solução transiente dessa equação, com uma posição fixa no fundo do poço ( $r = r_w$ ), em um reservatório infinito e homogêneo. A análise de testes de crescimento pode ser realizadas pelo método de ajustamento com curvas-tipo e por métodos convencionais e podemos mencionar o método de Horner, método de MDH, método de Agarwal, são utilizados outros derivados para estimativa da pressão média na área de influência de um poço.

Deve-se atentar ao sistema de unidades, para que os parâmetros sejam inseridos com as unidades corretamente e não haja problemas nos cálculos. A Tabela (3.1) mostra as unidades correspondentes de acordo com os sistemas existentes e que devem ser obedecidas.

Tabela 3.1: Variáveis no Sistema de Unidades

| Variáveis                | Sistema Petrobras        | <i>Oilfield</i>   |
|--------------------------|--------------------------|-------------------|
| Comprimento              | $m$                      | $ft$              |
| Compressibilidade        | $(kgf/cm^2)^{-1}$        | $psi^{-1}$        |
| Coeficiente de Estocagem | $\frac{m^3}{(kgf/cm^2)}$ | $\frac{bbl}{psi}$ |
| Tempo                    | $h$                      | $h$               |
| Permeabilidade           | $md$                     | $md$              |
| Pressão                  | $kgf/cm^2$               | $psi$             |
| Viscosidade              | $cp$                     | $cp$              |
| Vazão de Óleo            | $m^3/d$                  | $bbl/d$           |
| $C_1$                    | 0,0003484                | 0,0002637         |
| $C_2$                    | 19,03                    | 141,2             |
| $C_3$                    | $1/(2\pi)$               | 0,8936            |

### 3.2.2 Método de Horner

Com os dados obtidos da pressão e o tempo em que cada uma foi medida (o teste inteiro pode variar a duração desde algumas horas a alguns dias), gera-se um gráfico semi-logarítmico em que seu coeficiente angular possui uma relação intrínseca com a permeabilidade da formação e estima-se propriedades pelo método de Horner, ilustrado pela Figura 3.2.

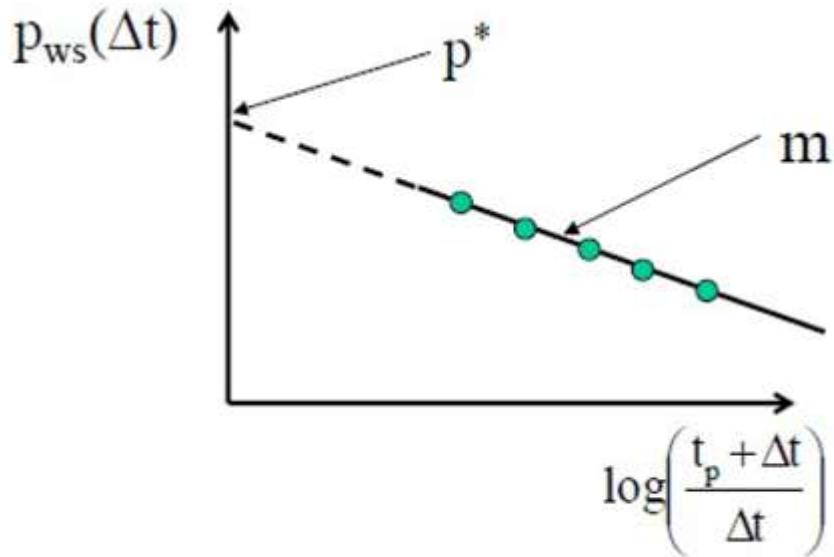


Figura 3.3: Gráfico semi-logarítmico obtido pelo Método de Horner, que fornece parâmetros referentes ao reservatório

As equações apresentadas a seguir para determinação dos parâmetros são vistas em [Adalberto Rosa, 2006].

A permeabilidade, que é definida como a capacidade de um material (tipicamente uma rocha) de transmitir fluídos, pode ser inferida pela equação 3.2. Na equação,  $q$  [barris/d]

é a vazão,  $B$  [adimensional] o fator volume-formação,  $h$  [pés] a altura da formação,  $\mu$  [cp] a viscosidade do fluido, e  $m$  [adimensional] é o coeficiente angular.

$$k = \frac{1.151\alpha_p q B \mu}{m h} \quad (3.2)$$

O fator de película  $S$  [adimensional] é definido como uma região ao redor do poço cuja permeabilidade foi alterada, reduzida (por fluido de perfuração/completação, inchamento de argilas, inversão de molhabilidade, etc) ou melhorada (através de processos de acidificação, fraturamento hidráulico, etc.) e se dá pela equação 3.3. É uma equação adimensional, onde  $ct$  [1/psi] é a compressibilidade total,  $rw$  [pés] é o raio do poço,  $tp$  [dias] o tempo de produção do poço,  $Pwf$  [psi] é a pressão registrada no fechamento do poço,  $\Delta P_{skin}$  [psi] é a queda de pressão devido ao dano Eq.3.4 e  $n$  [adimensional] é o coeficiente linear do gráfico. Quanto maior o valor do fator de película, maior o dano, resultando em menor produção e maior queda de pressão.

$$S = \frac{1.151(m \cdot \log(tp + 1) + n - Pwf)}{(-m - \log(k/(ctr_w^2)) + 3.23)} \quad (3.3)$$

$$\Delta P_{skin} = 0.869.(-m).s \quad (3.4)$$

A capacidade produtiva de um poço é caracterizada pelo índice de produtividade  $IP$  [barris/(dias.psi)], que indica a necessidade de injeção de fluidos para aumento da recuperação. A eficiência de fluxo  $EF$  [adimensional] indica o quanto a produção está sendo afetada pelo fator de película do poço. Esses fatores, dados em porcentagem, são importantes para a engenharia de reservatórios, pois definem a viabilidade de produção. São definidos pela equações 3.5 e 3.6, sendo  $Pi$  [psi] a pressão inicial do reservatório, indicada pelo gráfico. Valores maiores que 10 para o índice de produtividade são considerados bons.

$$IP = \frac{q}{Pi - Pwf} \quad (3.5)$$

$$EF = \frac{Pi - Pwf - \Delta P_{skin}}{Pi - Pwf} \quad (3.6)$$

O raio efetivo do poço  $r'_w$  [pés] é definido como o tamanho teórico do poço incluindo o dano, calculado pela equação 3.7, sua unidade é [m]. Quanto maior o dano, menor o raio efetivo, pois o poço produz menos do que deveria. O raio efetivo de um poço estimulado ( $s < 0$ ) é maior que o raio real, enquanto de um poço danificado ( $s > 0$ ) é menor que o raio real.

$$r'_w = r_w e^{-s}. \quad (3.7)$$

O efeito de estocagem ocorre nos primeiros momentos da produção, fazendo com que a vazão do poço não seja igual à do reservatório, havendo uma estocagem de fluidos no interior do poço pela expansão e compressão do volume dos hidrocarbonetos. O coeficiente de estocagem  $C$  [barris/psi] é descrito pela equação 3.8, e sua duração,  $t_{wbs}$ [horas], pela equação 3.9.

$$C = \frac{qB\Delta t}{24(P - P_{wf})} \quad (3.8)$$

$$t_{wbs} = \frac{60.0 + 3.5 * S}{(24C * \alpha p k h \mu)} \quad (3.9)$$

### 3.2.3 Método de MDH

O método de MDH (Miller, Dyes & Hutchinson, 1950) fornece meios de se estimar a permeabilidade, o fator de película e, ainda, a pressão média na região drenada pelo poço, através da análise de dados de crescimento de pressão.

A equação da pressão no poço, durante o período de crescimento de pressão, com a aplicação do princípio da superposição é mostrada na Equação :

$$\frac{kh}{C_2 q B \mu} (p_i - p_{ws}) = p_{wD} [(t_p + \Delta t)_D] - p_{wD} (\Delta t_D) \quad (3.10)$$

Se o período de produção é muito maior em relação ao tempo de fechamento, tem-se que:

$$p_{wD} [(t_p + \Delta t)_D] \cong p_{wD} (t_{pD}). \quad (3.11)$$

Assim, para  $t_p \gg \Delta t$  a Eq.(3.11), fica simplificada:

$$\frac{kh}{C_2 q B \mu} (p_i - p_{ws}) = p_{wD} (t_{pD}) - p_{wD} (\Delta t_D). \quad (3.12)$$

Para representar o termo de  $p_{wD} (\Delta t_D)$ , emprega-se a aproximação logarítmica, ou seja, é admitido comportamento de reservatório infinito após o fechamento:

$$p_{wD} (\Delta t_D) = \frac{1}{2} \ln \left( \frac{4\Delta t_D}{e^\gamma} \right) + s. \quad (3.13)$$

A Eq.(3.12) se transforma em:

$$\frac{kh}{C_2 q B \mu} (p_i - p_{ws}) = p_{wD} (t_{pD}) - \frac{1}{2} \ln \left( \frac{4\Delta t_D}{e^\gamma} \right) + s \quad (3.14)$$

que pode também ser reescrita substituindo no termo logarítmico a definição do tempo adimensional, usando o logaritmo na base 10 e rearranjando os termos, obtendo:

$$p_{ws} = p_i - 1,151 \frac{C_2 q B \mu}{kh} \left[ 0,8686 p_{wD}(t_{pD}) - \log \left( \frac{4C_1 k}{e^{\gamma} \phi \mu c_t r_w^2} \right) - 0,8686 s \right] + 1,151 \frac{C_2 q B \mu}{kh} \log \Delta t. \quad (3.15)$$

Definindo  $p_1$  a pressão durante o fechamento correspondente a um tempo de fechamento unitário,  $\Delta t = 1$ , então:

$$p_1 = p_i - 1,151 \frac{C_2 q B \mu}{kh} \left[ 0,8686 p_{wD}(t_{pD}) - \log \left( \frac{4C_1 k}{e^{\gamma} \phi \mu c_t r_w^2} \right) - 0,8686 s \right]. \quad (3.16)$$

A Eq. (3.15), pode ser compactada como:

$$p_{ws} = p_1 + m \log \Delta t \quad (3.17)$$

em que:

$$m = 1,151 \frac{C_2 q B \mu}{kh} \quad (3.18)$$

Os parâmetros do sistema poço-reservatório são determinados pelo ajuste dos dados a uma linha reta, no gráfico  $p_{ws}$  versus  $\log \Delta t$ , como mostrado na Figura (Com o coeficiente angular  $m$ , pode ser calculada a permeabilidade:

$$k = 1,151 \frac{C_2 q B \mu}{mh}. \quad (3.19)$$

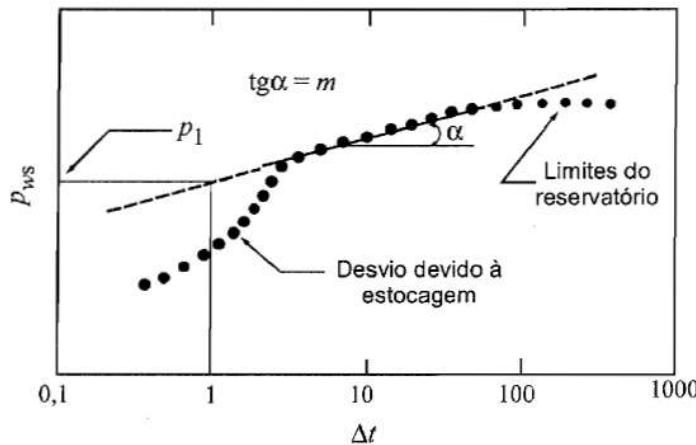


Figura 3.4: Gráfico de MDH

Para obter o fator de película, usa-se a queda de pressão no instante do fechamento, admitindo que o reservatório se comporte como infinito.

$$p_{wsD} \equiv \frac{kh}{C_2 q B \mu} (p_i - p_{wff}) = \frac{1}{2} \ln \left( \frac{4t_{pD}}{e^\gamma} \right) + s \quad (3.20)$$

onde:

$p_{wff}$  = última pressão de fluxo antes do fechamento.

Durante o fechamento, o comportamento da pressão pode ser escrito como:

$$\frac{kh}{C_2 q B \mu} (p_i - p_{ws}) = \frac{1}{2} \ln \left( \frac{4t_{pD}}{e^\gamma} \right) - \frac{1}{2} \ln \left( \frac{4C_1 \eta}{e^\gamma r_w^2} \right) - \frac{1}{2} \ln (\Delta t). \quad (3.21)$$

Onde  $\eta = k/\phi \mu C_t$ , é a constante de difusividade hidráulica. Fazendo então  $\Delta t = 1$ , a equação anterior é reescrita:

$$\frac{kh}{C_2 q B \mu} (p_i - p_1) = \frac{1}{2} \ln \left( \frac{4t_{pD}}{e^\gamma} \right) - \frac{1}{2} \ln \left( \frac{4C_1 \eta}{e^\gamma r_w^2} \right). \quad (3.22)$$

Subtraindo a Eq.(3.20) da Eq.(3.22), transformando para o logaritmo na base 10, utilizando a definição de  $m$ , explicita-se o fator de película:

$$s = 1,151 \left[ \frac{p_1 - p_{wff}}{m} - \log \left( \frac{C_1 \eta}{r_w^2} \right) - 0,3514 \right]. \quad (3.23)$$

Mais informações sobre métodos buildup, podem ser encontradas no *Well Testing*, J.Lee , *Oil Well Testing Handbook*, apostila Análise de Testes de Pressão, e notas de aula de avaliação de formações.

### 3.3 Identificação de pacotes – assuntos

Após explicitar os conceitos, serão desenvolvidos pelo *software* os seguintes conjuntos de assuntos (ou pacotes):

- Pacote TestePressão: apresenta os conceitos do teste de *build-up*. Importa os dados do teste de pressão de um arquivo .dat que serão processados para estimar os parâmetros do reservatório.
- Pacote Estatística: realiza a regressão linear dos dados passados no arquivo .dat. Deve apresentar conexão com o Pacote Reservatório, uma vez que recebe os dados contidos nele.
- Pacote Gráfico: gera os gráficos com a curva semi-logarítmica dos dados da regressão linear feita no Pacote Matemático. Também plota os dados de entrada da pressão versus tempo.
- Pacote Reservatório: é o pacote que contém os dados do sistema poço-reservatório.
- Pacote Buildup: contém os conceitos matemáticos de buildup e os métodos que podem ser utilizados para os cálculos das propriedades, MDH e Horner.
- Pacote Simulador: composto de diversas equações, calcula propriedades que podem ser obtidas pela curva semi-logarítmica. Os resultados dessas equações são parte da saída do programa.
- Pacote Resultados: Os parâmetros calculados no Pacote Propriedades são fornecidos ao usuário através de um arquivo .dat e armazenados em um banco de dados. O mesmo é feito com as curvas de pressão x tempo, que são fornecidas ao usuário através de um arquivo de imagem e posteriormente armazenadas.

### 3.4 Diagrama de pacotes – assuntos

A Figura 3.5 representa o diagrama de pacotes para o simulador de cálculos de parâmetros de reservatório.

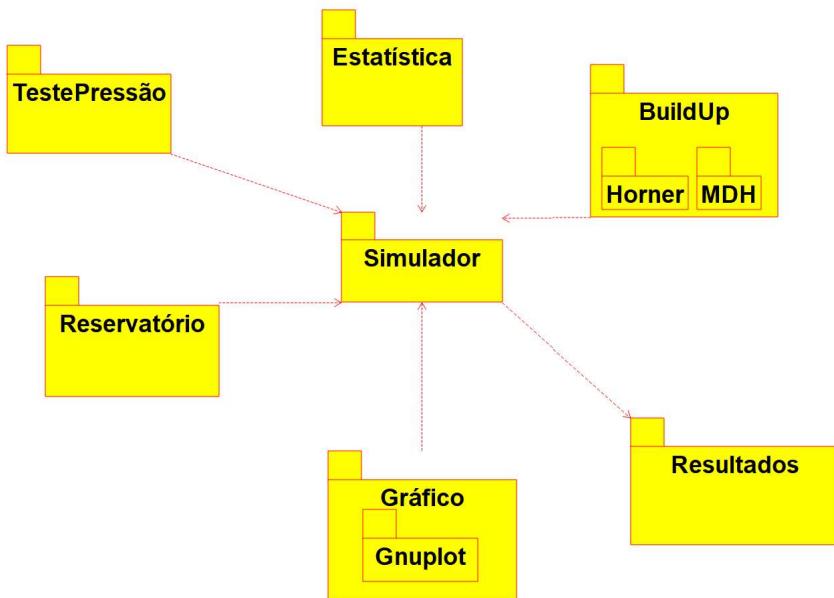


Figura 3.5: Diagrama de Pacotes

# Capítulo 4

## AOO – Análise Orientada a Objeto

A terceira etapa do desenvolvimento de um software é a AOO – Análise Orientada a Objeto. A AOO 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.

### 4.1 Diagramas de classes

O diagrama de classes é apresentado na Figura 4.1.

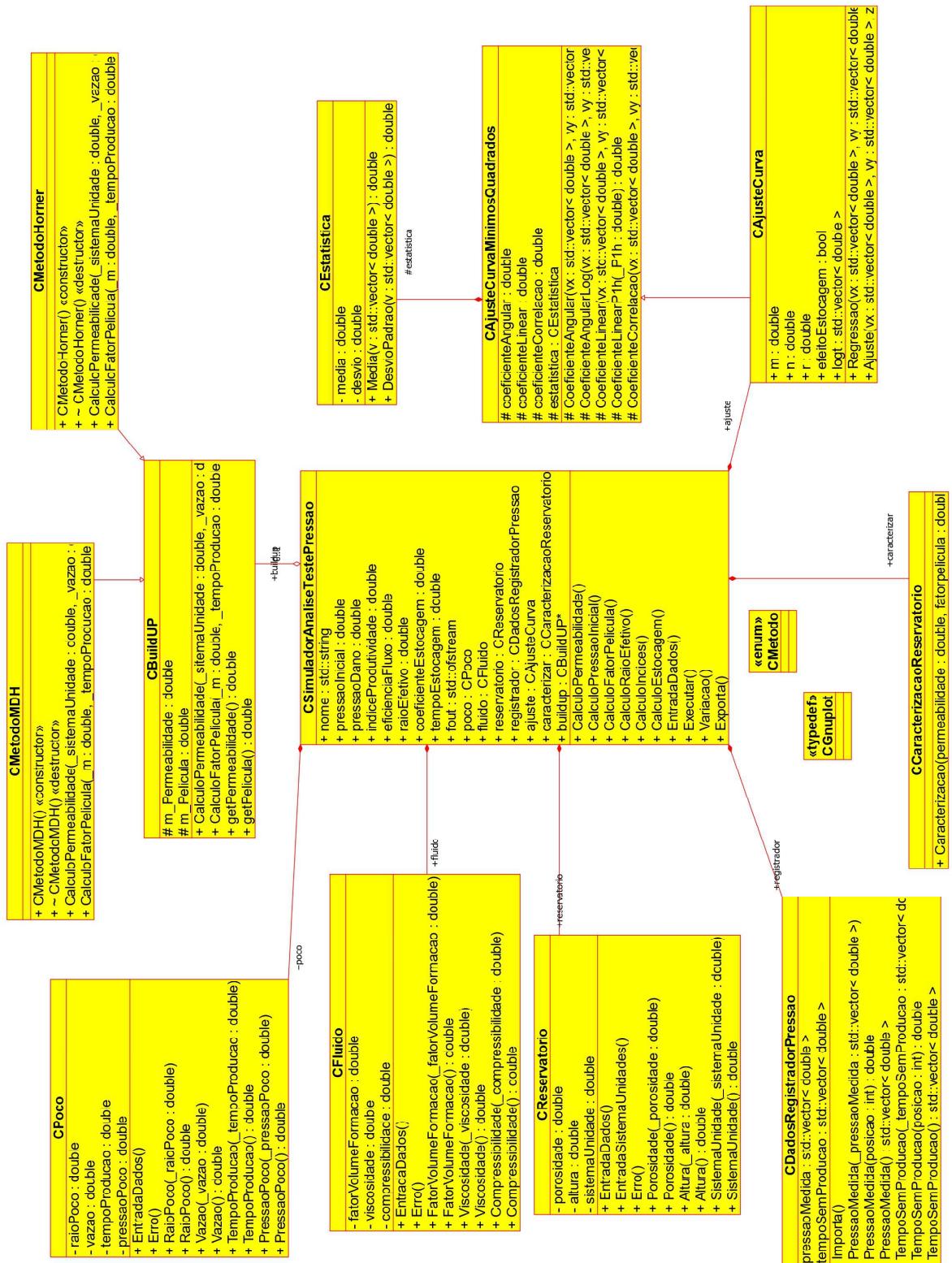


Figura 4.1: Diagrama de classes

#### 4.1.1 Dicionário de classes

- Classe CPoco: Classe que possui as características/atributos do poço, e tem uma função de entrada de dados por parte do usuário.
  - atributo vazao.
  - atributo tempoProducao.
  - atributo pressaoPoco.
  - atributo raioPoco.
  - método EntradaDados ( ): Método que pede ao usuário os parâmetros necessários para o programa.
  - método Erro ( ): Verifica e retorna uma mensagem de erro, caso haja alguma entrada equivocada do usuário.
  - método Vazao (\_vazao): Método que seta o valor do atributo vazao.
  - método Vazao ( ): Método que retorna o valor do atributo vazao.
  - método TempoProducao (\_tempoProducao): Método que seta o valor do atributo tempoProducao.
  - método TempoProducao ( ): Método que retorna o valor do atributo tempoProducao.
  - método PressaoPoco (\_pressaoPoco): Método que seta o valor do atributo pressaoPoco.
  - método PressaoPoco ( ): Método que retorna o valor do atributo pressaoPoco.
  - método RaioPoco (\_raioPoco): Método que seta o valor do atributo raioPoco.
  - método RaioPoco ( ): Método que retorna o valor do atributo raioPoco.
- Classe CReservatorio: Classe que possui as características/atributos do reservatório, e tem uma função de entrada de dados por parte do usuário.
  - atributo porosidade.
  - atributo altura.
  - atributo sistemaUnidade.
  - método EntradaDados ( ): Método que pede ao usuário os parâmetros necessários para o programa.
  - método SistemaUnidades ( ): Método do tipo void que pergunta ao usuário o sistema de unidades utilizado para os parâmetros fornecidos, através de um pequeno menu.

- método SistemaUnidade (\_sistemaUnidade): Método que seta o valor do atributo sistemaUnidade.
  - método SistemaUnidade ( ): Método que retorna o valor do atributo sistemaUnidade.
  - método Erro ( ): Verifica e retorna uma mensagem de erro, caso haja alguma entrada equivocada do usuário.
  - método Porosidade (\_porosidade): Método que seta o valor do atributo porosidade.
  - método Porosidade ( ): Método que retorna o valor do atributo porosidade.
  - método Altura (\_altura): Método que seta o valor do atributo altura.
  - método Altura ( ): Método que retorna o valor do atributo altura.
- Classe CFluido: Classe que possui as características/atributos do fluido, e tem uma função de entrada de dados por parte do usuário.
    - atributo fatorVolumeFormacao.
    - atributo viscosidade.
    - atributo compressibilidade.
    - método EntradaDados ( ): Método do tipo void que pede ao usuário os parâmetros necessários para o programa.
    - método Erro ( ): Verifica e retorna uma mensagem de erro, caso haja alguma entrada equivocada do usuário.
    - método FatorVolumeFormacao (\_fatorVolumeFormacao): Método que seta o valor do atributo fatorVolumeFormacao.
    - método FatorVolumeFormacao ( ): Método que retorna o valor do atributo fatorVolumeFormacao.
    - método Viscosidade (\_viscosidade): Método que seta o valor do atributo viscosidade.
    - método Viscosidade ( ): Método que retorna o valor do atributo viscosidade.
    - método Compressibilidade (\_compressibilidade): Método que seta o valor do atributo compressibilidade.
    - método Compressibilidade ( ): Método que retorna o valor do atributo compressibilidade.
  - Classe CDadosRegistradorPressao: Classe que cria 2 vetores e os preenche com os dados de teste de pressão importados de um arquivo de disco.

- atributo pressaoMedida.
  - atributo tempoSemProducao.
  - método Importa ( ): Método do tipo void que preenche os vetores.
  - método PressaoMedida (\_pressaoMedida): Método que seta o valor do atributo pressaoMedida.
  - método PressaoMedida (\_posicao): Método que seta o valor do atributo pressaoMedida na posicao desejada.
  - método PressaoMedida ( ): Método que retorna o valor do atributo pressaoMedida.
  - método TempoSemProducao (\_tempoSemProducao): Método que seta o valor do atributo tempoSemProducao.
  - método TempoSemProducao (\_posicao): Método que seta o valor do atributo tempoSemProducao na posição desejada.
  - método TempoSemProducao ( ): Método que retorna o valor do atributo tempoSemProducao.
- Classe CEstatistica: Classe que faz a média e desvio padrão de vetores, necessários para a regressão linear dos dados.
    - atributo media.
    - atributo desvio.
    - método Media (v): Retorna a média do vetor v.
    - método DesvioPadrao (v): Retorna o desvio padrão de v.
  - Classe CAjusteCurvaMinimosQuadrados: Classe que faz a regressão linear através do método dos mínimos quadrados.
    - atributo coeficienteAngular.
    - atributo coeficienteLinear.
    - atributo coeficienteCorrelacao.
    - atributo estatistica.
    - método CoeficienteAngular (vx,vy): Retorna o coeficiente angular da reta obtida da regressão dos vetores vx e vy.
    - método CoeficienteLinear (vx,vy): Retorna o coeficiente linear da reta obtida da regressão dos vetores vx e vy.
    - método CoeficienteCorrelacao (vx,vy): Retorna o coeficiente correlação da reta obtida da regressão dos vetores vx e vy.

- método CoeficienteAngularLog (vx,vy): Retorna o coeficiente angular logarítmico da reta obtida na regressão dos vetores vx e vy.
  - método CoeficienteLinearP1h (\_P1h) : Método que seta o valor do atributo coeficienteLinear na pressão P1h.
- Classe CMetodo: classe enumerativa que representa diferentes métodos. Essa enumeração tem três valores: none, METODO\_HORNER e METODO\_MDH. Eles são utilizados para identificar diferentes métodos.
  - Classe CBuildup: é uma classe abstrata, e qualquer classe que a derive deve implementar esses métodos.
    - atributo Permeabilidade
    - atributo Película
    - método CalculoPermeabilidade (): Função que calcula exibe a permeabilidade do reservatório.
    - método CalculoFatorPelícula (): Função que calcula e exibe o fator de película do reservatório.método CalculoPermeabilidade
    - método getPermeabilidade (): Método de acesso para o atributo permeabilidade em uma classe.
    - método getPelícula(): Método de acesso para o atributo fator de película em uma classe.
  - Classe CMetodoHorner: é a classe responsável por aplicar o método de Horner com os dados obtidos no software.
    - método CMetodoHorner (): Método construtor da classe.
    - método ~CMetodoHorner (): Método destrutor da classe.
    - método CalculoPermeabilidade (): Função que calcula exibe a permeabilidade do reservatório por esse método.
    - método CalculoFatorPelícula (): Função que calcula e exibe o fator de película do reservatório por esse método.
  - Classe CMetodoMDH: é a classe responsável por aplicar o método de MDH com os dados obtidos no software.
    - método CMetodoMDH (): Método construtor da classe.
    - método ~CMetodoMDH (): Método destrutor da classe.
    - método CalculoPermeabilidade (): Função que calcula exibe a permeabilidade do reservatório por esse método.

- método CalculoFatorPelicula (): Função que calcula e exibe o fator de película do reservatório por esse método.
- Classe CAjusteCurva: Classe que executa a regressão linear (de uma reta semilogarítmica) dos dados obtidos e verifica se o coeficiente de correlação é satisfatório, caso não seja, descobre-se a melhor aproximação (o ponto) onde começa a reta da curva (a curva sendo o efeito de estocagem).
  - atributo m: Representa o coeficiente angular da reta obtida na regressão linear.
  - atributo n: Representa o coeficiente linear da reta obtida na regressão linear.
  - atributo r: Coeficiente de correlação da reta, quanto mais próximo de 1, melhor a regressão linear.
  - atributo logt: Vetor que relaciona as variáveis tp e o vetor deltat.
  - atributo efeitoEstocagem
  - método Regressao (vx, vy, z): Função que executa a regressão linear propriamente dita dos vetores, calculando os valores de m, n e r.
  - método Ajuste (vx, vy, z): Função que analisa se a regressão linear tem um fator de correlação de Pearson suficiente para o programa gerar resultados confiáveis.
- Classe CSimuladorAnaliseTestePressao: Classe principal, que se comunica com os objetos das outras classes para inferir parâmetros do reservatório e calcular outras variáveis a partir de equações de correlação.
  - atributo permeabilidade: Permeabilidade, obtido a partir da regressão linear.
  - atributo pressaoInicial: Pressão inicial, obtido a partir da regressão linear.
  - atributo fatorPelicula.: Fator de película, obtido a partir da correlação.
  - atributo pressaoDano: Queda de pressão devido ao dano, obtido a partir de correlação.
  - atributo indiceProdutividade: Índice de produtividade, obtido a partir de correlação.
  - atributo eficienciaFluxo: Eficiência de Fluxo, obtido a partir de correlação
  - atributo raioEfetivo: Raio efetivo do poço, obtido a partir de correlação.
  - atributo coeficienteEstocagem: Coeficiente de Estocagem.
  - atributo tempoEstocagem: Período que dura o efeito de estocagem.
  - atributo caracterizar: Caracterizar o reservatório.
  - atributo buildup : ponteiro para a classe CBuildup.

- método EntradaDados ( ): Método que chama as funções de entrada das classes CFluido, CPoco e CReservatorio.
  - método CalculoPermeabilidade ( ): Função que calcula e exibe a permeabilidade do reservatório.
  - método CalculoPressaoInicial ( ): Função que calcula e exibe a pressão inicial pela extração da reta.
  - método CalculoFatorPelicula ( ): Função que calcula e exibe o fator de película do reservatório e a queda de pressão devido à esse fator.
  - método CalculoIndices ( ): Função que calcula e exibe o índice de produtividade do reservatório e a eficiência de fluxo.
  - método CalculoRaioEfetivo ( ): Função que calcula e exibe o raio efetivo.
  - método Exporta(): Método que exporta os resultados para um arquivo .dat com um nome escolhido pelo usuário.
  - método Variacao(): Método que permite ao usuário variar um parâmetro selecionado e visualizar a mudança no comportamento do reservatório nos gráficos gerados.
- Classe CCaracterizacaoReservatorio: Classe que caracteriza o reservatório, interpretando os resultados obtidos.
    - método Caracterizacao (permeabilidade, fatorPelicula, indiceProdutividade, raioPoco, raioEfetivo): Função do tipo void que analisa os resultados e informa ao usuário a qualidade do reservatório submetido ao teste de pressão.
    - método Caracterização
  - O programa externo Gnuplot é responsável pela geração dos gráficos e exportação de imagens.

## 4.2 Diagrama de sequência – eventos e mensagens

### 4.2.1 Diagrama de sequência geral

Veja o diagrama de seqüência na Figura 4.2.

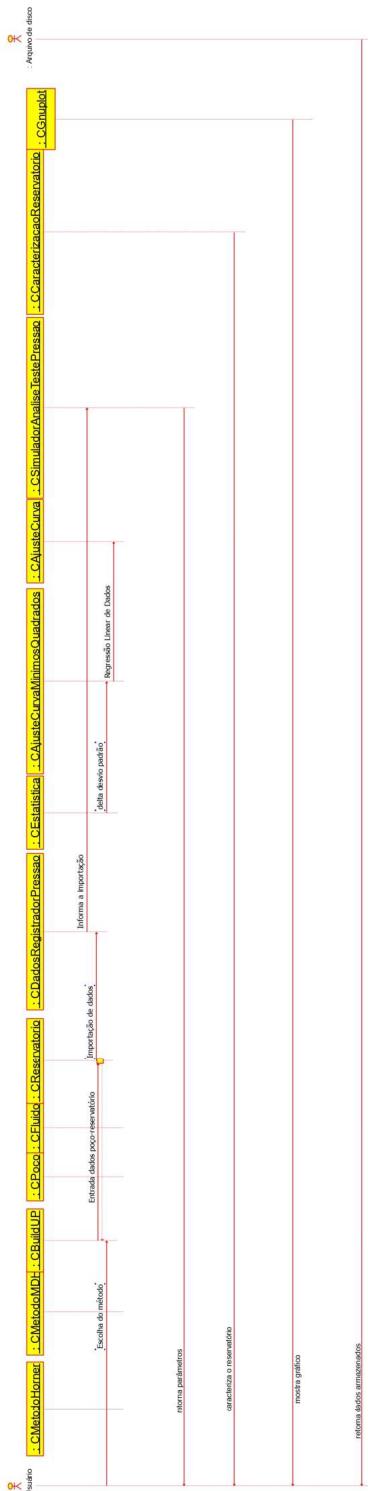


Figura 4.2: Diagrama de sequência

### 4.3 Diagrama de comunicação

A Figura 4.3 apresenta o diagrama de comunicação. Observe que há muita interação entre os objetos de cada classe, iniciando pela entrada de dados para preencher os

objetos de CPoco, CReservatorio e CFluido. Caso não ocorra erro na entrada, a classe CDadosRegistradorPressão importa os dados do arquivo de texto e informa para a classes de ajuste. Esta, por sua vez, utiliza a classe CAjuste para gerar a reta da regressão e CAjusteCurvaMimosQuadrados que faz a regressão de dois vetores usando a média e o desvio padrão, obtidos da classe CEstatistica. Após a função Ajuste (r), que encontra o coeficiente de estocagem, o Simulador faz os cálculos dos parâmetros do reservatório com esses dados de entrada e de importação. A classe CCaracterizaoReservatorio caracteriza o reservatorio com a função Resultados (permeabilidade, fatorPelícula)Além disso, a classe CGnuplot permite gerar os gráficos da regressão e salvá-los.

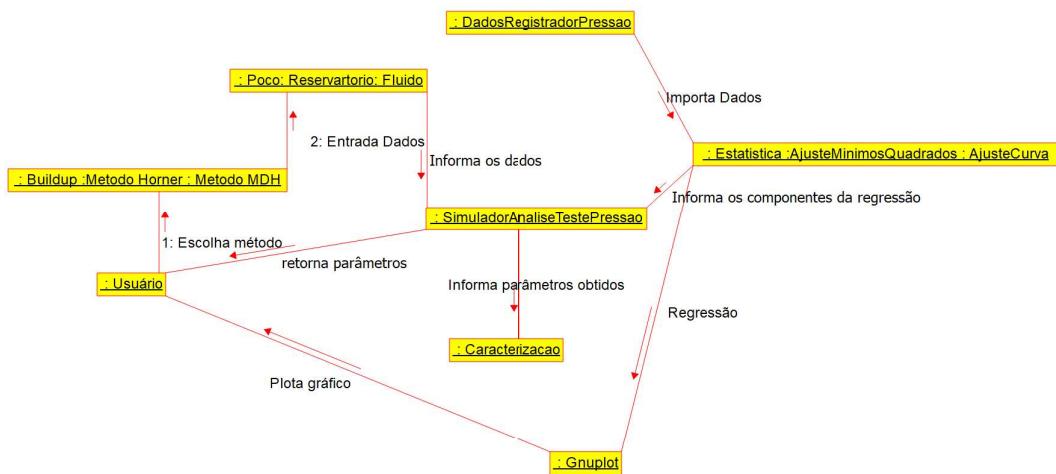


Figura 4.3: Diagrama de comunicação

## 4.4 Diagrama de máquina de estado

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

Veja na Figura 4.4 o diagrama de máquina de estado para um objeto da classe CCaracterizacãoReservatorio.

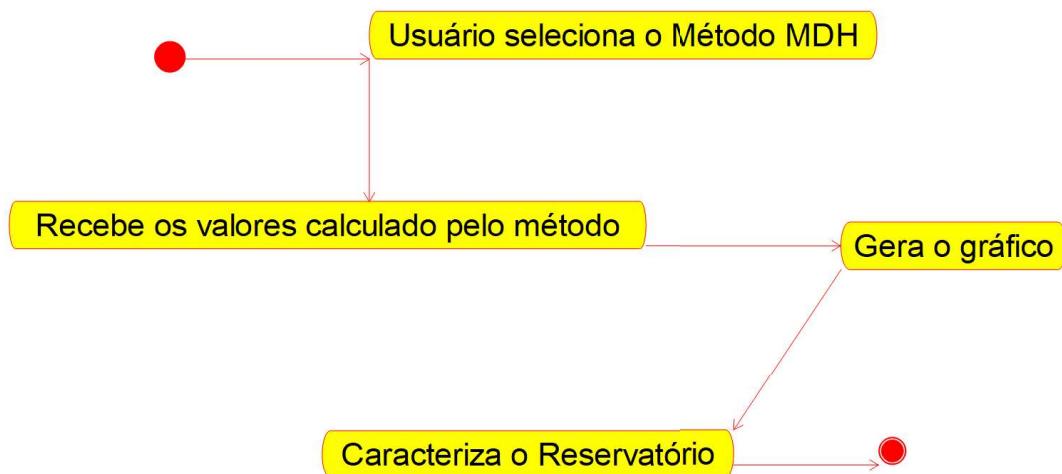


Figura 4.4: Diagrama de máquina de estado.

## 4.5 Diagrama de atividades

Veja na Figura 4.5 o diagrama de atividades correspondente a uma atividade específica do diagrama da máquina de estado. Os dados de entrada são fornecidos e importados. Não havendo erros, o programa prossegue com os dados para os cálculos pelo método de MDH e de Horner. Posteriormente os parâmetros calculados são utilizados para a caracterização do reservatório.

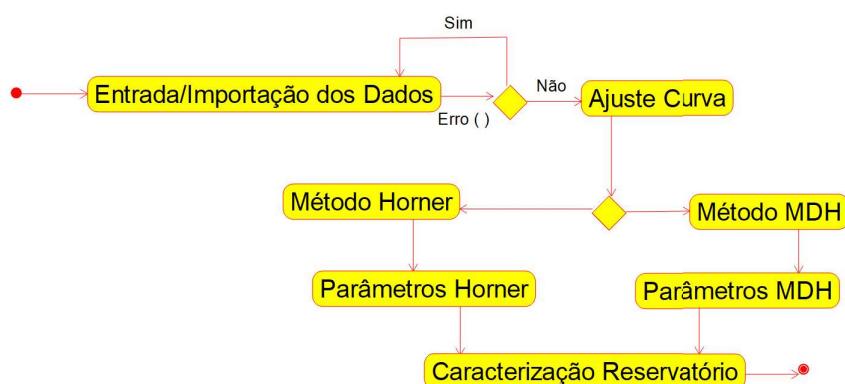


Figura 4.5: Diagrama de atividades da classe CCaracterizacaoReservatorio.

# Capítulo 5

## Projeto

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

### 5.1 Projeto do sistema

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

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

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

#### 1. Protocolos

- Neste projeto o software irá se comunicar com o componente externo Gnuplot, que plotará os gráficos, permitindo salvar imagens em disco.
- A entrada de dados será efetuada via arquivo de texto e através do teclado.
- Será utilizada a biblioteca padrão da linguagem C++.
- Definição do formato dos arquivos gerados pelo programa. Por exemplo: prefira formatos abertos, como arquivos txt e xml.
  - neste projeto o programa terá como entrada arquivos de extensão .dat. Serão gerados arquivos com extensão .dat e .jpeg.

## 2. Recursos

- O programa utiliza o HD, o processador, o teclado, a memória, a tela e os demais componentes internos do computador.
- Identificação da necessidade do uso de banco de dados. Implicam em modificações nos diagramas de atividades e de componentes.
  - Neste projeto não há necessidade da criação de um banco de dados. Os arquivos gerados serão salvo em um diretório pré-determinado pelo usuário.
- Será utilizado um arquivo de dados no formato .txt para leitura das informações do teste de pressão.

## 3. Controle

- Neste projeto o controle será sequencial.
- Neste projeto não há necessidade da criação de um banco de dados. Os arquivos gerados serão salvo em um diretório pré-determinado pelo usuário.
- Identificação da necessidade de otimização. Por exemplo: prefira sistemas com grande capacidade de memória; prefira vários hds pequenos a um grande.
  - Neste projeto não ha necessidade de uso de processos de otimização. Os cálculos realizados requerem pouco espaço na memória, tanto física, quanto de processamento.

## 4. Plataformas

- O programa usará a linguagem C++, portanto este será multiplataforma possuindo suporte em diversos sistemas operacionais.
- O software irá operar nos sistemas operacionais Windows e GNU/Linux, sendo desenvolvido e testado em ambos os sistemas.
- O software utilizará a biblioteca externa CGnuplot, permitindo o acesso ao programa Gnuplot, para gerar gráficos.
- O ambiente para montar a interface de desenvolvimento será o software Dev c++ (Windows). O compilador será o gcc/g++.

## 5.2 Projeto orientado a objeto – POO

O projeto orientado a objeto é a etapa posterior ao projeto do sistema. Baseia-se na análise, mas considera as decisões do projeto do sistema. Acrescenta a análise desenvolvida e as características da plataforma escolhida (hardware, sistema operacional e

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

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

### Efeitos do projeto no modelo estrutural

- Adicionar nos diagramas de pacotes as bibliotecas e subsistemas selecionados no projeto do sistema.
  - Neste projeto foi utilizada como biblioteca gráfica CGnuplot.
- Estabelecer as dependências e restrições associadas à plataforma escolhida.
  - O software pode ser executado nas plataformas Windows e GNU/Linux.
  - O programa depende da instalação do software externo Gnuplot, para o funcionamento do programa.

### Efeitos do projeto no modelo dinâmico

- A elaboração do programa foi focada a orientação de objetos e os diagramas foram modificados durante o desenvolvimento do código, com isso não houveram mudanças nessa etapa.

### Efeitos do projeto nos atributos

- Atributos para à leitura de dados do disco foram implementados.

### Efeitos do projeto nos métodos

- O uso da função de salvar os gráficos gerados em formato .png foi permitida pela classe externa Gnuplot.
- A inserção de dados pelo usuário pelo teclado foi implementado, além da leitura de disco.

### Efeitos do projeto nas heranças

- A classe CBuildup é herdada pelos métodos.

### Efeitos do projeto nas associações

- Houve a utilização de ponteiros, para apontar o método que deverá ser utilizado.

Depois de revisados os diagramas da análise você pode montar dois diagramas relacionados à infraestrutura do sistema. As dependências dos arquivos e bibliotecas podem ser descritos pelo diagrama de componentes, e as relações e dependências entre o sistema e o hardware podem ser ilustradas com o diagrama de implantação.

### 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.

Veja na Figura 5.1 o diagrama de componentes. A geração dos objetos depende dos arquivos de classe de extensão .h e .cpp. O simulador acessa as bibliotecas C++. O subsistema biblioteca respesta o simulador e contém os demais arquivos das classes presentes. O subsistema biblioteca Gnuplot, um subsistema externo, inclui os arquivos de código da biblioteca CGnuplot e a biblioteca em si .O subsistema banco de dados representa o arquivo que o programa importará os dados a serem manipulados. O software executável a ser gerado depende da biblioteca gerada, dos arquivos da biblioteca CGnuplot, do banco de dados.

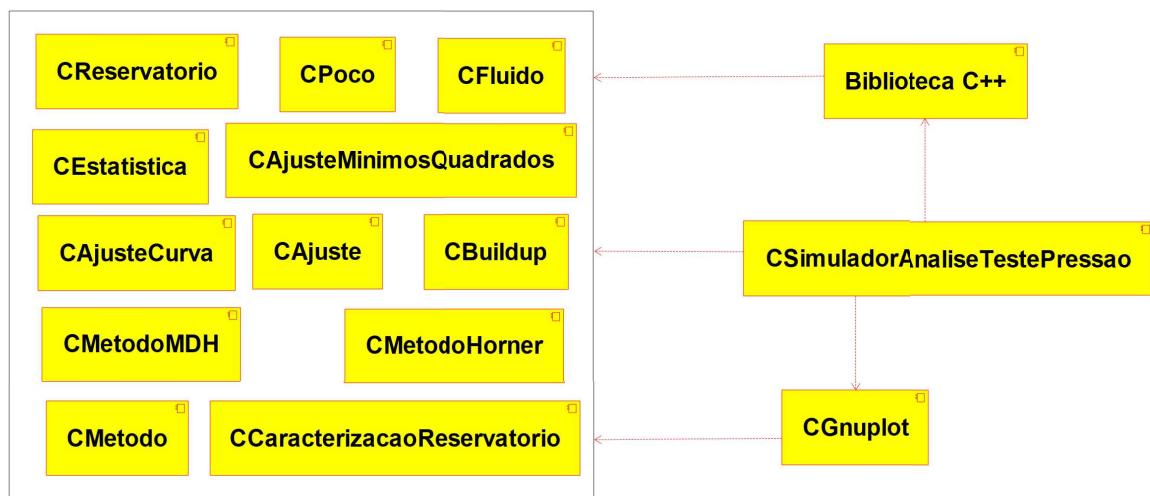


Figura 5.1: Diagrama de componentes

### 5.4 Diagrama de implantação

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

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

Veja na Figura 5.2 o diagrama de implantação do programa. Primeiramente, o simulador acessa os arquivos de dados no disco rígido, contendo as informações registradas pelo teste de pressão. O programa importa os arquivos e utiliza o teclado e o monitor para dispor os insumos e resultados ao usuário, possibilitando a comunicação. Os resultados e gráficos são armazenados no disco rígido.

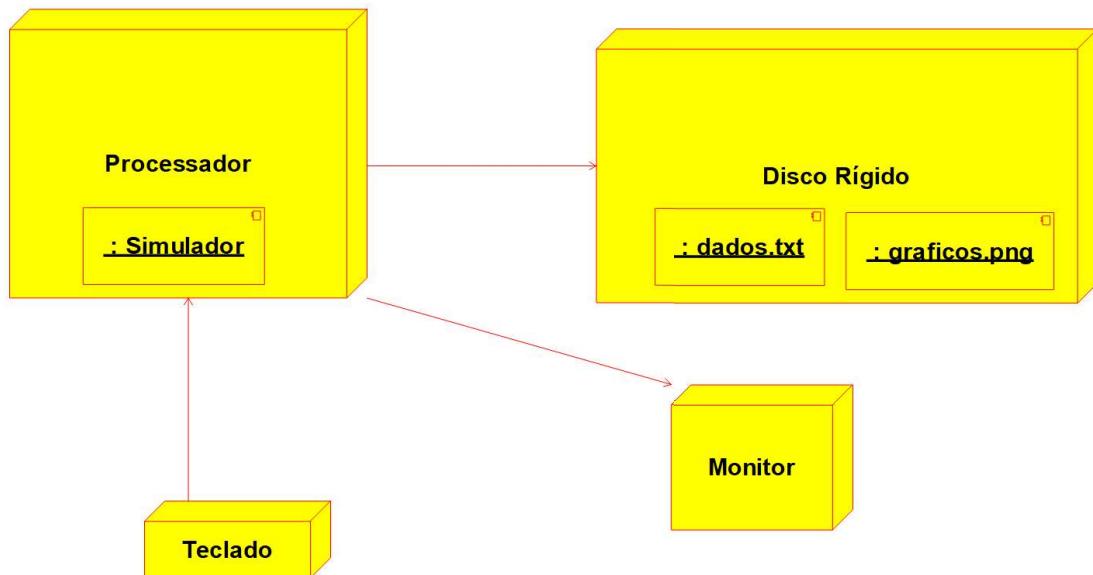


Figura 5.2: Diagrama de implantação

# Capítulo 6

## Ciclos Construção - Implementação

Neste capítulo está listado o código fonte do programa propriamente dito.

### 6.1 Código fonte

Apresenta-se a seguir um conjunto de classes (arquivos .h e .cpp) além do programa main.

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

Listagem 6.1: Arquivo de cabeçalho da classe CPoco.h

```
//Condicao para nao definir a classe mais de uma vez
#ifndef CPoco_h
#define CPoco_h

class CPoco;

//Classe contendo as caracteristicas do poco
class CPoco
{
    ///privados, so acessados por meio de funcoes get
    private:

        ///raio do poco
    double raioPoco;
        ///vazao de producao
    double vazao;
        ///tempo de producao
    double tempoProducao;
```

```

    /// pressao no poto
    double pressaoPoco;

public:

    /// Funcao que recebe dados do usuario, preenchendo atributos raio
    void EntradaDados();

    /// Funcao que verifica se houve erro na entrada de dados e pede n
    void Erro();

    /// Funcao que seta o raiopoco
    void RaioPoco(double _raioPoco);
    /// Funcao get do raiopoco
    double RaioPoco() const;

    /// Funcao que seta a vazao
    void Vazao(double _vazao);
    /// Funcao get da vazao
    double Vazao() const;

    /// Funcao que seta o tempoproducao
    void TempoProducao(double _tempoProducao);
    /// Funcao get do tempoproducao
    double TempoProducao() const;

    /// Funcao que seta a pressaopoco
    void PressaoPoco(double _pressaoPoco);
    /// Funcao get da pressaopoco
    double PressaoPoco() const;

};

#endif

```

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

Listagem 6.2: Arquivo de implementação da classe CPoco.cpp.

```
#include "CPoco.h"
```

```
// inclui a biblioteca iostrem pois usa funcoes de entrada e saida de dados
#include <iostream>
```

```

//usa funcoes pertencentes ao namespace std
using namespace std;

//funcao de entrada de dados da classe
void CPoco::EntradaDados()
{
    cout << "Informe_a_vazao_de_producao:" << endl;
    cin >> vazao;
    cin.get();

    cout << "Informe_o_tempo_de_producao:" << endl;
    cin >> tempoProducao;
    cin.get();

    cout << "Informe_a_pressao_no_poco:" << endl;
    cin >> pressaoPoco;
    cin.get();

    cout << "Informe_o_raio_do_poco:" << endl;
    cin >> raioPoco;
    cin.get();
}

//funcao que acusa e conserta erro de entrada
void CPoco::Erro()
{
    //repete a entrada enquanto o valor for equivocado

    while (tempoProducao<0.00)
    {
        cout << "Reinforme_o_tempo_de_producao:" << endl;
        cin >> tempoProducao;
        cin.get();
    }

    while (pressaoPoco<0.00)
    {
}

```

```

        cout << "Reinforme_a_pressão_no_poco:" << endl;
        cin >> pressaoPoco;
        cin .get();

    }

while (( raioPoco <0.00) || (raioPoco >3.0))
{
    cout << "Reinforme_o_raio_do_poco:" << endl;
    cin >> raioPoco;
    cin .get();
}

//set
void CPoco::Vazao(double _vazao)
{
    vazao = _vazao;
}

//get
double CPoco::Vazao() const
{
    return vazao;
}

//set
void CPoco::TempoProducao(double _tempoProducao)
{
    tempoProducao = _tempoProducao;
}

//get
double CPoco::TempoProducao() const
{
    return tempoProducao;
}

//set
void CPoco::PressaoPoco(double _pressaoPoco)
{

```

```

    pressaoPoco = _pressaoPoco;
}

//get
double CPoco::PressaoPoco() const
{
    return pressaoPoco;
}

//set
void CPoco::RaioPoco(double _raioPoco)
{
    raioPoco = _raioPoco;
}

//get
double CPoco::RaioPoco() const
{
    return raioPoco;
}

```

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

Listagem 6.3: Arquivo de cabeçalho da classe CReservatorio.h.

```

///Condicao para nao definir a classe mais de uma vez
#ifndef CReservatorio_h
#define CReservatorio_h

class CReservatorio;

///Classe contendo as caracteristicas do reservatorio
class CReservatorio
{
    ///privados, so acessados por meio de funcoes get
    private:

        ///porosidade do reservatorio
        double porosidade;
        ///altura do reservatorio
        double altura;
        ///sistema de unidades utilizado do reservatorio

```

```

double sistemaUnidade;

public:

///Funcao que recebe dados do usuario , preenchendo atributos por padrao
void EntradaDados();

///Funcao que exibe um menu e recebe dado do usuario , esse dado pode ser usado para a execucao de outras funcoes
void EntradaSistemaUnidades();

///Funcao que verifica se houve erro na entrada de dados e pede novo input
void Erro();

///Funcao que seta a porosidade
void Porosidade(double _porosidade);
///Funcao get da porosidade
double Porosidade() const;

///Funcao que seta a altura
void Altura(double _altura);
///Funcao get da porosidade
double Altura() const;

///Funcao que seta o sistemaunidade
void SistemaUnidade(double _sistemaUnidade);
///Funcao get do sistemaunidade
double SistemaUnidade() const;

};

#endif

```

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

Listagem 6.4: Arquivo de implementação da classe CReservatorio.cpp.

```
#include "CReservatorio.h"
```

```

//inclui a biblioteca iostrem pois usa funcoes de entrada e saida de dados
#include <iostream>

//usa funcoes pertencentes ao namespace std
using namespace std;

```

```
//funcao de entrada de dados da classe
void CReservatorio :: EntradaDados()
{
    cout << "Informe a porosidade da rocha reservatorio:" << endl;
    cin >> porosidade;
    cin .get ();

    cout << "Informe a altura do reservatorio:" << endl;
    cin >> altura;
    cin .get ();

}

//Funcao que preenche o sistemaunidade por um pequeno menu
void CReservatorio :: EntradaSistemaUnidades()
{
    cout << "Qual o sistema de unidades utilizado para informar os parametros"
        "1 - Americano ( Oilfield )" << endl << "2 - Brasileiro ( Petrobras )"
        "3 - Sistema Internacional" << endl;
    int i;
    cin >> i;
    cin .get ();

    //repete a entrada enquanto o valor for equivocado
    while (( i!=1)&&(i!=2)&&(i!=3))
    {
        cout << "Reinforme o sistema de unidades utilizado ." << endl;
        cin >> i;
        cin .get ();
    }

    if ( i==1)
        sistemaUnidade = 141.2;

    if ( i==2)
        sistemaUnidade = 19.03;

    if ( i==3)
```

```

        sistemaUnidade = 0.3183;
    }

//funcao que acusa e conserta erro de entrada
void CReservatorio::Erro()
{
    //repete a entrada enquanto o valor for equivocado
    while ( altura < 0.00 )
    {
        cout << "Reinforme a altura:" << endl;
        cin >> altura;
        cin .get ();
    }

    while (( porosidade < 0.00 ) || ( porosidade > 1.00 ))
    {
        cout << "Reinforme a porosidade:" << endl;
        cin >> porosidade;
        cin .get ();
    }

}

//set
void CReservatorio::Porosidade(double _porosidade)
{
    porosidade = _porosidade;
}

//get
double CReservatorio::Porosidade() const
{
    return porosidade;
}

//set
void CReservatorio::Altura(double _altura)
{
    altura = _altura;
}

```

```

//get
double CReservatorio :: Altura() const
{
    return altura;
}

//set
void CReservatorio :: SistemaUnidade(double _sistemaUnidade)
{
    sistemaUnidade = _sistemaUnidade;
}

//get
double CReservatorio :: SistemaUnidade() const
{
    return sistemaUnidade;
}

```

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

Listagem 6.5: Arquivo de cabeçalho da classe CFluido.h.

```

//Condicao para nao definir a classe mais de uma vez
#ifndef CFluido_h
#define CFluido_h

class CFluido;

//Classe contendo as caracteristicas do fluido produzido
class CFluido
{
    //privados, so acessados por meio de funcoes get
    private:

        //Fator Volume formacao do fluido
        double fatorVolumeFormacao;
        //Viscosidade
        double viscosidade;
        //Compressibilidade Total
        double compressibilidade;

```

```

public:

    ///Funcao que recebe dados do usuario , preenchendo atributos fato
    /// compressibilidade
    void EntradaDados();

    ///Funcao que verifica se houve erro na entrada de dados e pede n
    void Erro();

    ///Funcao que seta o fatorvolumeformacao
    void FatorVolumeFormacao(double _fatorVolumeFormacao);
    ///Funcao get do fatorvolumeformacao
    double FatorVolumeFormacao() const;

    ///Funcao que seta a viscosidade
    void Viscosidade(double _viscosidade);
    ///Funcao get da viscosidade
    double Viscosidade() const;

    ///Funcao que seta a compressibilidade
    void Compressibilidade(double _compressibilidade);
    ///Funcao get da compressibilidade
    double Compressibilidade() const;

};

#endif

```

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

Listagem 6.6: Arquivo de implementação da classe CFluido.cpp.

```

#include "CFluido.h"

//inclui a biblioteca iostrem pois usa funcoes de entrada e saida de dado
#include <iostream>

using namespace std;

//funcao de entrada de dados da classe
void CFluido::EntradaDados()
{

```

```

cout << "Informe_o_fator_volume-formacao_do_fluido:" << endl;
cin >> fatorVolumeFormacao;
cin.get();

cout << "Informe_a_viscosidade_do_fluido:" << endl;
cin >> viscosidade;
cin.get();

cout << "Informe_a_compressibilidade_total_(fluido+rocha):" << endl;
cin >> compressibilidade;
cin.get();
}

//funcao que acusa e conserta erro de entrada
void CFLuido::Erro()
{
//repete a entrada enquanto o valor for equivocado

while (fatorVolumeFormacao<0.00)
{
    cout << "Reinforme_o_Fator_Volume-Formacao:" << endl;
    cin >> fatorVolumeFormacao;
    cin.get();
}

while (viscosidade<0.00)
{
    cout << "Reinforme_a_viscosidade." << endl;
    cin >> viscosidade;
    cin.get();
}

while (compressibilidade<0.00)
{
    cout << "Reinforme_a_compressibilidade." << endl;
    cin >> compressibilidade;
    cin.get();
}
}

```

```

//set
void CFluido::FatorVolumeFormacao(double _fatorVolumeFormacao)
{
    fatorVolumeFormacao = _fatorVolumeFormacao;
}

//get
double CFluido::FatorVolumeFormacao() const
{
    return fatorVolumeFormacao;
}

//set
void CFluido::Viscosidade(double _viscosidade)
{
    viscosidade = _viscosidade;
}

//get
double CFluido::Viscosidade() const
{
    return viscosidade;
}

//set
void CFluido::Compressibilidade(double _compressibilidade)
{
    compressibilidade = _compressibilidade;
}

//get
double CFluido::Compressibilidade() const
{
    return compressibilidade;
}

```

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

Listagem 6.7: Arquivo de cabeçalho da classe CDadosRegistradorPressao.h.

```

//Condicao para nao definir a classe mais de uma vez
#ifndef CDadosRegistradorPressao_h
#define CDadosRegistradorPressao_h

//inclui a biblioteca vector pois ha declaracao de vetor
#include<vector>
#include<string>

class CDadosRegistradorPressao;

//Classe que contem dados registrados do registrador de pressao
class CDadosRegistradorPressao
{
    //privados, so acessados por meio de funcoes get
private:
    //pressao medida apos o fechamento da producao
    std::vector<double> pressaoMedida;
    //tempo apos o fechamento da producao em que foi medida a pressao
    std::vector<double> tempoSemProducao;

public:
    //Funcao que importa os dados registrados do arquivo .dat, preenche o vetor
    void Importa();

    //Funcao que seta a pressao medida
    void PressaoMedida(std::vector<double> _pressaoMedida);
    //Funcao get da posicao informada do vetor pressao medida
    double PressaoMedida(int posicao) const;
    //Funcao get da pressao medida
    std::vector<double> PressaoMedida() const;

    //Funcao que seta o tempo sem producao
    void TempoSemProducao(std::vector<double> _tempoSemProducao);
    //Funcao get da posicao informada do vetor tempo sem producao
    double TempoSemProducao(int posicao) const;
    //Funcao get do tempo sem producao
    std::vector<double> TempoSemProducao() const;

```

```
};  
#endif
```

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

Listagem 6.8: Arquivo de implementação da classe CDadosRegistradorPressao.cpp.

```
#include "CDadosRegistradorPressao.h" //inclui o cabecalho da classe  
  
//inclui biblioteca para importacao de arquivos de disco  
#include <fstream>  
  
//inclui a biblioteca iostream pois usa funcoes cin, cout  
#include <iostream>  
  
//inclui a biblioteca vector pois usa vetores  
#include <vector>  
  
//inclui a biblioteca string pois usa variaveis string  
#include <string>  
  
//usa funcoes pertencentes ao namespace std  
using namespace std;  
  
void CDadosRegistradorPressao :: Importa()  
{  
    //limpa os vetores de importacao para novo preenchimento  
    tempoSemProducao.resize(0);  
    pressaoMedida.resize(0);  
  
    //indica o que o eixo x representa  
    string eixox;  
    //indica o que o eixo y representa  
    string eixoy;  
    double x;  
    double y;  
    //nome do arquivo com os dados a serem importados  
    string nomeArquivo;  
  
    cout << "Informe o nome do arquivo com os dados do registrador de pressao:  
    //armazena a string digitada em nomearquivo  
    getline (cin, nomeArquivo);
```

```
//cria objeto de importacao
ifstream fin;
//converte a string de c++ para c, necessario para funcao
fin.open (nomeArquivo.c_str ());

//pega o primeiro valor, o nome do eixo x
fin >> eixox;
//pega o segundo valor, o nome do eixo y
fin >> eixoy;

//fazer ate o fim do arquivo
while (!fin.eof ())
{
    //valores de x e y alternados e separados por um espaco
    fin >> x;
    //para adicionar no fim do vetor, otimizando memoria
    tempoSemProducao.push_back (x);
    fin >> y;
    pressaoMedida.push_back (y);
}

//set
void CDadosRegistradorPressao::PressaoMedida (vector<double> _pressaoMedida)
{
    pressaoMedida = _pressaoMedida;
}

//get da posicao
double CDadosRegistradorPressao::PressaoMedida (int posicao) const
{
    return pressaoMedida [posicao];
}

//get
vector<double> CDadosRegistradorPressao::PressaoMedida () const
{
    return pressaoMedida;
}
```

```

//set
void CDadosRegistradorPressao :: TempoSemProducao( vector<double> _tempoSemProducao )
{
    tempoSemProducao = _tempoSemProducao;
}

//get da posicao
double CDadosRegistradorPressao :: TempoSemProducao( int posicao ) const
{
    return tempoSemProducao[ posicao ];
}

//get
vector<double> CDadosRegistradorPressao :: TempoSemProducao() const
{
    return tempoSemProducao;
}

```

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

Listagem 6.9: Arquivo de cabeçalho da classe CEstatistica.h.

```

///Condicao para nao definir a classe mais de uma vez
#ifndef CEstatistica_h
#define CEstatistica_h

///inclui vector pois ha parametros declarados que sao vetores
#include <vector>

class CEstatistica;

///Classe que calcula estatisticas do vetor, como media e desvio padrao,
class CEstatistica
{
private:

    double media;
    double desvio;

public:

```

```

///retorna a media do vetor informado
double Media( std :: vector<double> v );

///retorna o desvio padrao do vetor informado
double DesvioPadrao( std :: vector<double> v );

};

#endif

```

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

Listagem 6.10: Arquivo de implementação da classe **CEstatistica.cpp**.

```

#include "CEstatistica.h"

//inclui a biblioteca vector pois usa a funcao size
#include <vector>

//inclui a biblioteca cmath pois usa a funcao pow
#include <cmath>

using namespace std ;

double CEstatistica :: Media( vector<double> v )
{
    double soma = 0.0;
    //loop que faz a soma de todos os elementos do vetor
    for ( int i = 0 ; i < (v.size ()) ; i++)
        soma = soma + v[ i ];

    return media = soma/v.size ();
}

double CEstatistica :: DesvioPadrao( vector<double> v )
{
    double soma = 0.0;
    double vquadrado = 0.0;
    desvio = 0.0;

    //loop que faz a soma dos elementos do vetor elevados ao quadrado
    for ( int i = 0 ; i < (v.size ()) ; i++)

```

```

{
    soma = soma + v[ i ];
    vquadrado = vquadrado + (v[ i ]*v[ i ]);
}

return desvio = pow(((vquadrado - ((1.0/v.size())*soma*soma))/(v.size()))
}

```

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

Listagem 6.11: Arquivo de cabeçalho da classe CAjusteCurvaMinimosQuadrados.h.

```

//Condicao para nao definir a classe mais de uma vez
#ifndef CAjusteCurvaMinimosQuadrados_h
#define CAjusteCurvaMinimosQuadrados_h

//inclui o cabecalho da classe que sera utilizada
#include "CEstatistica.h"
//inclui vector pois ha parametros declarados que sao vetores
#include <vector>

class CAjusteCurvaMinimosQuadrados;

//Classe que obtém os coeficiente da regressao linear por meio do metodo
class CAjusteCurvaMinimosQuadrados
{
    //encapsulamento que permite o acesso para a classe e para a classe herada
protected:
    //coeficiente angular da reta do tipo y=ax+b
    double coeficienteAngular;
    //coeficiente linear da reta do tipo y=ax+b
    double coeficienteLinear;
    //coeficiente de correlacao da reta do tipo y=ax+b
    double coeficienteCorrelacao;

    //cria um objeto da classe CEstatistica para ser utilizado funcoes
    CEstatistica estatistica;

protected:

```

```

///Funcao que retorna o valor do coeficiente angular
double CoeficienteAngular ( std :: vector<double> vx , std :: vector<double> vy )
{
    ///Funcao que retorna o valor do coeficiente angular logaritimico
    double CoeficienteAngularLog ( std :: vector<double> vx , std :: vector<double> vy )
    {
        ///Funcao que retorna o valor do coeficiente linear
        double CoeficienteLinear ( std :: vector<double> vx , std :: vector<double> vy )
        {
            ///Set do P1h
            double CoeficienteLinearP1h ( double _P1h );
            ///Funcao que retorna o valor do coeficiente de correlacao
            double CoeficienteCorrelacao ( std :: vector<double> vx , std :: vector<double> vy )
            {
                #endif
            }
        }
    }
}

```

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

Listagem 6.12: Arquivo de implementação da classe CAjusteCurvaMinimosQuadrados.cpp.

```

#include "CAjusteCurvaMinimosQuadrados.h"

//inclui a biblioteca cmath pois usa a funcao logaritmica
#include <cmath>

//inclui a biblioteca vector pois usa funcoes dos vetores: push_back, resize
#include <vector>

//usa funcoes pertencentes ao namespace std
using namespace std ;

//retorna o valor do coeficiente angular
double CAjusteCurvaMinimosQuadrados :: CoeficienteAngular ( vector<double> vx ,
    vector<double> vy )
{
    double mnum = 0.0; //termo do denominador
    double mden = 0.0; //termo do numerador

    double mediax = estatistica . Media(vx);
    double mediay = estatistica . Media(vy);
}

```

```

for (int j=0 ; j<vx.size() ; j++) //percorre todo o vetor
{
    //metodo dos minimos quadrados
    mnum = mnum + (vx[j] * (vy[j] - mediay));
    mden = mden + (vx[j] * (vx[j] - mediam));
}

return coeficienteAngular = mnum/mden;

}

double CAjusteCurvaMinimosQuadrados:: CoeficienteAngularLog (std::vector<double>
{
    double mnum = 0.0; //termo do denominador
    double mden = 0.0; //termo do numerador

    double mediam = estatistica.Media(vx);
    double mediay = estatistica.Media(vy);

    mnum = log10(vx[vx.size() - 1]) - log10(vx[0]);
    mden = vy[vy.size() - 1] - vy[0];

    return coeficienteAngular = mden/mnum;
}

//retorna o valor do coeficiente linear
double CAjusteCurvaMinimosQuadrados:: CoeficienteLinear (vector<double> vx
{
    return coeficienteLinear = estatistica.Media(vy) - (coeficienteAngular *
}

double CAjusteCurvaMinimosQuadrados:: CoeficienteLinearP1h (double _P1h)
{
    return coeficienteLinear = _P1h;
}

//retorna o valor do coeficiente de correlacao
double CAjusteCurvaMinimosQuadrados:: CoeficienteCorrelacao (vector<double>
{

```

```

double somar = 0.0;
double variax = 0.0;
double variay = 0.0;

double mediax = estatistica.Media(vx);
double mediay = estatistica.Media(vy);

for (int j=0 ; j<vx.size() ; j++)
{
    somar = somar + ((vx[j] - mediax) * (vy[j] - mediay));
    variax = variax + (pow ((vx[j] - mediax),2));
    variay = variay + (pow ((vy[j] - mediay),2));
}

return coeficienteCorrelacao = -somar / pow ((variax * variay),0.5)
}

```

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

Listagem 6.13: Arquivo de cabeçalho da classe CAjusteCurva.h.

```

//Condicao para nao definir a classe mais de uma vez
#ifndef CAjusteCurva_h
#define CAjusteCurva_h

//inclui a biblioteca vector pois ha declaracao de vetor
#include <vector>

//inclui o cabecalho da classe pai
#include "CAjusteCurvaMinimosQuadrados.h"
#include "CMetodo.h"

//Declaracao da Classe filha de CAjusteCurvaMinimosQuadrados
class CAjusteCurva;

//Classe que executa a regressao linear e ajusta ate a correlacao satisfazente
class CAjusteCurva: public CAjusteCurvaMinimosQuadrados
{
public:
    ///coef. angular
    double m;
    ///coef. linear

```

```

double n;
//coef. de correlacao
double r;
//indica que o coef. de correlacao nao foi satisfatorio (ha estocagem)
bool efeitoEstocagem;
//ajuste de variavel para logaritmica
std::vector<double> logt;

public:
//Funcao que executa a regressao linear atraves do metodo dos minimos quadrados
void Regressao(std::vector<double> vx, std::vector<double> vy, double &n, double &r);
//Funcao que ajusta a regressao para o periodo correto, removendo os pontos de sao
void Ajuste(std::vector<double> vx, std::vector<double> vy, double &n, double &r);

};

//Fim da condicao de definicao da classe CAjusteCurva
#endif

```

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

Listagem 6.14: Arquivo de implementação da classe CAjusteCurva.cpp.

```

#include "CAjusteCurva.h"

//inclui a biblioteca iostream pois usa funcoes de entrada e saida de dados
#include <iostream>

//inclui a biblioteca cmath pois usa a funcao logaritmica
#include <cmath>

//inclui a biblioteca vector pois usa funcoes dos vetores: push_back, resize, etc
#include <vector>

//inclui biblioteca para usar modulo
#include <cstdlib>

//usa funcoes pertencentes ao namespace std
using namespace std;

// Funcao que cria a variavel logaritmica a partir dos parametros 1 e 3

```

```

// regressao linear atraves do metodo dos minimos quadrados.
void CAjusteCurva :: Recessao (vector<double> vx, vector<double> vy, double
{
    //limpa o vetor do eixo x para novo preenchimento
    logt.resize (0);

    //loop que percorre todo o vetor vx e preenche logt
    switch (_metodo)
    {
        case CMetodo ::METODO_HORNER:

            for (int i=0 ; i<vx.size () ; i++)
                //transformacao da variavel em logaritmica
                logt.push_back (log10 ((z+vx[i]) / vx[i]));

            m = CoeficienteAngular (logt ,vy);
            n = CoeficienteLinear (logt ,vy);
            r = CoeficienteCorrelacao (logt ,vy);

            break;
        case CMetodo ::METODO_MDH:

            for (int i=0 ; i<vx.size () ; i++)
                //transformacao da variavel em logaritmica
                logt.push_back (vx[i]);

            m = CoeficienteAngularLog (logt ,vy);
            n = CoeficienteLinearP1h (_P1h);
            r = CoeficienteCorrelacao (logt ,vy);

            break;
        default:
            break;
    }

    cout << "EQUACAO: y = " << m << " * x + " << n << endl << " r = " << r <
}

```

```

//Funcao que ajusta a regressao para o periodo correto , removendo os pontos extras
void CAjusteCurva::Ajuste(vector<double> vx, vector<double> vy, double z,
{
    // variavel que contem os coef. de correlacao
    vector<double> coef(vx.size() / 1.2, r);

    //variavel que ajusta o eixo y
    vector<double> y;

    //variavel que ajusta o eixo x
    vector<double> t;

    //loop principal que vai aumentando o valor de k e retirando as primeiras k-1
    for (int k=1 ; k<(vx.size() / 1.2) ; k++)
    {
        //repete o loop ate achar o coef. de correlacao aceitavel
        if(abs(coef[k-1])<0.9900)
        {
            //ocorre estocagem se cair na condicao
            efeitoEstocagem = true;
            cout << "Necessario novo Ajuste." << endl;

            //limpa os vetores
            t.resize(0);
            y.resize(0);

            switch (_metodo)
            {
                case CMetodo::METODO_HORNER :
                {
                    for(int l=0 ; l<vx.size() ; l++)
                    {
                        //define o eixo x
                        t.push_back(log10((z + vx[l])/vx[l]));
                        //define o eixo y
                        y.push_back(vy[l]);
                    }
                }
            }
        }
    }
}

```

```

for(int w=0 ; w<(vx.size()-k) ; w++)
{
    //retira o primeiro valor do vetor (estoc
    t[w] = t[w+k];
    //se repetir o if, vai retirando
    y[w] = vy[w+k];
    //até terminar a estocagem (coef. sera
}

// redefine o tamanho dos vetores
t.resize (vx.size()-k);
y.resize (vx.size()-k);

//nova regressao linear
m = CoeficienteAngular (t,y);
n = CoeficienteLinear (t,y);
//novo coeficiente de correlacao
coef[k] = CoeficienteCorrelacao (t,y);

cout << "EQUACAO: y = " << m << "*x + " << n <<
}
break;
```

**case** CMetodo::METODO\_MDH:

```

{
    for(int l=0; l<vx.size(); l++)
    {
        //define o eixo x
        t.push_back(vx[l]);
        //define o eixo y
        y.push_back (vy[l]);
    }

    for(int w=0 ; w<(vx.size()-k) ; w++)
    {
        //retira o primeiro valor do vetor (estoc
        t[w] = t[w+k];
        //se repetir o if, vai retirando
    }
}
```

```

cout << "t[x] = " << t[w] << endl;
y[w] = vy[w+k];
//até que terminar a estocagem (coef. sera
}

t.resize (vx.size()-k);
y.resize (vx.size()-k);

//nova regressao linear
m = CoeficienteAngularLog (t,y);
n = CoeficienteLinearP1h (_P1h);
//novo coeficiente de correlacao
coef[k] = CoeficienteCorrelacao (t,y);

cout << "EQUACAO: y = " << m << "*x + " << n <<
}
break;

default:
    break;
}

if (abs(1.2*coef[k])<coef[0])
    cout << "Regressao Linear nao tao perfeita, Indicando que a regressao e linear." << endl;

if (k>1) //apos a segunda regressao
{
    if ((abs(1.2*coef[k]))<coef[k-1])
    {
        k = vx.size()/1.2; //para terminar o loop
        cout << "Maximo Coeficiente de Correlacao atingido." << endl;
    }
}

} // Fecha a condicao inicial.
} // Fecha loop.
} // Fecha o Metodo.

```

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

Listagem 6.15: Arquivo de cabeçalho da classe CBuildUp.h.

```
#ifndef CBUILDUP_H
#define CBUILDUP_H

/**
 * Classe virtual para metodos aplicaveis
 * Declaração dos calculos basicos de reservatorio
 *
 * @param m_Permeabilidade, m_Viscosidade
 *
*/
class CBuildUP {

public:
    virtual ~CBuildUP() = default;
    virtual void CalculoPermeabilidade(double _sistemaUnidade, double _vazaao);
    virtual void CalculoFatorPelicula(double _m, double _tempoProducao, double _tempoConsumo);

    double getPermeabilidade() { return m_Permeabilidade; }
    double getPelicula() {return m_Pelicula; }

protected:
    double m_Permeabilidade;
    double m_Pelicula;

};

#endif
```

Apresenta-se na listagem ?? o arquivo de implementação da classe CBuildUp.

Listagem 6.16: Arquivo de implementação da classe CBuildUp.cpp.

```
#include "CBuildUp.h"
```

```
void CBuildUP::CalculoPermeabilidade(double _sistemaUnidade, double _vazaao)
```

```
{
}

void CBuildUP:: CalculoFatorPelicula(double _m, double _tempoProducao, double _tempoProducao)
{
}
```

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

Listagem 6.17: Arquivo de cabeçalho da classe CMetodo.h.

```
#ifndef CMETODO_H
#define CMETODO_H

/**
*
* Metodos atualmente compatíveis
*
* @param none , HORNER, MDH
*
*/

enum class CMetodo {

    none = 0, METODO_HORNER, METODO_MDH

};

#endif
```

Apresenta-se na listagem 6.18 o arquivo com código da classe CMetodoMDH.

Listagem 6.18: Arquivo de cabeçalho da classe CMetodoMDH.h.

```
#ifndef METODOMDH_H
#define METODOMDH_H

#include "CBuildUp.h"

class CMetodoMDH : public CBuildUP{

    public:
```

```

CMetodoMDH() { };
~CMetodoMDH() { };

virtual void CalculoPermeabilidade(double _sistemaUnidade, double _vazao);
virtual void CalculoFatorPelicula(double _m, double _tempoProducao);

};

#endif

```

Apresenta-se na listagem 6.19 o arquivo de implementação da classe CMetodoMDH.

Listagem 6.19: Arquivo de implementação da classe CMetodoMDH.cpp.

```

#include "CMetodoMDH.h"

#include <iostream>
#include <cmath>

void CMetodoMDH::CalculoPermeabilidade(double _sistemaUnidade, double _vazao)
{
    m_Permeabilidade = (1.151 * _sistemaUnidade * _vazao * _fatorVolumeFoco *
                         _viscosidade) / (_m * _altura);

    std :: cout << "Permeabilidade:" << m_Permeabilidade;

    if (( _sistemaUnidade==141.2) || ( _sistemaUnidade==19.03))
        std :: cout << " milidarcy" << std :: endl;

    if ( _sistemaUnidade==0.3183)
        std :: cout << " metros quadrados" << std :: endl;

}

void CMetodoMDH::CalculoFatorPelicula(double _m, double _tempoProducao, double _n,
{
    m_Pelicula = 1.151 * (((_m * log10(_tempoProducao)) + _n - _pressao) /
                           (_m) - log10((_sistemaUnidade * m_Permeabilidade) /
                                         (_porosidade * _viscosidade * _compressibilidade *
                                         _raioPoco * _raioPoco)) - 0.3514 + log10 (_tempoProducao));
}

```

```

    std :: cout << "Fator_de_Pelicula:" << m_Pelicula << std :: endl;

}

```

Apresenta-se na listagem 6.20 o arquivo com código da classe CMetodoHorner.

Listagem 6.20: Arquivo de cabecalho da classe CMetodoHorner.h.

```

#ifndef METODOHORNER_H
#define METODOHORNER_H

#include "CBuildUp.h"

class CMetodoHorner : public CBuildUP {

public:

    CMetodoHorner() {};
    ~CMetodoHorner() {};

    virtual void CalculoPermeabilidade(double _sistemaUnidade, double _vazao);
    virtual void CalculoFatorPelicula(double _m, double _tempoProducao);

};

#endif

```

Apresenta-se na listagem 6.21 o arquivo de implementação da classe CMetodoHorner.

Listagem 6.21: Arquivo de implementação da classe CMetodoHorner.cpp.

```

#include "CMetodoHorner.h"

#include <iostream>
#include <cmath>

void CMetodoHorner::CalculoPermeabilidade(double _sistemaUnidade, double _vazao)
{
    m_Permeabilidade = (1.151 * _sistemaUnidade * _vazao * _fatorVolumeFoco *
                         _viscosidade) / (-_m * _altura);
}

```

```

    std :: cout << "Permeabilidade:" << m_Permeabilidade;

    if (( _sistemaUnidade==141.2) || (_sistemaUnidade==19.03))
        std :: cout << " milidarcy" << std :: endl;

    if (_sistemaUnidade==0.3183)
        std :: cout << " metros quadrados" << std :: endl;

}

void CMetodoHorner::CalculoFatorPelicula(double _m, double _tempoProducao)
{
    m_Pelicula = 1.151 * (((_m * log10(_tempoProducao)) + _n - _pressao)
                           - _m) - log10((_sistemaUnidade * m_Permeabilidade)
                           (_porosidade * _viscosidade * _compressibilidade
                           * _raioPoco * _raioPoco)) - 0.3514 + log10 (_tempoProducao);

    std :: cout << "Fator_de_Pelicula:" << m_Pelicula << std :: endl;
}

```

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

Listagem 6.22: Arquivo de cabeçalho da classe CCaracterizacaoReservatorio.h.

```

//Condicao para nao definir a classe mais de uma vez
#ifndef CCaracterizacaoReservatorio_h
#define CCaracterizacaoReservatorio_h

class CCaracterizacaoReservatorio;

//Classe que caracteriza o reservatorio.
class CCaracterizacaoReservatorio
{
public:
    //Funcao que analisa os resultados e caracteriza o reservatorio.
    void Caracterizacao(double permeabilidade, double fatorpelicula,
};

```

```
#endif
```

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

Listagem 6.23: Arquivo de implementação da classe `CCaracterizacaoReservatorio.cpp`.

```
#include "CCaracterizacaoReservatorio.h"
```

```
//inclui a biblioteca iostream pois usa funcoes de entrada e saida de dados
```

```
#include <iostream>
```

```
//usa funcoes pertencentes ao namespace std
```

```
using namespace std;
```

```
//Funcao que caracteriza o reservatorio, dado os parametros calculados
```

```
void CCaracterizacaoReservatorio::Caracterizacao (double permeabilidade,
```

```
{
```

```
//condicoes que se satisfeitas, escrevem na tela caracteristicas do
```

```
if (permeabilidade<=10)
```

```
cout << "1 - Reservatorio com permeabilidade ruim." << endl;
```

```
if ((permeabilidade<100)&&(permeabilidade>10))
```

```
cout << "1 - Reservatorio com permeabilidade boa." << endl;
```

```
if (permeabilidade>=100)
```

```
cout << "1 - Reservatorio com permeabilidade excelente." << endl;
```

```
if (fatorpelicula==0)
```

```
cout << "2 - Reservatorio sem dano e sem estimulo." << endl;
```

```
if (fatorpelicula<0)
```

```
cout << "2 - Reservatorio estimulado" << endl;
```

```
if ((fatorpelicula>0)&&(fatorpelicula <5))
```

```
cout << "2 - Reservatorio com dano baixo, nao necessita de processos"
```

```
if ((fatorpelicula >5)&&(fatorpelicula <10))
```

```
cout << "2 - Reservatorio com dano intermediario, pode ser usado para"
```

```
if (fatorpelicula >10)
```

```

cout << "2 - Reservatorio_com_dano_alto , necessita_de_processos_de_"

if (indiceprodutividade <=0.01)
    cout << "3 - Reservatorio_com_produtividade_baixo , considerar_uso_de_"

if ((indiceprodutividade >0.01)&&(indiceprodutividade <0.1))
    cout << "3 - Reservatorio_com_produtividade_regular , considerar_uso_de_"

if (indiceprodutividade >0.1)
    cout << "3 - Reservatorio_com_produtividade_boa ." << endl;

if ((raioefetivo / raiopoco)<=0.0001)
    cout << "4 - Razao_de_dano_alto , pois_o_raio_efetivo_e_muito_menor_q

}

```

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

Listagem 6.24: Arquivo de cabeçalho da classe CSimuladorAnaliseTestePressao.h.

```

#ifndef CSimuladorAnaliseTestePressao_h
#define CSimuladorAnaliseTestePressao_h

#include <fstream>
#include <iostream>
//inclusao de arquivos de classe necessarios

#include "cgnuplot.h"
#include "CReservatorio.h"
#include "CFluido.h"
#include "CPoco.h"
#include "CDadosRegistradorPressao.h"
#include "CEstatistica.h"
#include "CAjusteCurva.h"
#include "CCaracterizacaoReservatorio.h"

#include "CMetodo.h"
#include "CBuildUp.h"
#include "CMetodoHorner.h"
#include "CMetodoMDH.h"

```

```

///criacao do objeto caracterizar da classe CCaracterizacao.h"

class CSimuladorAnaliseTestePressao;

//Classe que faz a analise do teste de pressao realizado no campo e inf...
class CSimuladorAnaliseTestePressao
{
    public:
        ///Nome do Arquivo exportado
        std::string nome;
        ///pressao inicial que se encontrava o reservatorio
        double pressaoInicial;
        ///queda de pressao referente ao fator de pelicula
        double pressaoDano;
        ///indice de produtividade do reservatorio
        double indiceProdutividade;
        ///eficiencia de fluxo do reservatorio
        double eficienciaFluxo;
        ///raio efetivo do poco
        double raioEfetivo;
        ///coeficiente de estocagem do poco
        double coeficienteEstocagem;
        ///tempo de duracao do efeito de estocagem
        double tempoEstocagem;
        ///cria objeto de armazenamento de dados
        std::ofstream fout;
        ///criacao do objeto poco da classe CPoco
        CPoco poco;
        ///criacao do objeto fluido da classe CFluido
        CFluido fluido;
        ///criacao do objeto reservatorio da classe CReservatorio
        CReservatorio reservatorio;
        ///criacao do objeto registrador da classe CRegistrador
        CDadosRegistradorPressao registrador;
        ///criacao do objeto ajuste da classe CAjuste
        CAjusteCurva ajuste;
        ///criacao do objeto caracterizar da classe CCaracterizacao
        CCaracterizacaoReservatorio caracterizar;

```

```

    ///criacao do objeto buildup
    CBuildUP* buildup;
    ///criacao do objeto plot da classe CGnuplot
    //CGnuplot plot;

public:

    ///Funcao que calcula e preenche o atributo permeabilidade
void CalculoPermeabilidade ();
    ///Funcao que calcula e preenche o atributo pressao inicial
void CalculoPressaoInicial ();
    ///Funcao que calcula e preenche o atributo fator pelicula
void CalculoFatorPelicula ();
    ///Funcao que calcula e preenche o atributo raio efetivo
void CalculoRaioEfetivo ();
    ///Funcao que calcula e preenche os atributos indice produtividade
void CalculoIndices ();
    ///Funcao que calcula e preenche os atributos coeficiente estocagem
void CalculoEstocagem ();
    ///Funcao que chama as entradas de dados necessarias
void EntradaDados ();
    ///Funcao principal que executa a simulacao do teste
void Executar ();

    ///Varia parametros de reservatorio
void Variacao();

    ///Funcao que exporta os dados para um arquivo.dat
void Exporta ();

};

#endif

```

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

Listagem 6.25: Arquivo de implementação da classe CSimuladorAnaliseTestePressao.cpp.

```
#include "CSimuladorAnaliseTestePressao.h"
```

```
//inclui a biblioteca iostream pois usa funcoes de entrada e saida de dados
#include <iostream>

//inclui a biblioteca cmath pois usa a funcao logaritmica
#include <cmath>

//inclui a biblioteca vector pois usa funcoes dos vetores: push_back, resize, etc
#include <vector>

//inclui a biblioteca que guarda os dados
#include <fstream>

//inclui biblioteca string que le caracteres
#include <string>

//biblioteca que permite manipulacao de variaveis
#include <sstream>

//usa funcoes pertencentes ao namespace std
using namespace std;

///sobrecarga operador >>
std::istream &operator>>(std::istream &is, CMetodo &metodo)
{
    unsigned int a;
    is >> a;
    metodo = static_cast<CMetodo>(a);

    return is;
}

///Funcao principal que executa a simulacao do teste
void CSimuladorAnaliseTestePressao::Executar()
{
    cout << endl << "PROGRAMA_PARA_CALCULO_DE_PARAMETROS_DE_RESERVATORIO"
        << endl << "1-Rodar_o_Programa" << endl << "2-Sair" << endl << endl;
    int i = 0;
    cin >> i;

    while (i==1) //quando terminar a execucao do programa, se o usuario quiser
        {
            cout << endl << "1-Rodar_o_Programa" << endl << "2-Sair" << endl << endl;
            cout << endl << endl;
            cout << endl << endl;
        }
}
```

```

{

    CMetodo ans;
    do
    {
        cout << "Escolha_o_metodo_a_ser_aplicado:_1_-_Horner,_2_-_";
        cin >> ans;
        cin . get ();

        switch (ans)
        {
            case CMetodo::METODO_HORNER:
                buildup = new CMetodoHorner ();
                break;
            case CMetodo::METODO_MDH:
                buildup = new CMetodoMDH ();
                break;
            default:
                cout << "Opcão_invalida!!" << endl;
                ans = CMetodo::none;
                break;
        }
    }

}while (ans == CMetodo::none);

cout << "Entrada_de_Dados_do teste_de_pressao_realizado" << endl
     << "_____"
     << endl;

EntradaDados ();
double P1h = 0.0;
if (ans == CMetodo::METODO_MDH)
{
    cout << "entre_com_valor_P1h" << endl;
    cin >> P1h;
    cin . get ();
}

cout << "Entrada_de_Dados_Finalizada" << endl
     << "_____"
     << endl;
cout << "Importacao_dos_dados_no_registrador_de_Pressao" << endl
     << "_____"
     << endl;

```

```

    registrador.Importa();
    cout << "Dados do registrador importados com sucesso" << endl;

    cout << "Regressao Linear dos Dados" << endl
        << "__________________________________"
        ajuste.Regressao( registrador.TempoSemProducao() , registrador.Pri
        cin.get();
        cout << "Localizando o Periodo Transiente" << endl
        << "__________________________________"

        ajuste.Ajuste( registrador.TempoSemProducao() , registrador.Pri
        cout << "Regressao linear feita com sucesso" << endl
        << "__________________________________"
        cout << "Ajuste N = " << ajuste.n << endl;

//GERA O GRAFICO CASO USUARIO QUEIRA

cout << "Deseja gerar o grafico : " << endl << "1-Sim" << endl;
int j;
cin >> j;

if (j==1)
{
    CGnuplot plot;
    //gera o grafico com a reta perfeita obtida
    if (ans == CMetodo::METODO_HORNER)
    {
        plot.plot_slope (ajuste.m, ajuste.n);
        cin.get();
        if (ajuste.efeitoEstocagem==1)
            //compara com os pontos originais
            plot.plot_xy (ajuste.logt, registrador.PressaoMe
            cin.get();
    } else if (ans == CMetodo::METODO_MDH)
    {

```

```

        plot.XLogscale();
        plot.plot_logSlope (ajuste.m, ajuste.n);
        cin.get();
        if (ajuste.efeitoEstocagem==1)
            //compara com os pontos originais
            plot.plot_xy (ajuste.logt, registrador.PressaoMedia);
        cin.get();
    }

}

cout << "Parametros do Reservatorio" << endl
<< "____"
//CALCULOS
CalculoPermeabilidade();
CalculoPressaoInicial();
CalculoFatorPelicula(); //guardar
CalculoRaioEfetivo();
CalculoIndices();
Exporta();
cin.get();

//se nao ocorreu estocagem
if (ajuste.efeitoEstocagem==false)
    cout << "Reservatorio sem o periodo de estocagem" << endl
else
{
    CalculoEstocagem ();
    //zera o valor em caso de novo calculo
    ajuste.efeitoEstocagem = false;
}

cout << "Caracterizacao do Reservatorio." << endl
<< "____"

caracterizar.Caracterizacao(buildup->getPermeabilidade(), buildup->getPressaoMedia());
cin.get();

Variacao();

```

```

    //Nova Escolha
    cout << "\n1-Rodar_o_Programa" << endl << "2-Sair" << endl <
    cin >> i;
}
}

//chama as outras entradas de dados
void CSimuladorAnaliseTestePressao::EntradaDados()
{
    reservatorio . EntradaSistemaUnidades ();
    reservatorio . EntradaDados ();
    reservatorio . Erro ();
    fluido . EntradaDados ();
    fluido . Erro ();
    poco . EntradaDados ();
    poco . Erro ();
}

//Calcula a permeabilidade
void CSimuladorAnaliseTestePressao::CalculoPermeabilidade ()
{
    buildup->CalculoPermeabilidade (reservatorio . SistemaUnidade () , poco .

}

//Calcula a pressao inicial
void CSimuladorAnaliseTestePressao::CalculoPressaoInicial ()
{
    pressaoInicial = ajuste . n;
    cout << "Pressao_Inicial:" << pressaoInicial;

    if ( reservatorio . SistemaUnidade () == 141.2)
        cout << " psi" << endl;

    if ( reservatorio . SistemaUnidade () == 19.03)
        cout << " kgf/cm2" << endl;

    if ( reservatorio . SistemaUnidade () == 0.3183)
        cout << " Pascal" << endl;
}

```

```
}
```

```
//Calcula fator pelicula
```

```
void CSimuladorAnaliseTestePressao :: CalculoFatorPelicula ()
{
    buildup->CalculoFatorPelicula(ajuste.m, poco.TempoProducao(), ajuste);
}
```

```
//Calcula raio efetivo
```

```
void CSimuladorAnaliseTestePressao :: CalculoRaioEfetivo ()
{
    raioEfetivo = poco.RaioPoco() * exp(-buildup->getPelicula ());
    cout << "Raio_Efetivo:" << raioEfetivo;

    if ((reservatorio.SistemaUnidade() == 0.3183) || (reservatorio.SistemaUnidade() == 141.2))
        cout << "metros" << endl;

    if (reservatorio.SistemaUnidade() == 141.2)
        cout << "ft" << endl;
}
```

```
//Calcula do indice de produtividade , eficiencia de fluxo e queda de pressao
```

```
void CSimuladorAnaliseTestePressao :: CalculoIndices ()
{
    pressaoDano = 0.869 * (-ajuste.m) * buildup->getPelicula ();

    indiceProdutividade = poco.Vazao() / (pressaoInicial - poco.PressaoPoco());
    eficienciaFluxo = (pressaoInicial - poco.PressaoPoco()) - pressaoDano;

    cout << "Queda_de_Pressao_devido_ao_dano:" << pressaoDano << endl;

    //Exibir em porcentagens
    cout << "Indice_de_Produtividade:" << indiceProdutividade * 100.0 << endl;
    cout << "Eficiencia_de_Fluxo:" << eficienciaFluxo * 100.0 << "%" << endl;
}
```

```

void CSimuladorAnaliseTestePressao :: Variacao()
{
    cout << "____________________________________" << endl;
    cout << "Deseja variar algum parametro?" << endl;
    cout << "1-Sim | 2-Nao" << endl;
    int resp;
    cin >> resp;
    cin .get ();
    do
    {
        if ( resp!=1 && resp!=2)
        {
            cout << "Alternativa Invalida" << endl;
            cout << "1-Porosidade | 2-Fator Volume de Formacao | 3-Comprimento" << endl;
            cin >> resp;
            cin .get ();
        }
    }
    while ( resp!=1 && resp!=2);
    if ( resp == 1)
    {
        cout << "\nQual parametro deseja variar?" << endl;
        cout << "| 1-Porosidade | 2-Fator Volume de Formacao | 3-Comprimento" << endl;
        cin >> resp;
        cin .get ();
    }
    do
    {
        if ( resp!=1 && resp!=2 && resp!=3 && resp!=4)
        {
            cout << "Alternativa Invalida" << endl;
            cout << "| 1-Porosidade | 2-Fator Volume de Formacao | 3-Comprimento" << endl;
            cin >> resp;
            cin .get ();
        }
    }
    while ( resp!=1 && resp!=2 && resp!=3 && resp!=4);
    int intervalo;
    double dp;
    switch ( resp)
    {

```

```

case 1:
    cout << "Digite um valor inicial para porosidade" << endl;
    double porosidadeInicial;
    cin >> porosidadeInicial;
    cin .get ();
    while ((porosidadeInicial < 0.00) || (porosidadeInicial > 1.00))
    {
        cout << "Reinforme a porosidade Inicial:" << endl;
        cin >> porosidadeInicial;
        cin .get ();
    }
    cout << "Digite um valor Final para porosidade" << endl;
    double porosidadeFinal;
    cin >> porosidadeFinal;
    cin .get ();
    while ((porosidadeFinal < 0.00) || (porosidadeFinal > 1.00) || (porosidadeFinal < porosidadeInicial))
    {
        cout << "Reinforme a porosidade Final:" << endl;
        cin >> porosidadeFinal;
        cin .get ();
    }
    cout << "Em quantos intervalos deseja dividir a porosidade?" << endl;
    cin >> intervalo;
    cin .get ();
    dp=(porosidadeFinal - porosidadeInicial)/intervalo;
    for (int x = porosidadeInicial; porosidadeInicial <= porosidadeFinal; x++)
    {
        reservatorio.Porosidade(porosidadeInicial);
        cout << "\nCalculo para porosidade:" << porosidadeInicial << endl;
        cout << "-----" << endl;
        CalculoPermeabilidade();
        CalculoPressaoInicial();
        CalculoFatorPelicula(); //guardar
        CalculoRaioEfetivo();
        CalculoIndices();
        cout << "-----" << endl;
        ostringstream strs;
        strs << porosidadeInicial;
        string porosidadeString = strs.str();
    }

```

```

    string formato = ".dat";
    string Saida = nome+"porosidade"+porosidadeString+formato;
                    // abre arquivo
fout.open (Saida.c_str());

fout << "Permeabilidade:" << buildup->getPermeabilidade();
if ((reservatorio.SistemaUnidade()==141.2) || (reservatorio.Sistema
fout << "_milidarcy" << endl;
if (reservatorio.SistemaUnidade()==0.3183)
    fout << "_metros_quadrados" << endl; //criacao do objeto armaze
fout << "Pressao_Inicial:" << pressaoInicial;

if (reservatorio.SistemaUnidade()==141.2)
    fout << "_psi" << endl;

if (reservatorio.SistemaUnidade()==19.03)
    fout << "_kgf/cm2" << endl;

if (reservatorio.SistemaUnidade()==0.3183)
    fout << "_Pascal" << endl;
    fout << "Fator_de_Pelicula:" << buildup->getPelicula() << endl;
    fout << "Raio_Efetivo:" << raioEfetivo;

if ((reservatorio.SistemaUnidade()==0.3183) || (reservatorio.Sistem
fout << "_metros" << endl;

if (reservatorio.SistemaUnidade()==141.2)
    fout << "_ft" << endl;
    fout << "Queda_de_Pressao_devido_ao_dano:" << pressaoDano << endl;

//Exibir em porcentagens
fout << "Indice_de_Produtividade:" << indiceProdutividade*100.0 << "%"
fout.close();
cout << "Dados_Salvos_com_sucesso!" << endl;
cout << "-----" << endl;
cin.get();
}

break;

```

```

case 2:
    cout << "Digite um valor inicial para fator volume de formacao" <
    double FatorVolformacaoInicial;
    cin >> FatorVolformacaoInicial;
    cin .get ();
    while ((FatorVolformacaoInicial < 0.00))
    {
        cout << "Reinforme o fator volume de formacao Inicial:" << endl;
        cin >> FatorVolformacaoInicial;
        cin .get ();
    }
    cout << "Digite um valor Final para fator volume de formacao" <<
    double FatorVolformacaoFinal;
    cin >> FatorVolformacaoFinal;
    cin .get ();
    while ((FatorVolformacaoFinal < 0.00)|| (FatorVolformacaoInicial >Fat
    {
        cout << "Reinforme o fator volume de formacao Final:" << endl;
        cin >> FatorVolformacaoFinal;
        cin .get ();
    }
    cout << "Em quantos intervalos deseja dividir o fator volume de f
    cin >> intervalo;
    cin .get ();
    dp=(FatorVolformacaoFinal - FatorVolformacaoInicial)/intervalo;
    for (int x = FatorVolformacaoInicial; FatorVolformacaoInicial <= Fator
    {
        fluido.FatorVolumeFormacao(FatorVolformacaoInicial);
        cout << "\nCalculo para fator volume de formacao:" << FatorVolf
        cout << "_____"
        CalculoPermeabilidade();
        CalculoPressaoInicial();
        CalculoFatorPelicula(); //guardar
        CalculoRaioEfetivo();
        CalculoIndices();
        cout << "_____"
        ostrngstream strs;
        strs << FatorVolformacaoInicial;
        string FatorVolformacaoString = strs.str();

```

```

    string formato = ".dat";
    string Saida = nome+FatorVolF("+" + FatorVolformacaoString + formato;
                                    // abre arquivo
    fout.open (Saida.c_str ());

    fout << "Permeabilidade:" << buildup->getPermeabilidade ();
    if ((reservatorio.SistemaUnidade() == 141.2) || (reservatorio.Sistema
    fout << " milidarcy" << endl;
    if (reservatorio.SistemaUnidade() == 0.3183)
        fout << " metros quadrados" << endl; // criação do objeto armazena
    fout << "Pressao_Inicial:" << pressaoInicial;

    if (reservatorio.SistemaUnidade() == 141.2)
        fout << " psi" << endl;

    if (reservatorio.SistemaUnidade() == 19.03)
        fout << " kgf/cm2" << endl;

    if (reservatorio.SistemaUnidade() == 0.3183)
        fout << " Pascal" << endl;
        fout << "Fator_de_Pelicula:" << buildup->getPelicula () << endl;
        fout << "Raio_Efetivo:" << raioEfetivo;

    if ((reservatorio.SistemaUnidade() == 0.3183) || (reservatorio.Sistema
    fout << " metros" << endl;

    if (reservatorio.SistemaUnidade() == 141.2)
        fout << " ft" << endl;
        fout << "Queda_de_Pressao_devido_ao_dano:" << pressaoDano << endl;

        // Exibir em porcentagens
    fout << "Indice_de_Produtividade:" << indiceProdutividade * 100.0 << "%"
    fout.close ();
    cout << "Dados_Salvos_com_sucesso!" << endl;
    cout << "-----" << endl;
    cin.get ();
}

break;

```

```

case 3:
    cout << "Digite_um_valor_inicial_para_Compressibilidade" << endl;
    double CompressibilidadeIncial;
    cin >> CompressibilidadeIncial;
    cin .get ();
    while (( CompressibilidadeIncial < 0.00))
    {
        cout << "Reinforme_a_compressibilidade:" << endl;
        cin >> CompressibilidadeIncial;
        cin .get ();
    }
    cout << "Digite_um_valor_Final_para_Compressibilidade" << endl;
    double CompressibilidadeFinal;
    cin >> CompressibilidadeFinal;
    cin .get ();
    while (( CompressibilidadeFinal < 0.00) || ( CompressibilidadeIncial
    {
        cout << "Reinforme_a_Compressibilidade:" << endl;
        cin >> CompressibilidadeFinal;
        cin .get ();
    }
    cout << "Em_quantos_intervalos_deseja_dividir_o_Compressibilidade"
    cin >> intervalo;
    cin .get ();
    dp=(CompressibilidadeFinal - CompressibilidadeIncial)/intervalo;
    for (int x = CompressibilidadeIncial; CompressibilidadeIncial <= Co
    {
        fluido .Compressibilidade(CompressibilidadeIncial);
        cout << "\nCalculo_para_Compressibilidade:" << CompressibilidadeInici
        cout << "_____" << endl;
        CalculoPermeabilidade();
        CalculoPressaoInicial();
        CalculoFatorPelicula(); //guardar
        CalculoRaioEfetivo();
        CalculoIndices();
        cout << "_____" << endl;

        ostringstream strs;
        strs << CompressibilidadeIncial;

```

```

    string CompressibilidadeString = strs.str();

    string formato = ".dat";
    string Saida = nome+"Compressibilidade"+CompressibilidadeString+formato;
    // abre arquivo
    fout.open (Saida.c_str());

    fout << "Permeabilidade:" << buildup->getPermeabilidade();
    if ((reservatorio.SistemaUnidade()==141.2) || (reservatorio.Sistema
    fout << "_milidarcy" << endl;
    if (reservatorio.SistemaUnidade()==0.3183)
        fout << "_metros_quadrados" << endl; //criacao do objeto armaze
    fout << "Pressao_Inicial:" << pressaoInicial;

    if (reservatorio.SistemaUnidade()==141.2)
        fout << "_psi" << endl;

    if (reservatorio.SistemaUnidade()==19.03)
        fout << "_kgf/cm2" << endl;

    if (reservatorio.SistemaUnidade()==0.3183)
        fout << "_Pascal" << endl;
        fout << "Fator_de_Pelicula:" << buildup->getPelicula() << endl;
        fout << "Raio_Efetivo:" << raioEfetivo;

    if ((reservatorio.SistemaUnidade()==0.3183) || (reservatorio.Sistem
        fout << "_metros" << endl;

    if (reservatorio.SistemaUnidade()==141.2)
        fout << "_ft" << endl;
        fout << "Queda_de_Pressao_devido_ao_dano:" << pressaoDano << endl;

        //Exibir em porcentagens
    fout << "Indice_de_Produtividade:" << indiceProdutividade*100.0 << "%";
    fout.close();
    cout << "Dados_Salvos_com_sucesso!" << endl;
    cout << "_____________________________________" << endl;
    cin.get();
}

break;

```

```

case 4:
    cout << "Digite_um_valor_inicial_para_Viscosidade" << endl;
    double ViscosidadeInicial;
    cin >> ViscosidadeInicial;
    cin .get ();
    while ((ViscosidadeInicial < 0.00))
    {
        cout << "Reinforme_a_compressibilidade:" << endl;
        cin >> ViscosidadeInicial;
        cin .get ();
    }
    cout << "Digite_um_valor_Final_para_Viscosidade" << endl;
    double ViscosidadeFinal;
    cin >> ViscosidadeFinal;
    cin .get ();
    while ((ViscosidadeFinal < 0.00) || (ViscosidadeInicial > ViscosidadeFinal))
    {
        cout << "Reinforme_a_Viscosidade:" << endl;
        cin >> ViscosidadeFinal;
        cin .get ();
    }
    cout << "Em_quantos_intervalos_deseja_dividir_o_Viscosidade?" << endl;
    cin >> intervalo;
    cin .get ();
    dp=(ViscosidadeFinal - ViscosidadeInicial)/intervalo;
    for (int x = ViscosidadeInicial; ViscosidadeInicial <= ViscosidadeFinal;
    {
        fluido .Viscosidade(ViscosidadeInicial);
        cout << "\nCalculo_para_Viscosidade:" << ViscosidadeInicial << endl;
        cout << "_____" << endl;
        CalculoPermeabilidade();
        CalculoPressaoInicial();
        CalculoFatorPelicula(); //guardar
        CalculoRaioEfetivo();
        CalculoIndices();
        cout << "_____" << endl;
        ostrstringstream strs;
        strs << ViscosidadeInicial;
    }
}

```

```

    string ViscosidadeString = strs.str();

    string formato = ".dat";
    string Saida = nome+"Viscosidade"+ViscosidadeString+formato;
                    // abre arquivo
fout.open (Saida.c_str());

fout << "Permeabilidade:" << buildup->getPermeabilidade();
if ((reservatorio.SistemaUnidade()==141.2) || (reservatorio.Sistema
fout << "_milidarcy" << endl;
if (reservatorio.SistemaUnidade()==0.3183)
    fout << "_metros_quadrados" << endl; //criacao do objeto armaze
fout << "Pressao_Inicial:" << pressaoInicial;

if (reservatorio.SistemaUnidade()==141.2)
    fout << "_psi" << endl;

if (reservatorio.SistemaUnidade()==19.03)
    fout << "_kgf/cm2" << endl;

if (reservatorio.SistemaUnidade()==0.3183)
    fout << "_Pascal" << endl;
    fout << "Fator_de_Pelicula:" << buildup->getPelicula() << endl;
    fout << "Raio_Efetivo:" << raioEfetivo;

if ((reservatorio.SistemaUnidade()==0.3183) || (reservatorio.Sistem
fout << "_metros" << endl;

if (reservatorio.SistemaUnidade()==141.2)
    fout << "_ft" << endl;
    fout << "Queda_de_Pressao_devido_ao_dano:" << pressaoDano << endl;

//Exibir em porcentagens
fout << "Indice_de_Produtividade:" << indiceProdutividade*100.0 << "%"
fout.close();
cout << "Dados_Salvos_com_sucesso!" << endl;
cout << "_____" << endl;
cin.get();
}

```

```

        break;
    }
}
}

void CSimuladorAnaliseTestePressao :: Exporta()
{
    // armazena a string digitada em nomeSaida

    cout << "\nInforme o nome do arquivo de saida com os parametros calculados";
    cin >> nome;
    cin . get ();
    // getline (cin , nome);
    string formato = ".dat";
    string Saida = nome+formato;
    // abre arquivo
    fout . open ( Saida . c _ str ());
}

fout << "Permeabilidade:" << buildup->getPermeabilidade();
if (( reservatorio . SistemaUnidade() == 141.2) || ( reservatorio . SistemaUnidade() == 0.3183))
fout << " milidarcy" << endl;
if ( reservatorio . SistemaUnidade() == 0.3183)
    fout << " metros quadrados" << endl; // criacao do objeto armazenado
fout << "Pressao Inicial:" << pressaoInicial;

if ( reservatorio . SistemaUnidade() == 141.2)
    fout << " psi" << endl;

if ( reservatorio . SistemaUnidade() == 19.03)
    fout << " kgf/cm2" << endl;

if ( reservatorio . SistemaUnidade() == 0.3183)
    fout << " Pascal" << endl;
    fout << "Fator de Pelicula:" << buildup->getPelicula() << endl;
    fout << "Raio Efetivo:" << raioEfetivo;

// if ((reservatorio . SistemaUnidade() == ESistemaUnidade :: SI) // (reservatorio . SistemaUnidade() == 0.3183) || (reservatorio . SistemaUnidade() == 141.2))
if (( reservatorio . SistemaUnidade() == 0.3183) || ( reservatorio . SistemaUnidade() == 141.2))
fout << " metros" << endl;
}

```

```

if ( reservatorio .SistemaUnidade() == 141.2)
    fout << " " << endl;
    fout << "Queda_de_Pressao_devido_ao_dano: " << pressaoDano << endl;

    //Exibir em porcentagens
fout << "Indice_de_Produtividade: " << indiceProdutividade * 100.0 << "%";
fout .close ();
cout << "Dados_Salvos_com_sucesso!" << endl;
cout << "____________________________________" << endl;
}

//Calcula os parametros da estocagem, se houver
void CSimuladorAnaliseTestePressao :: CalculoEstocagem ()
{
    coeficienteEstocagem = ( poco .Vazao () * fluido .FatorVolumeFormacao ()
                            / ( 24.0 * ( registrador .PressaoMedida (0) - poco .Vazao () ) ) );

    tempoEstocagem = (( 60.0 + 3.5 * buildup->getPelicula ()) / ( buildup->getPelicula ()
                            * reservatorio .SistemaUnidade () * 24.0 * coeficienteEstocagem ) );
}

cout << "Coeficiente_de_Estocagem: " << coeficienteEstocagem << endl;
cout << "Tempo_de_Estocagem: " << tempoEstocagem;

if ( ( reservatorio .SistemaUnidade () == 141.2) || ( reservatorio .SistemaUnidade () == 0.3183) )
    cout << "horas" << endl;

if ( reservatorio .SistemaUnidade () == 0.3183)
    cout << "segundos" << endl;
}

```

Apresenta-se na listagem 6.26 o programa que usas a classes listadas acima.

Listagem 6.26: Arquivo de implementação da função `main()`.

```
#include "CSimuladorAnaliseTestePressao.h"
```

```

int main ()
{
    // cria o objeto simulador da classe CSimuladorAnaliseTestePressao
    CSimuladorAnaliseTestePressao simulador;

    //executa a funcao de analise do teste de pressao
}

```

```
    simulador.Executar();  
  
    //retorna 0 se o programa rodou normalmente  
    return 0;  
}
```

# Capítulo 7

## Teste

Neste capítulo se apresenta os testes realizados para assegurar que o programa esteja funcionando corretamente.

### 7.1 Teste 1: Teste no Windows

O teste realizado no sistema operacional Windows 10 (plataforma onde foi desenvolvida a maior parte do código do programa), com auxilio do compilador 'Dev C++', teve como objetivo: Verificar se havia algum tipo de '*bug*' no programa, se ele retornava os valores corretos dos parâmetros do reservatório, e se os métodos condicionais do programa funcionariam, como uma entrada de dados errada por parte do usuário e uma nova regressão linear caso o fator de correlação não fosse satisfatório.

Primeiramente, como mostra a Figura 7.1, o programa pede a seleção do método, do sistema de unidades e a entrada dos parâmetros. Neste capítulo se apresenta os testes realizados para assegurar que o programa esteja funcionando corretamente.

```
C:\Users\User\Desktop\Andreia\TestedePressao.exe

PROGRAMA PARA CALCULO DE PARAMETROS DE RESERVATORIO POR TESTES DE PRESSAO

1-Rodar o Programa
2-Sair

1
Escolha o metodo a ser aplicado: 1 - Horner, 2 - MDH
2
Entrada de Dados do teste de pressao realizado
-----
Qual o sistema de unidades utilizado para informar os parametros:
1 - Americano (Oilfield)
2 - Brasileiro (Petrobras)
3 - Sistema Internacional
2
Informe a porosidade da rocha reservatorio:
0.16
Informe a altura do reservatorio:
-5
Reinforme a altura:
9.8
Informe o fator volume-formacao do fluido:
1.25
Informe a viscosidade do fluido:
1
Informe a compressibilidade total (fluido+rocha):
0.000147
Informe a vazao de producao:
88
Informe o tempo de producao:
25
Informe a pressao no poco:
171.90
Informe o raio do poco:
0.1
entre com valor P1h
228.30
Entrada de Dados Finalizada
-----
```

Figura 7.1: Tela do programa mostrando a entrada de dados.

Em seguida, como mostrado na Figura 7.2, o programa solicita os dados do registrador de pressão, e realiza a regressão linear, exibindo o resultado na tela.

```
Selecionar C:\Users\User\Desktop\Andreia\TestedePressao.exe
Entrada de Dados Finalizada
-----
Importacao dos dados do registrador de Pressao
-----
Informe o nome do arquivo com os dados do registrador de pressao:
DadosRegistrador-MDH.dat
Dados do registrador importados com sucesso
Regressao Linear dos Dados
-----
EQUACAO: y = 14.7233 * x + 228.3
r = -0.641288
```

Figura 7.2: Tela do programa mostrando seleção do arquivo de entrada, e posterior regressão linear e o ajuste da curva.

Após a realização da regressão linear, o usuário pode solicitar a geração do gráfico, como mostrado nas Figuras 7.3 e 7.4. Os resultados são exibidos em tela, assim como podem ser exportados para um arquivo .dat, com o nome escolhido pelo usuário, Figura 7.5.

```
C:\Users\User\Desktop\Andreia\TestedePressao.exe
r = -0.94852
Necessario novo Ajuste.
t[x] = 8
t[x] = 12
t[x] = 16
t[x] = 24
EQUACAO: y = 5.5751 * x + 228.3
r = -0.971487
Necessario novo Ajuste.
t[x] = 12
t[x] = 16
t[x] = 24
EQUACAO: y = 5.11577 * x + 228.3
r = -0.991534
Regressao linear feita com sucesso
-----
Ajuste N = 228.3
Deseja gerar o grafico:
1-Sim
2-Nao
```

Figura 7.3: Tela do programa mostrando a possibilidade de geração do gráfico.

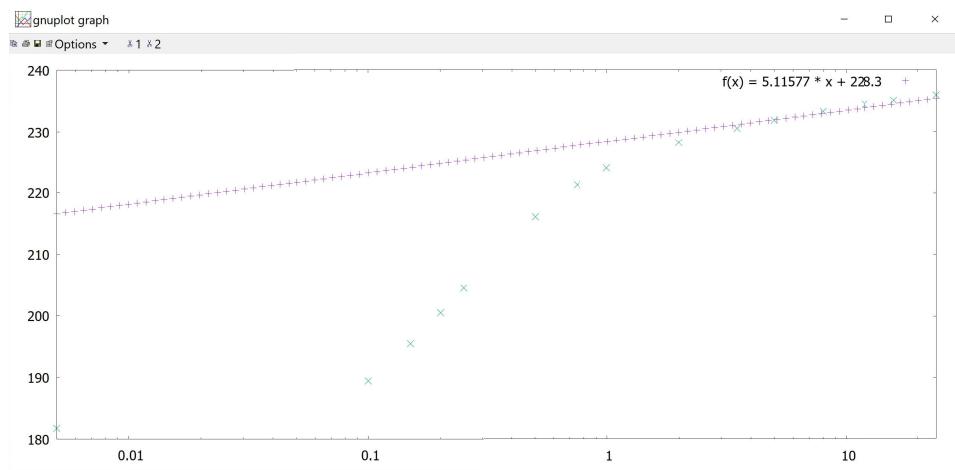


Figura 7.4: Gráfico de  $P_{ws}$  vs.  $\log \Delta t$  gerado pelo Gnuplot.

Além disso, a Figura 7.5 mostra a última etapa, onde o usuário pode realizar a variação de uma determinada variável. Para isso, deve entrar com os valores inicial e final, além da quantidade de intervalos em que deseja realizar tal variação. Assim, o programa gera o resultado para todos os valores calculados e exporta para um arquivo externo .dat automaticamente, como exibido na Figura 7.6.

```
C:\Users\User\Desktop\Andreia\TestedePressao.exe
Permeabilidade: 48.0585 milidarcy
Pressao Inicial: 228.3 kgf/cm2
Fator de Pelicula: 4.48485
Raio Efetivo: 0.00112786 metros
Queda de Pressao devido ao dano: -19.9378
Indice de Produtividade: 156.028 %
Eficiencia de Fluxo: 135.351 %

Informe o nome do arquivo de saida com os parametros calculados pelo simulador:
ResultadoMDH-ex1
Dados Salvos com sucesso!

-----
Coeficiente de Estocagem: 0.00233367
Tempo de Estocagem: 0.171306 horas
Caracterizacao do Reservatorio.

-----
1 - Reservatorio com permeabilidade boa.
2 - Reservatorio com dano baixo, nao necessita de processos de acidificacao e/ou fraturamento hidraulico.
3 - Reservatorio com produtividade boa.

-----
Deseja variar algum parametro?
1- Sim | 2 - Nao
1

Qual parametro deseja variar?
| 1 - Porosidade | 2 - Fator Volume de Formacao | 3 - Compressibilidade | 4 - Viscosidade |
1
Digite um valor inicial para porosidade
```

Figura 7.5: Tela do programa mostrando a exportação dos resultados para um arquivo de saída .dat, a caracterização do reservatório, e a possibilidade de se variar um parâmetro.

Todos os arquivos exportados são salvos no diretório onde se encontram as listagens. Como é destacado na Figura 7.6, os arquivos tem o nome dado pelo usuário, e na realização da variação, o nome é automaticamente composto pelo nome escolhido pelo usuário, mais o nome do parâmetro variado, e o valor de tal parâmetro.

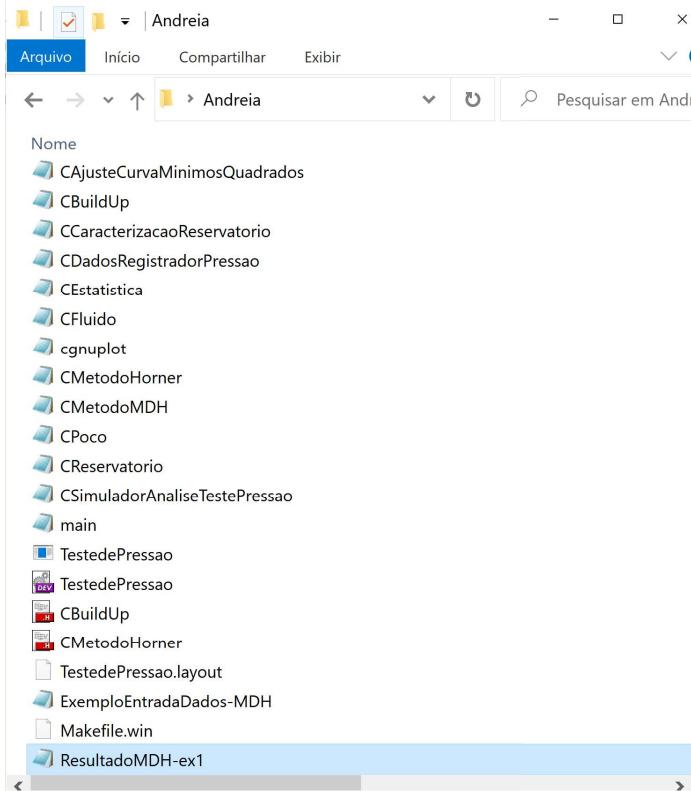
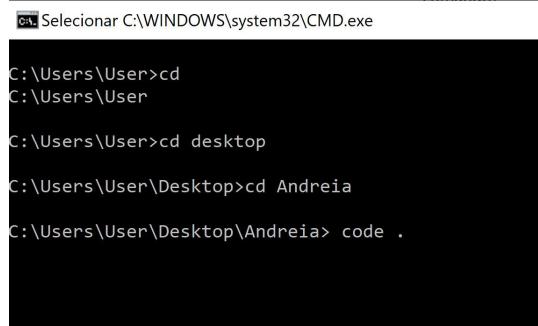


Figura 7.6: Diretório onde se encontram as listagens e onde são salvos os arquivos de saída (em destaque).

## 7.2 Teste 2: Teste no GNU/Linux (Gerando o gráfico com o Gnuplot)

O teste realizado com auxílio do editor Visual Studio Code e dos comandos do terminal. Figura 7.7 mostra a abertura desse compilador pelo prompt de comando na pasta em que se encontra o código do programa. A entrada de dados do fluido, do poço e da rocha é mostrada na Figura 7.8. Os próximos passos, mostrados nas Figuras 7.9 e 7.10 foram idênticos aos feitos no teste no Windows. Na Figura 7.11 tem a escolha do nome do arquivo de saída com seus dados do resultado salvos. Nesse caso do exemplo nomeado como ResultadoMDH-ex1.

Testando agora para um exemplo do programa calculando pelo método de Horner. A Figura 7.12 mostra a escolha do método Horner e entrada dos dados. Em sequência, a Figura 7.13 mostra a entrada dos dados do registrador de pressão e a escolha do gráfico, que será mostrado na Figura 7.14. E por fim, escolha do nome do arquivo de saída e salvando os dados, Figura 7.15.



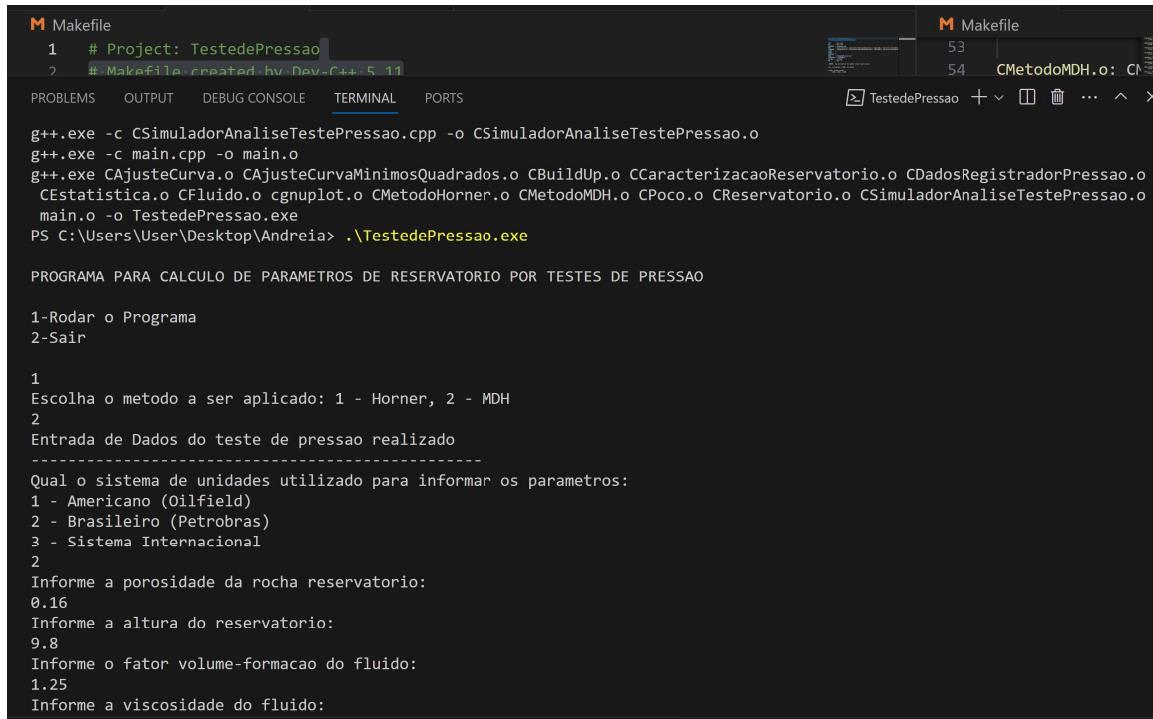
```
C:\Users\User>cd
C:\Users\User

C:\Users\User>cd desktop

C:\Users\User\Desktop>cd Andreia

C:\Users\User\Desktop\Andreia> code .
```

Figura 7.7: Tela do programa mostrando abertura do compilador Visual Code.



```
1 # Project: TestedePressao
2 # Makefile created by Dev-C++ 5.11
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
g++.exe -c CSimuladorAnaliseTestePressao.cpp -o CSimuladorAnaliseTestePressao.o
g++.exe -c main.cpp -o main.o
g++.exe CAjusteCurva.o CAjusteCurvaMinimosQuadrados.o CBuildUp.o CCaracterizacaoReservatorio.o CDadosRegistradorPressao.o
CEstatistica.o CFluido.o cgnuplot.o C MetodoHorner.o C MetodoMDH.o CPoco.o CReservatorio.o CSimuladorAnaliseTestePressao.o
main.o -o TestedePressao.exe
PS C:\Users\User\Desktop\Andreia> .\TestedePressao.exe

PROGRAMA PARA CALCULO DE PARAMETROS DE RESERVATORIO POR TESTES DE PRESSAO

1-Rodar o Programa
2-Sair

1
Escolha o metodo a ser aplicado: 1 - Horner, 2 - MDH
2
Entrada de Dados do teste de pressao realizado
-----
Qual o sistema de unidades utilizado para informar os parametros:
1 - Americano (Oilfield)
2 - Brasileiro (Petrobras)
3 - Sistema Internacional
2
Informe a porosidade da rocha reservatorio:
0.16
Informe a altura do reservatorio:
9.8
Informe o fator volume-formacao do fluido:
1.25
Informe a viscosidade do fluido:
```

Figura 7.8: Tela do programa mostrando o Visual Code compilando o make do programa, com as entradas dos dados.

```

Makefile.win      M Makefile      Untitled-1
M Makefile
1 # Project: TestedePressao
2 # Makefile created by Dev-C++ 5.11

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
Terminal: TestedePressao

Informe o raio do poco:
0.1
entre com valor P1h
228.30
Entrada de Dados Finalizada
-----
Importacao dos dados do registrador de Pressao
-----

Informe o nome do arquivo com os dados do registrador de pressao:
DadosRegistrador-MDH.dat
Dados do registrador importados com sucesso

Regressao Linear dos Dados
-----

EQUACAO: y = 14.7233 * x + 228.3
r = -0.641288

Localizando o Periodo Transiente
-----

Necessario novo Ajuste.
t[x] = 0.1
t[x] = 0.15
t[x] = 0.2
t[x] = 0.25
t[x] = 0.5

```

In 1, Col 26 / 36 selected

Figura 7.9: Tela do programa mostrando a leitura dos dados do Registrador de pressão.

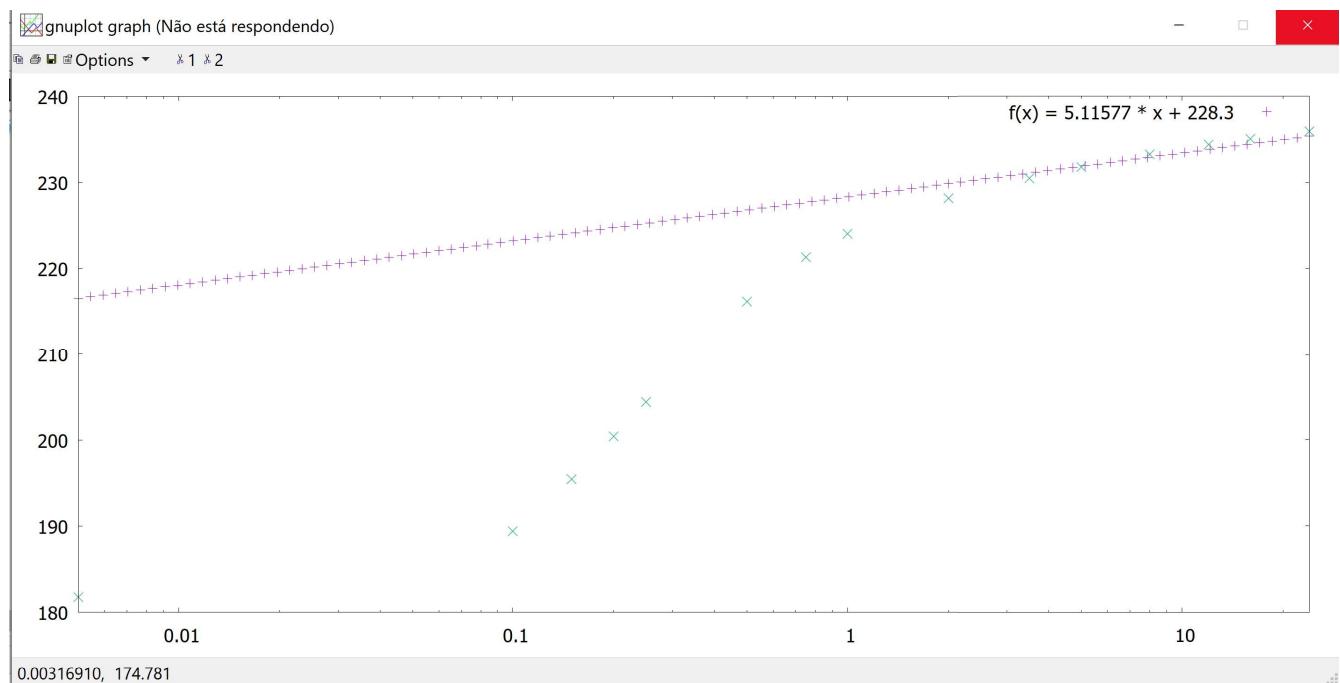


Figura 7.10: Gráfico de  $P_{ws}$  vs.  $\log \Delta t$  gerado pelo Gnuplot.

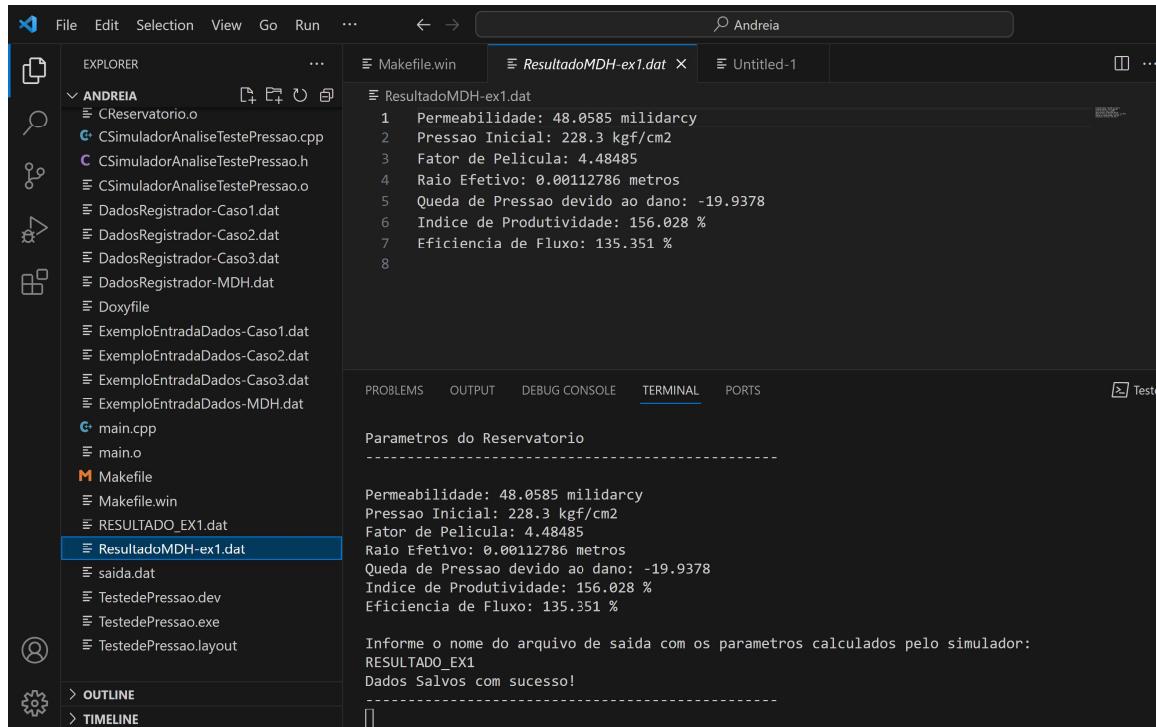


Figura 7.11: Tela do programa mostrando o nome do arquivo de saída a ser salvo, onde ele está e o seu resultado.

```

PROGRAMA PARA CALCULO DE PARAMETROS DE RESERVATORIO POR TESTES DE PRESSAO

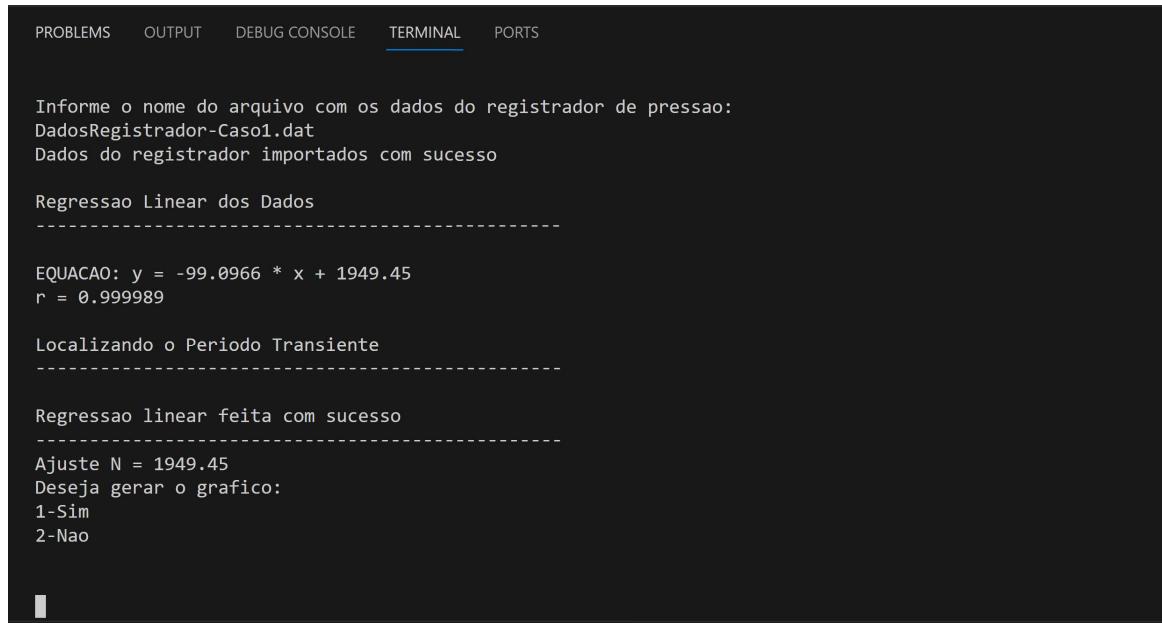
1-Rodar o Programa
2-Sair

1
Escolha o metodo a ser aplicado: 1 - Horner, 2 - MDH
1
Entrada de Dados do teste de pressao realizado
-----
Qual o sistema de unidades utilizado para informar os parametros:
1 - Americano (Oilfield)
2 - Brasileiro (Petrobras)
3 - Sistema Internacional
1
Informe a porosidade da rocha reservatorio:
0.2
Informe a altura do reservatorio:
22.0
Informe o fator volume-formacao do fluido:
1.3
Informe a viscosidade do fluido:
1.0

```

Ln 69 Col 1 Tab Size: 4

Figura 7.12: Tela do programa mostrando escolha método Horner e entrada de dados.



```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

Informe o nome do arquivo com os dados do registrador de pressao:
DadosRegistrador-Caso1.dat
Dados do registrador importados com sucesso

Regressao Linear dos Dados
-----
EQUACAO: y = -99.0966 * x + 1949.45
r = 0.999989

Localizando o Periodo Transiente
-----
Regressao linear feita com sucesso
-----
Ajuste N = 1949.45
Deseja gerar o grafico:
1-Sim
2-Nao

```

Figura 7.13: Entrada de dados do Registrador de pressão para o caso de Horner e a escolha do gráfico.

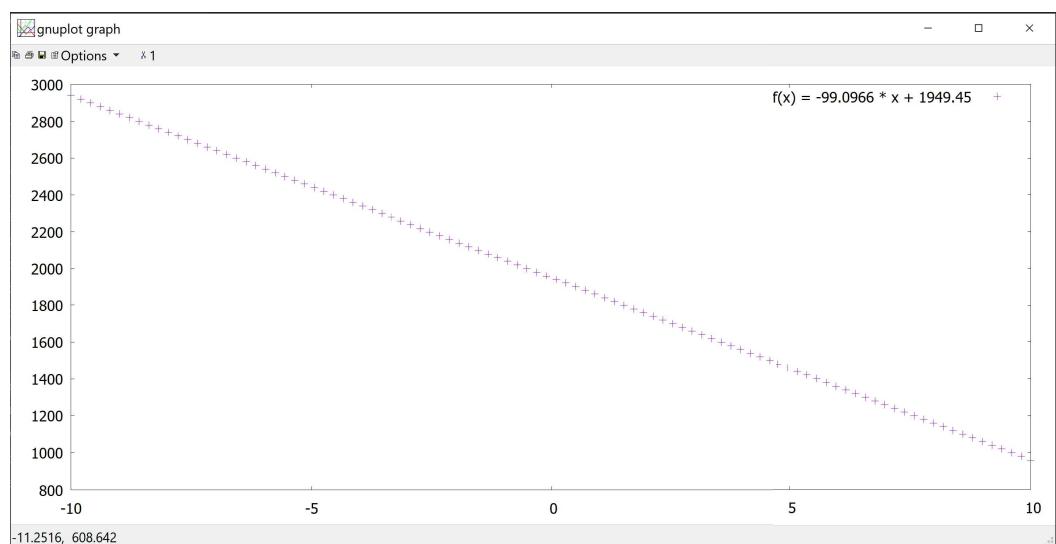


Figura 7.14: Gráfico de  $P_{ws}$  vs.  $\log[(t + \Delta t)/\Delta t]$  gerado pelo Gnuplot.

```
Deseja gerar o grafico:  
1-Sim  
2-Nao  
  
1  
Parametros do Reservatorio  
-----  
Permeabilidade: 48.4554 milidarcy  
Pressao Inicial: 1949.45 psi  
Fator de Pelicula: -2.94305  
Raio Efetivo: 5.69207 ft  
Queda de Pressao devido ao dano: -253.44  
Indice de Produtividade: 62.5433 %  
Eficiencia de Fluxo: 131.702 %  
  
Informe o nome do arquivo de saida com os parametros calculados pelo simulador:  
ResultadoCas01  
Dados Salvos com sucesso!  
-----
```

Figura 7.15: Tela do programa mostrando o nome do arquivo de saída a ser salvo.

# Capítulo 8

## Documentação para o Desenvolvedor

A presente documentação refere-se ao uso do "Programa em C++ para avaliação de formações por dados de teste de pressão". Esta documentação tem o formato de uma apostila que explica passo a passo ao usuário como usar o programa.

### 8.1 Documentação para desenvolvedor

Apresenta-se nesta seção a documentação para desenvolvedor, contendo assim as informações para usuários que queiram modificar ou ampliar este software.

#### 8.1.1 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 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`.

#### 8.1.2 Documentação usando doxygen

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

## 8.2 Sugestões para trabalhos futuros

- O desenvolvedor pode acrescentar outro método de teste de pressão para avaliação de formações ao programa. Pela classe enumeração Método ele já consegue incluir na sequência para os métodos de Horner e MDH que já existem no programa.
- Otimizar o código.
- Realizar comparação com os valores encontrados para cada método.

## 8.3 Como gerar a documentação usando doxygen

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

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

Passos para gerar a documentação usando o `doxygen`.

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

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

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

```
doxygen -g
```

- Peça para o `doxygen` gerar a documentação.

doxygen

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

firefox html/index.html

ou

chrome html/index.html

A Figura 8.1 apresenta a listagem de classes gerado pelo doxygen.

The screenshot shows a web-based doxygen class list. At the top, there's a header with the project name 'Projeto\_engenhariaAndreia\_2' and a subtitle 'Avaliação de Formações - Teste de pressão - MDH'. Below the header, there are navigation links for 'Página Principal', 'Classes', and 'Arquivos', along with a search bar. The main content area is titled 'Lista de Classes' and contains a table with 15 entries, each showing a class name with a small icon and its brief description. The classes listed are: CAjusteCurva, CAjusteCurvaMimosQuadrados, CBuildUP, CCaracterizacaoReservatorio, CDadosRegistradorPressao, CEstatistica, CFluido, CMetodoHorner, CMetodoMDH, CPoco, CReservatorio, CSimuladorAnaliseTestePressao, and Gnuplot.

| Lista de Classes  |  |
|---|--|
| Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições: |  |
| <a href="#">CAjusteCurva</a>  | Classe que executa a regressao linear e ajusta ate a correlacao satisfatoria                             |
| <a href="#">CAjusteCurvaMimosQuadrados</a>  | Classe que obtém os coeficiente da regressao linear por meio do metodo dos minimos quadrados             |
| <a href="#">CBuildUP</a>  |  |
| <a href="#">CCaracterizacaoReservatorio</a>   | Classe que caracteriza o reservatorio  |
| <a href="#">CDadosRegistradorPressao</a>  | Classe que contém dados registrados do registrador de pressao  |
| <a href="#">CEstatistica</a>  | Classe que calcula estatisticas do vetor, como media e desvio padrao, util para regressao                |
| <a href="#">CFluido</a>   | Classe contendo as características do fluido produzido   |
| <a href="#">CMetodoHorner</a>   |  |
| <a href="#">CMetodoMDH</a>  |  |
| <a href="#">CPoco</a>   | Classe contendo as características do poço   |
| <a href="#">CReservatorio</a>   | Classe contendo as características do reservatorio   |
| <a href="#">CSimuladorAnaliseTestePressao</a>   | Classe que faz a analise do teste de pressao realizado no campo e infere as propriedades do reservatorio |
| <a href="#">Gnuplot</a>   | Classe de interface para acesso ao programa gnuplot  |

Figura 8.1: Lista de classes pelo doxygen.

## Referências

---

# Referências Bibliográficas

- [Bueno, 2003] Bueno, A. D. (2003). *Programação Orientada a Objeto com C++ - Aprenda a Programar em Ambiente Multiplataforma com Software Livre*. Novatec, São Paulo.
- [Bueno, 2022] Bueno, A. D. (2022). *Programação Orientada a Objeto com C++ - Aprenda a Programar em Ambiente Multiplataforma com Software Livre*. o autor, Macaé.
- [Chaudhry, 2004] Chaudhry, A. (2004). *Oil well testing handbook*. Elsevier.
- [Grossens et al., 1993] Grossens, M., Mittelbach, F., and Samarin, A. (1993). *Latex Companion*. Addison-Wesley, New York.
- [Karger, 2004] Karger, A. (2004). *O Tutorial de Lyx*. LyX Team - <http://www.lyx.org>.
- [Lee, 1982] Lee, J. (1982). *Well Testing*. SPE, New York.
- [Álvaro M. M. Peres, 2008] Álvaro M. M. Peres (2008). Teoria dos testes de pressão em poços. Rio de Janeiro. Anais do VI Encontro ENGEP.
- [LyX-Team, 2004] LyX-Team, editor (2004). *The LyX User's Guide*. LyX Team - <http://www.lyx.org>.
- [Ortiz, 2014] Ortiz, C. (2014). Avaliação de formações- testes de pressão em poços. LENEP-UENF.
- [Rosa et al., 2006] Rosa, A. J., de Souza Carvalho, R., and Xavier, J. A. D. (2006). *Engenharia de reservatórios de petróleo*. Interciênciac.
- [Rosa and Correa, 1987] Rosa, J. and Correa, A. (1987). Análise de testes de pressão em poços. *Apostila Petrobras-março*.
- [Steding-Jessen, 2000] Steding-Jessen, K. (2000). *Latex demo: Exemplo com Latex 2e*.

# Índice Remissivo

## A

Análise orientada a objeto, 20  
AOO, 20  
Associações, 33  
atributos, 33

## C

Casos de uso, 5  
Ciclo construção, 36  
colaboração, 28  
comunicação, 28  
Concepção, 4  
Controle, 32

## D

Diagrama de colaboração, 28  
Diagrama de componentes, 34  
Diagrama de execução, 34  
Diagrama de máquina de estado, 29  
Diagrama de sequência, 27

## E

Efeitos do projeto nas associações, 33  
Efeitos do projeto nas heranças, 33  
Efeitos do projeto nos métodos, 33  
Elaboração, 9  
Especificação, 4  
especificação, 4  
estado, 29  
Eventos, 27

## H

Heranças, 33  
heranças, 33

## I

Implementação, 36

## M

Mensagens, 27  
Metodologia utilizada, 2  
métodos, 33  
modelo, 33

## P

Plataformas, 32  
POO, 32  
Projeto do sistema, 31  
Projeto orientado a objeto, 32  
Protocolos, 31

## R

Recursos, 32  
Requisitos, 5  
Requisitos funcionais, 5  
Requisitos não funcionais, 5