

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

1	Introdução	1
1.1	Escopo do problema	1
1.2	Objetivos	2
1.3	Metodologia utilizada	2
2	Concepção	4
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
3	Elaboração	9
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
4	AOO – Análise Orientada a Objeto	20
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

5 Projeto	31
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
6 Ciclos Construção - Implementação	36
6.1 Código fonte	36
7 Teste	93
7.1 Teste 1: Teste no Windows	93
7.2 Teste 2: Teste no GNU/Linux (Gerando o gráfico com o Gnuplot)	97
8 Documentação para o Desenvolvedor	103
8.1 Documentação para desenvolvedor	103
8.1.1 Dependências	103
8.1.2 Documentação usando doxygen	103
8.2 Sugestões para trabalhos futuros	104
8.3 Como gerar a documentação usando doxygen	104
Referências Bibliográficas	106

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 -Escolhendo o método e variando um parâmetro de entrada, sendo salvo 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.	94
7.2	Tela do programa mostrando seleção do arquivo de entrada, e posterior regressão linear e o ajuste da curva.	95
7.3	Tela do programa mostrando a possibilidade de geração do gráfico.	95
7.4	Gráfico de Pws vs. $\log\Delta t$ gerado pelo Gnuplot.	96

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.	96
7.6	Diretório onde se encontram as listagens e onde são salvos os arquivos de saída (em destaque).	97
7.7	Tela do programa mostrando abertura do compilador Visual Code.	98
7.8	Tela do programa mostrando o Visual Code compilando o make do programa, com as entradas dos dados.	98
7.9	Tela do programa mostrando a leitura dos dados do Registrador de pressão.	99
7.10	Gráfico de P_{ws} vs. $\log\Delta t$ gerado pelo Gnuplot.	99
7.11	Tela do programa mostrando o nome do arquivo de saída a ser salvo, onde ele está e o seu resultado.	100
7.12	Tela do programa mostrando escolha método Horner e entrada de dados.	100
7.13	Entrada de dados do Registrador de pressão para o caso de Horner e a escolha do gráfico.	101
7.14	Gráfico de P_{ws} vs. $\log[(t + \Delta t)/\Delta t]$ gerado pelo Gnuplot.	101
7.15	Tela do programa mostrando o nome do arquivo de saída a ser salvo.	102
8.1	Lista de classes pelo doxygen.	105

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.	48
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.	63
6.17	Arquivo de cabeçalho da classe CMetodo.h.	63
6.18	Arquivo de cabeçalho da classe CMetodoMDH.h.	64
6.19	Arquivo de implementação da classe CMetodoMDH.cpp.	64
6.20	Arquivo de cabeçalho da classe CMetodoHorner.h.	66
6.21	Arquivo de implementação da classe CMetodoHorner.cpp.	66
6.22	Arquivo de cabeçalho da classe CCaracterizacaoReservatorio.h.	67
6.23	Arquivo de implementação da classe CCaracterizacaoReservatorio.cpp. . .	68
6.24	Arquivo de cabeçalho da classe CSimuladorAnaliseTestePressao.h.	69
6.25	Arquivo de implementação da classe CSimuladorAnaliseTestePressao.cpp. .	72
6.26	Arquivo de implementação da função main().	91

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:

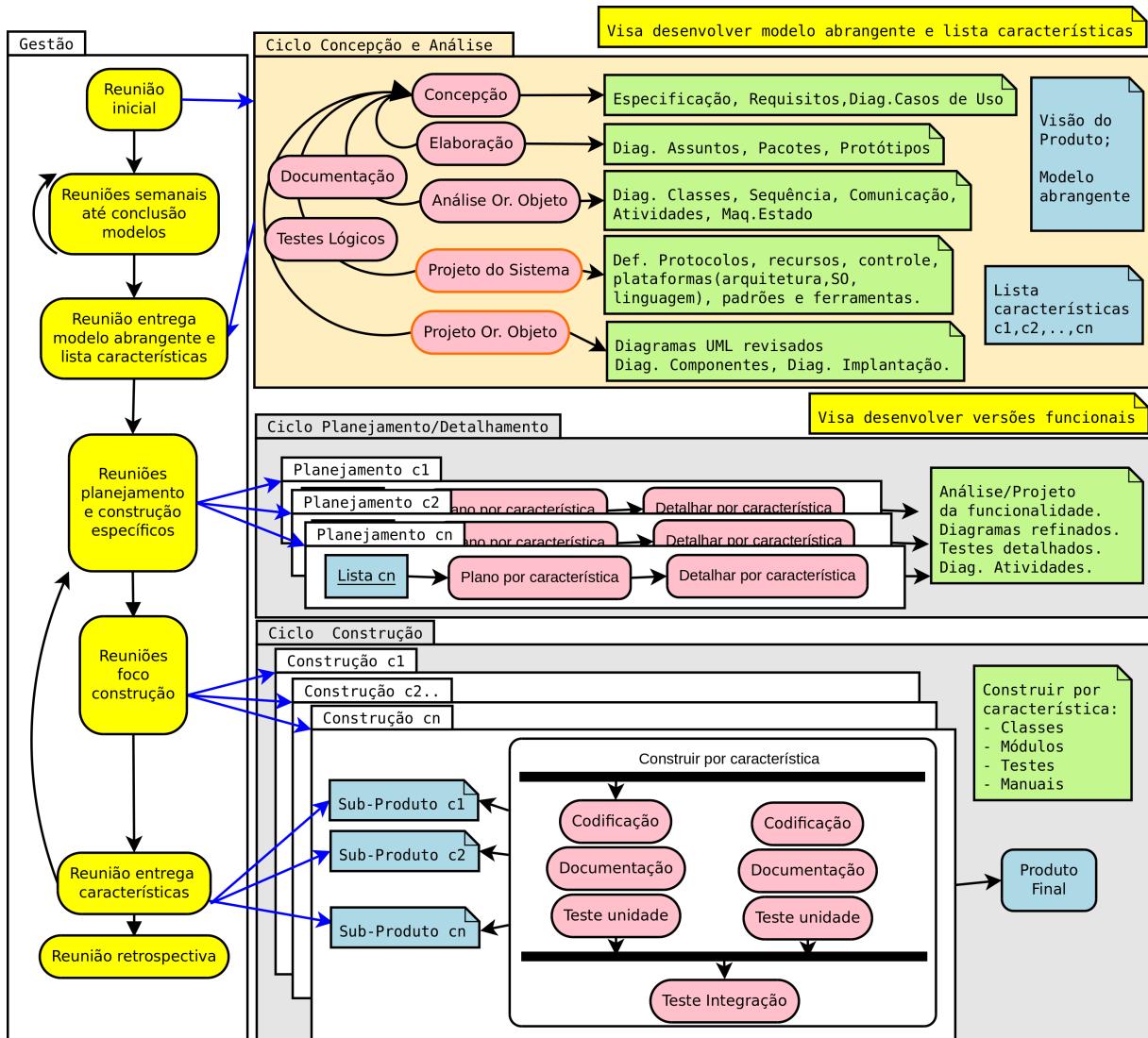
- Objetivo geral:
 - Criar um software capaz de fornecer ao usuário, através da análise dos dados obtidos em testes de pressão em poços utilizando método de Horner ou método MDH, possibilitando estimar as dimensões do campo e sua potencialidade econômica.
- Objetivos específicos:
 - Calcular a permeabilidade 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 reservatorio, 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.

RF-01	O sistema deverá conter uma base de dados referentes a um teste de crescimento de pressão.
RF-02	O usuário deverá informar o nome do arquivo texto de onde serão lidas as informações do teste de pressão
RF-03	O usuário deverá entrar com os parâmetros do fluido, do reservatório e do poço.
RF-04	O usuário deverá entrar com os parâmetros do fluido, do reservatório e do poço.
RF-05	O usuário deverá ter liberdade para escolher o método de Horner ou MDH.
RF-06	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

RNF-01	Os cálculos devem ser feitos utilizando-se fórmulas provenientes da disciplina de Avaliação de Formações.
RNF-02	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"> 1. Importar dados do teste de pressão de arquivo texto. 2. Fornecer dados do sistema poço-reservatório. 3. Selecionar o método de Horner ou método de MDH. 4. Gerar gráficos, conforme o método escolhido. 5. Exibir resultados na tela exportar para um arquivo .dat. 6. Caracterizar o reservatório também com a escolha do método MDH. 7. Selecionar uma variável e definir um intervalo de variação para a mesma. 8. Gerar novos resultados e analisar a interferência da variação nestes.
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 logarítmico 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 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.

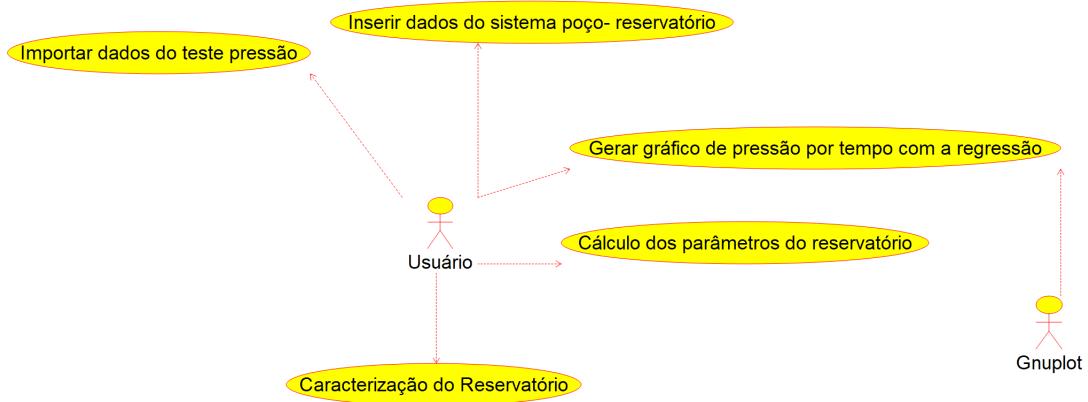


Figura 2.1: Diagrama de caso de uso – Caso de uso geral



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 de parâmetro.

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 solução 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.

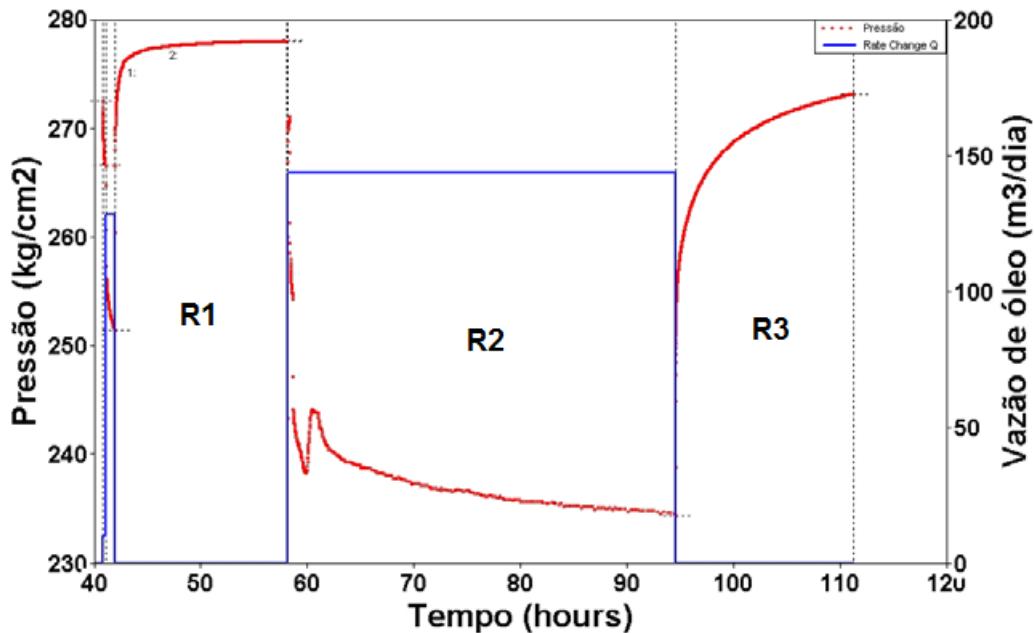


Figure 3.1: Exemplo de teste de poço convencional, onde é possível observar o comportamento da pressão em função da variação da vazão de produção. As regiões 1 e 3 mostram etapas com o poço fechado ($q=0$), onde é possível ver um aumento da pressão no reservatório. Já na região 2, o poço está produzindo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

Os testes 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.

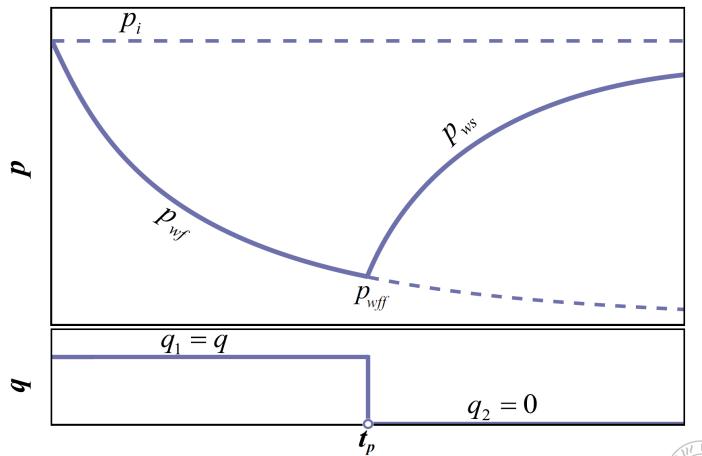


Figura 3.2: Esquema de vazão e comportamento da pressão em um poço submetido a um teste de pressão

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).

Δt = 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.

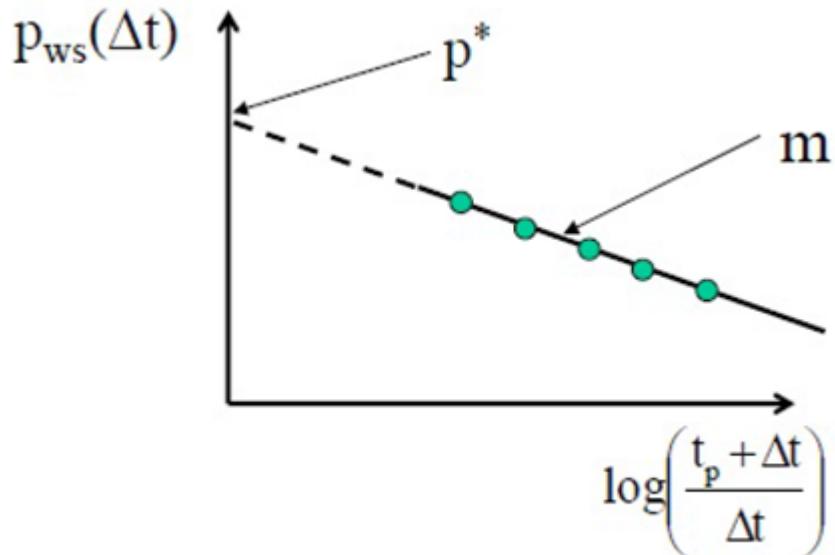


Figura 3.3: Gráfico semi-logarítmico obtido pelo Método de Horner, que fornece parâmetros referentes ao reservatório

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, μ [cp] a viscosidade do fluido, e m [adimensional] é o coeficiente angular.

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

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, ΔP_{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.

$$S = \frac{1.151(m \log(t_p + 1) + n - P_{wf})}{(-m - \log(k/(ct r_w^2)) + 3.23)} \quad (3.3)$$

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

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.

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

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

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 $t_p >> \Delta t$ 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:

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

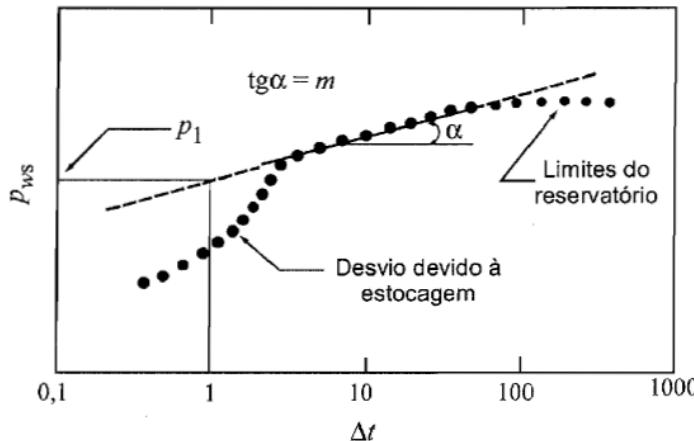


Figura 3.4: Gráfico de MDH

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

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

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 [Lee, 1982], [Chaudhry, 2004], [Rosa and Correa, 1987], [Rosa, 2006],[Ortiz, 2014].

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.

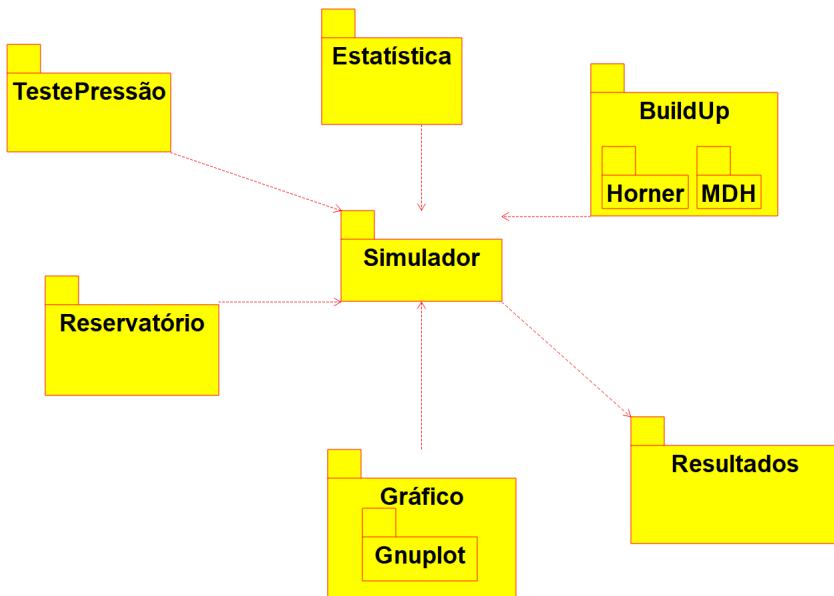


Figura 3.5: Diagrama de Pacotes

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.

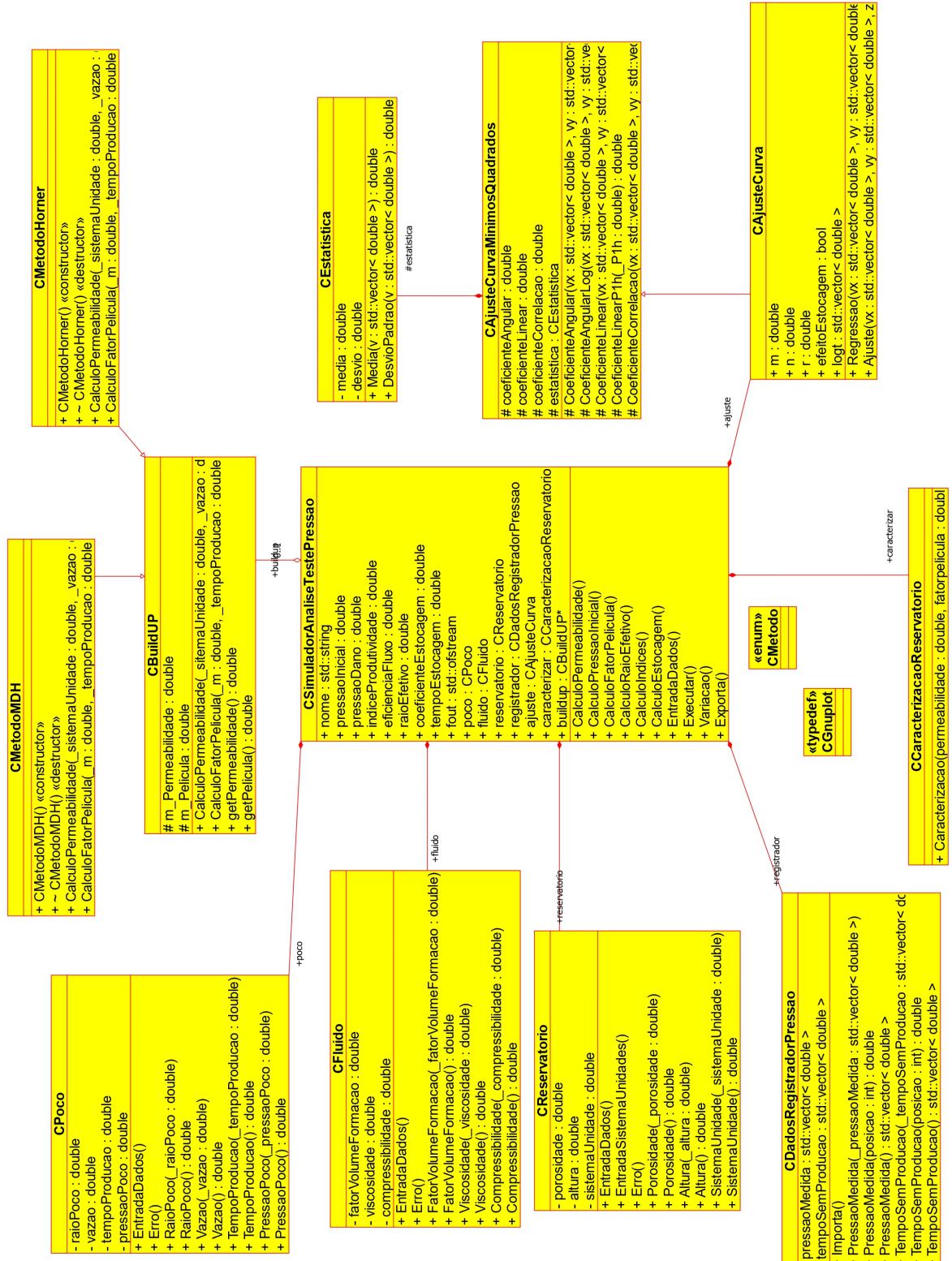


Figura 4.1: Diagrama de classes

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 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 CalculoFatorPelícula (): 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, fatorPelícula, 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.

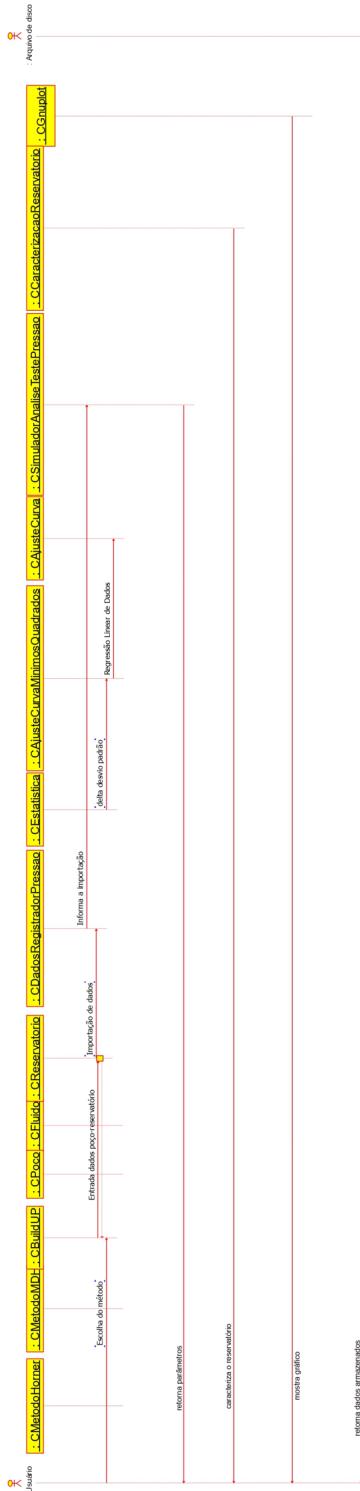


Figura 4.2: Diagrama de sequência

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 CAjusteCurvaMínimosQuadrados 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 CCaracterizaçãoReservatorio caracteriza o reservatório com a função Resultados (permeabilidade, fatorPelícula) Além disso, a classe CGnuplot permite gerar os gráficos da regressão e salvá-los.

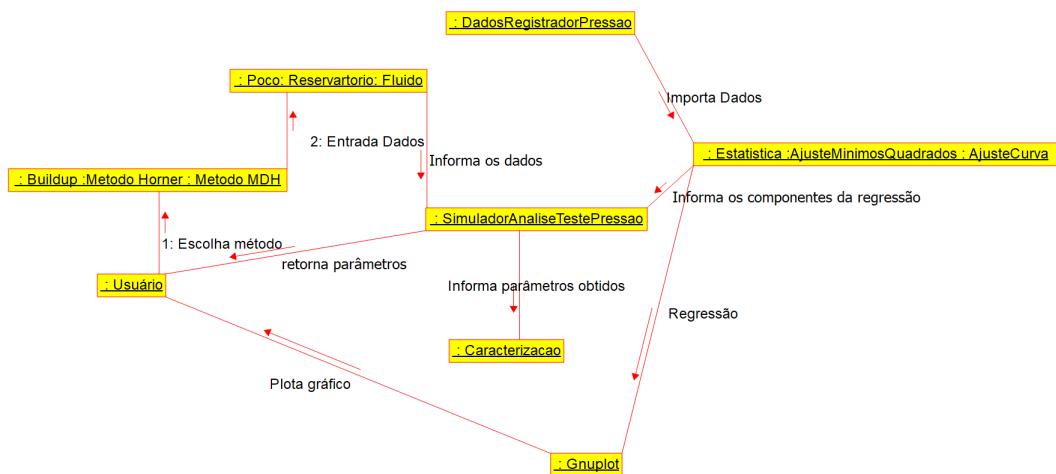


Figura 4.3: Diagrama de comunicação

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 CCaracterizaçãoReservatorio.

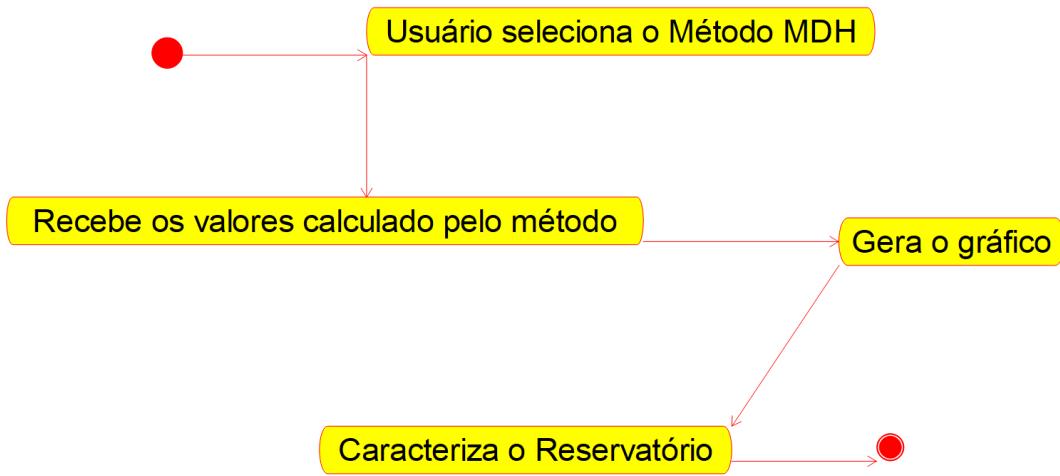


Figura 4.4: Diagrama de máquina de estado.

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.

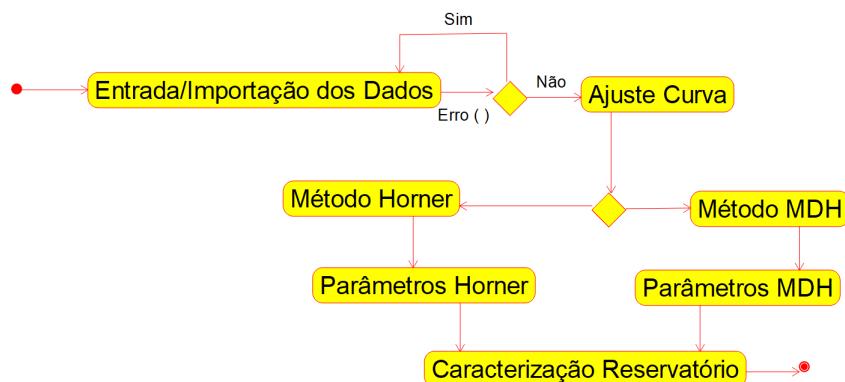


Figura 4.5: Diagrama de atividades da classe CCaracterizacaoReservatorio.

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.

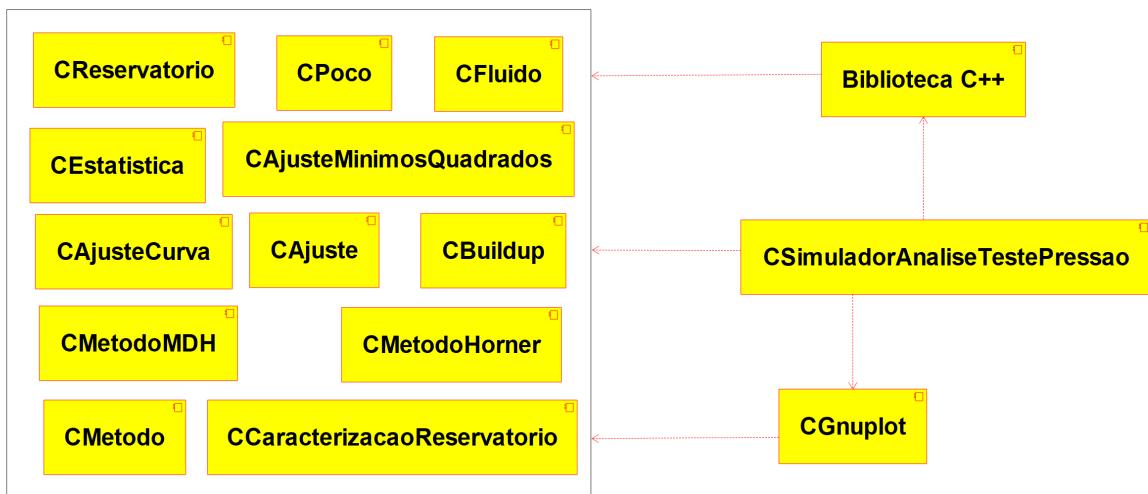


Figura 5.1: Diagrama de componentes

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.

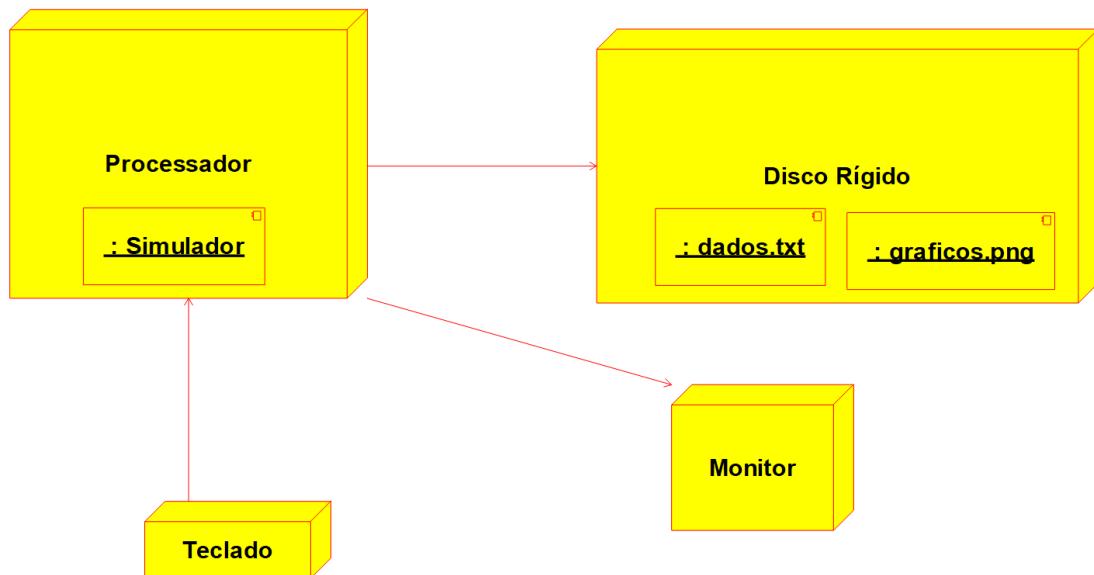


Figura 5.2: Diagrama de implantação

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`

```
1///Condicao para nao definir a classe mais de uma vez
2#ifndef CPoco_h
3#define CPoco_h
4
5class CPoco;
6
7///Classe contendo as caracteristicas do poco
8class CPoco
9{
10    ///privados, so acessados por meio de funcoes get
11    private:
12
13    ///raio do poco
14    double raioPoco;
15    ///vazao de producao
16    double vazao;
17    ///tempo de producao
18    double tempoProducao;
19    ///pressao no poco
20    double pressaoPoco;
```

```
21
22     public:
23
24         ///Funcao que recebe dados do usuario, preenchendo
25             atributos raiopoco, vazao, tempoproducao, pressaopoco
26         void EntradaDados();
27
28         ///Funcao que verifica se houve erro na entrada de dados e
29             pede nova entrada ate nao ocorrer erro
30         void Erro();
31
32         ///Funcao que seta o raiopoco
33         void RaioPoco(double _raioPoco);
34         ///Funcao get do raiopoco
35         double RaioPoco() const;
36
37         ///Funcao que seta a vazao
38         void Vazao(double _vazao);
39         ///Funcao get da vazao
40         double Vazao() const;
41
42         ///Funcao que seta o tempoproducao
43         void TempoProducao(double _tempoProducao);
44         ///Funcao get do tempoproducao
45         double TempoProducao() const;
46
47         ///Funcao que seta a pressaopoco
48         void PressaoPoco(double _pressaoPoco);
49         ///Funcao get da pressaopoco
50         double PressaoPoco() const;
51};

51#endif
```

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

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

```
1 #include "CPoco.h"
2
3 // inclui a biblioteca iostream pois usa funcoes de entrada e saida
4     de dados para a tela
4 #include <iostream>
5
```

```
6 //usa funcoes pertencentes ao namespace std
7 using namespace std;
8
9 //funcao de entrada de dados da classe
10 void CPoco::EntradaDados()
11 {
12     cout << "Informe a vazao de producao:" << endl;
13     cin >> vazao;
14     cin.get();
15
16     cout << "Informe o tempo de producao:" << endl;
17     cin >> tempoProducao;
18     cin.get();
19
20     cout << "Informe a pressao no poco:" << endl;
21     cin >> pressaoPoco;
22     cin.get();
23
24     cout << "Informe o raio do poco:" << endl;
25     cin >> raioPoco;
26     cin.get();
27 }
28
29 //funcao que acusa e conserta erro de entrada
30 void CPoco::Erro()
31 {
32
33     //repete a entrada enquanto o valor for equivocado
34
35     while (tempoProducao<0.00)
36     {
37         cout << "Reinforme o tempo de producao:" << endl;
38         cin >> tempoProducao;
39         cin.get();
40
41     }
42
43     while (pressaoPoco<0.00)
44     {
45         cout << "Reinforme a pressao no poco:" << endl;
46         cin >> pressaoPoco;
47         cin.get();
```

```
48
49  }
50  while ((raioPoco<0.00) || (raioPoco>3.0))
51  {
52      cout << "Reinforme o raiô do pôco:" << endl;
53      cin >> raioPoco;
54      cin.get();
55  }
56 }
57
58 // set
59 void CPoco::Vazao(double _vazao)
60 {
61     vazao = _vazao;
62 }
63
64 // get
65 double CPoco::Vazao() const
66 {
67     return vazao;
68 }
69
70 // set
71 void CPoco::TempoProducao(double _tempoProducao)
72 {
73     tempoProducao = _tempoProducao;
74 }
75
76 // get
77 double CPoco::TempoProducao() const
78 {
79     return tempoProducao;
80 }
81
82 // set
83 void CPoco::PressaoPoco(double _pressaoPoco)
84 {
85     pressaoPoco = _pressaoPoco;
86 }
87
88 // get
89 double CPoco::PressaoPoco() const
```

```

90 {
91     return pressaoPoco;
92 }
93
94 //set
95 void CPoco::RaioPoco(double _raioPoco)
96 {
97     raioPoco = _raioPoco;
98 }
99
100 //get
101 double CPoco::RaioPoco() const
102 {
103     return raioPoco;
104 }

```

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

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

```

1 ///Condicao para nao definir a classe mais de uma vez
2 #ifndef CReservatorio_h
3 #define CReservatorio_h
4
5 class CReservatorio;
6
7 ///Classe contendo as caracteristicas do reservatorio
8 class CReservatorio
9 {
10     //privados, so acessados por meio de funcoes get
11     private:
12
13         ///porosidade do reservatorio
14         double porosidade;
15         ///altura do reservatorio
16         double altura;
17         ///sistema de unidades utilizado do reservatorio
18         double sistemaUnidade;
19
20     public:
21
22         ///Funcao que recebe dados do usuario, preenchendo
23             atributos porosidade e altura
24         void EntradaDados();

```

```

24
25     ///Funcao que exibe um menu e recebe dado do usuario , esse
26     /// dado preenche o atributo sistemaunidade
27     void EntradaSistemaUnidades();
28
29     ///Funcao que verifica se houve erro na entrada de dados e
30     /// pede nova entrada ate nao ocorrer erro
31     void Erro();
32
33     ///Funcao que seta a porosidade
34     void Porosidade(double _porosidade);
35     ///Funcao get da porosidade
36     double Porosidade() const;
37
38     ///Funcao que seta a altura
39     void Altura(double _altura);
40     ///Funcao get da porosidade
41     double Altura() const;
42
43     ///Funcao que seta o sistemaunidade
44     void SistemaUnidade(double _sistemaUnidade);
45     ///Funcao get do sistemaunidade
46     double SistemaUnidade() const;
47};

#endif

```

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

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

```

1 #include "CReservatorio.h"
2
3 // inclui a biblioteca iostrem pois usa funcoes de entrada e saida
4 // de dados para a tela
4 #include <iostream>
5
6 // usa funcoes pertencentes ao namespace std
7 using namespace std;
8
9 //funcao de entrada de dados da classe
10 void CReservatorio::EntradaDados()
11 {
12

```

```
13 cout << "Informe a porosidade da rocha reservatorio:" << endl;
14 cin >> porosidade;
15 cin.get();
16
17 cout << "Informe a altura do reservatorio:" << endl;
18 cin >> altura;
19 cin.get();
20
21 }
22
23 //Funcao que preenche o sistemaunidade por um pequeno menu
24 void CReservatorio::EntradaSistemaUnidades()
25 {
26     cout << "Qual o sistema de unidades utilizado para informar os
27         parametros:" << endl <<
28         "1 - Americano (Oilfield)" << endl << "2 - Brasileiro (
29             Petrobras)" << endl <<
30         "3 - Sistema Internacional" << endl;
31     int i;
32     cin >> i;
33     cin.get();
34
35     //repete a entrada enquanto o valor for equivocado
36     while ((i!=1)&&(i!=2)&&(i!=3))
37     {
38         cout << "Reinforme o sistema de unidades utilizado." <<
39             endl;
40         cin>>i;
41         cin.get();
42     }
43
44     if (i==1)
45         sistemaUnidade = 141.2;
46
47     if (i==2)
48         sistemaUnidade = 19.03;
49
50     if (i==3)
51         sistemaUnidade = 0.3183;
52 }
53
54 //funcao que acusa e conserta erro de entrada
```

```
52 void CReservatorio::Erro()
53 {
54     //repete a entrada enquanto o valor for equivocado
55     while (altura<0.00)
56     {
57         cout << "Reinforme a altura:" << endl;
58         cin >> altura;
59         cin.get();
60     }
61
62     while ((porosidade<0.00) || (porosidade>1.00))
63     {
64         cout << "Reinforme a porosidade:" << endl;
65         cin >> porosidade;
66         cin.get();
67     }
68
69 }
70
71 //set
72 void CReservatorio::Porosidade(double _porosidade)
73 {
74     porosidade = _porosidade;
75 }
76
77 //get
78 double CReservatorio::Porosidade() const
79 {
80     return porosidade;
81 }
82
83 //set
84 void CReservatorio::Altura(double _altura)
85 {
86     altura = _altura;
87 }
88
89 //get
90 double CReservatorio::Altura() const
91 {
92     return altura;
93 }
```

```
94
95 //set
96 void CReservatorio::SistemaUnidade(double _sistemaUnidade)
97 {
98     sistemaUnidade = _sistemaUnidade;
99 }
100
101 //get
102 double CReservatorio::SistemaUnidade() const
103 {
104     return sistemaUnidade;
105 }
```

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

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

```
1 ///Condicao para nao definir a classe mais de uma vez
2 #ifndef CFluido_h
3 #define CFluido_h
4
5 class CFluido;
6
7 ///Classe contendo as caracteristicas do fluido produzido
8 class CFluido
9 {
10     //privados, so acessados por meio de funcoes get
11     private:
12
13         ///Fator Volume formacao do fluido
14         double fatorVolumeFormacao;
15         ///Viscosidade
16         double viscosidade;
17         ///Compressibilidade Total
18         double compressibilidade;
19
20     public:
21
22         ///Funcao que recebe dados do usuario, preenchendo
23             atributos fatorvolumeformacao, viscosidade,
24             /// compressibilidade
25         void EntradaDados();
26
27         ///Funcao que verifica se houve erro na entrada de dados e
```

```

          pede nova entrada ate nao ocorrer erro
27      void Erro();

28
29      ///Funcao que seta o fatorvolumeformacao
30      void FatorVolumeFormacao(double _fatorVolumeFormacao);
31      ///Funcao get do fatorvolumeformacao
32      double FatorVolumeFormacao() const;

33
34      ///Funcao que seta a viscosidade
35      void Viscosidade(double _viscosidade);
36      ///Funcao get da viscosidade
37      double Viscosidade() const;

38
39      ///Funcao que seta a compressibilidade
40      void Compressibilidade(double _compressibilidade);
41      ///Funcao get da compressibilidade
42      double Compressibilidade() const;

43
44};

45#endif

```

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

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

```

1 #include "CFluido.h"
2
3 // inclui a biblioteca iostream pois usa funcoes de entrada e saida
4 // de dados para a tela
4 #include <iostream>
5
6 using namespace std;
7
8
9 //funcao de entrada de dados da classe
10 void CFluido::EntradaDados()
11 {
12     cout << "Informe o fator volume-formacao do fluido:" << endl;
13     cin >> fatorVolumeFormacao;
14     cin.get();
15
16     cout << "Informe a viscosidade do fluido:" << endl;
17     cin >> viscosidade;
18     cin.get();

```

```
19
20 cout << "Informe a compressibilidade total (fluido+rocha):" <<
21 endl;
22 cin >> compressibilidade;
23 cin.get();
24
25 //funcao que acusa e conserta erro de entrada
26 void CFluido::Erro()
27 {
28
29     //repete a entrada enquanto o valor for equivocado
30
31     while (fatorVolumeFormacao<0.00)
32     {
33         cout << "Reinforme o Fator Volume-Formacao:" << endl;
34         cin >> fatorVolumeFormacao;
35         cin.get();
36     }
37
38     while (viscosidade<0.00)
39     {
40         cout << "Reinforme a viscosidade." << endl;
41         cin >> viscosidade;
42         cin.get();
43     }
44
45     while (compressibilidade<0.00)
46     {
47         cout << "Reinforme a compressibilidade." << endl;
48         cin >> compressibilidade;
49         cin.get();
50     }
51 }
52
53
54 //set
55 void CFluido::FatorVolumeFormacao(double _fatorVolumeFormacao)
56 {
57     fatorVolumeFormacao = _fatorVolumeFormacao;
58 }
59
```

```

60 //get
61 double CFluido::FatorVolumeFormacao() const
62 {
63     return fatorVolumeFormacao;
64 }
65
66 //set
67 void CFluido::Viscosidade(double _viscosidade)
68 {
69     viscosidade = _viscosidade;
70 }
71
72 //get
73 double CFluido::Viscosidade() const
74 {
75     return viscosidade;
76 }
77
78 //set
79 void CFluido::Compressibilidade(double _compressibilidade)
80 {
81     compressibilidade = _compressibilidade;
82 }
83
84 //get
85 double CFluido::Compressibilidade() const
86 {
87     return compressibilidade;
88 }

```

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

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

```

1 ///Condicao para nao definir a classe mais de uma vez
2 #ifndef CDadosRegistradorPressao_h
3 #define CDadosRegistradorPressao_h
4
5 ///incluir a biblioteca vector pois ha declaracao de vetor
6 #include <vector>
7 #include <string>
8
9 class CDadosRegistradorPressao;
10

```

```

11///Classe que contem dados registrados do registrador de pressao
12class CDadosRegistradorPressao
13{
14    ///privados, so acessados por meio de funcoes get
15    private:
16        ///pressao medida apos o fechamento da producao
17        std::vector<double> pressaoMedida;
18        ///tempo apos o fechamento da producao em que foi medida a
19        pressao
20        std::vector<double> tempoSemProducao;
21
22
23    ///Funcao que importa os dados registrados do arquivo .dat,
24    ///preenchendo os atributos da classe
24    void Importa();
25
26    ///Funcao que seta a pressaomedida
27    void PressaoMedida(std::vector<double> _pressaoMedida);
28    ///Funcao get da posicao informada do vetor pressaomedida
29    double PressaoMedida(int posicao) const;
30    ///Funcao get da pressaomedida
31    std::vector<double> PressaoMedida() const;
32
33    ///Funcao que seta o temposemproducao
34    void TempoSemProducao(std::vector<double>
35        _tempoSemProducao);
36    ///Funcao get da posicao informada do vetor
37    ///temposemproducao
38    double TempoSemProducao(int posicao) const;
39    ///Funcao get do temposemproducao
40    std::vector<double> TempoSemProducao() const;
41};

41#endif

```

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

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

```

1#include "CDadosRegistradorPressao.h" //inclui o cabecalho da
2classe
3//inclui biblioteca para importacao de arquivos de disco

```

```
4 #include <fstream>
5
6 // inclui a biblioteca iostream pois usa funcoes cin, cout
7 #include <iostream>
8
9 // inclui a biblioteca vector pois usa vetores
10 #include <vector>
11
12 // inclui a biblioteca string pois usa variaveis string
13 #include <string>
14
15 // usa funcoes pertencentes ao namespace std
16 using namespace std;
17
18 void CDadosRegistradorPressao::Importa()
19 {
20     // limpa os vetores de importacao para novo preenchimento
21     tempoSemProducao.resize(0);
22     pressaoMedida.resize(0);
23
24     // indica o que o eixo x representa
25     string eixox;
26     // indica o que o eixo y representa
27     string eixoy;
28     double x;
29     double y;
30     // nome do arquivo com os dados a serem importados
31     string nomeArquivo;
32
33     cout << "Informe o nome do arquivo com os dados do registrador de"
34         " pressao:" << endl;
35     // armazena a string digitada em nomearquivo
36     getline (cin,nomeArquivo);
37
38     ifstream fin;
39     // converte a string de c++ para c, necessario para funcao
40     fin.open (nomeArquivo.c_str());
41
42     // pega o primeiro valor, o nome do eixo x
43     fin >> eixox;
44     // pega o segundo valor, o nome do eixo y
```

```
45     fin >> eixoy;
46
47     //fazer ate o fim do arquivo
48     while (!fin.eof())
49     {
50         //valores de x e y alternados e separados por um espaco
51         fin >> x;
52         //para adicionar no fim do vetor, otimizando memoria
53         tempoSemProducao.push_back (x);
54         fin >> y;
55         pressaoMedida.push_back (y);
56     }
57 }
58
59 //set
60 void CDadosRegistradorPressao::PressaoMedida(vector<double>
61 _pressaoMedida)
62 {
63     pressaoMedida = _pressaoMedida;
64 }
65
66 //get da posicao
67 double CDadosRegistradorPressao::PressaoMedida(int posicao) const
68 {
69     return pressaoMedida[posicao];
70 }
71
72 //get
73 vector<double> CDadosRegistradorPressao::PressaoMedida() const
74 {
75     return pressaoMedida;
76 }
77
78 //set
79 void CDadosRegistradorPressao::TempoSemProducao(vector<double>
80 _tempoSemProducao)
81 {
82     tempoSemProducao = _tempoSemProducao;
83 }
84
85 //get da posicao
86 double CDadosRegistradorPressao::TempoSemProducao(int posicao)
```

```
    const
85 {
86     return tempoSemProducao[posicao];
87 }
88
89 //get
90 vector<double> CDadosRegistradorPressao::TempoSemProducao() const
91 {
92     return tempoSemProducao;
93 }
```

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

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

```
1 ///Condicao para nao definir a classe mais de uma vez
2 #ifndef CEstatistica_h
3 #define CEstatistica_h
4
5 ///inclui vector pois ha parametros declarados que sao vetores
6 #include <vector>
7
8 class CEstatistica;
9
10 ///Classe que calcula estatisticas do vetor, como media e desvio
11 padrao, util para regressao
12 class CEstatistica
13 {
14
15     double media;
16     double desvio;
17
18 public:
19
20     ///retorna a media do vetor informado
21     double Media(std::vector<double> v);
22
23     ///retorna o desvio padrao do vetor informado
24     double DesvioPadrao(std::vector<double> v);
25
26 };
27
28#endif
```

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

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

```

1 #include "CEstatistica.h"
2
3 // inclui a biblioteca vector pois usa a função size
4 #include <vector>
5
6 // inclui a biblioteca cmath pois usa a função pow
7 #include <cmath>
8
9 using namespace std;
10
11 double CEstatistica::Media(vector<double> v)
12 {
13     double soma = 0.0;
14     // loop que faz a soma de todos os elementos do vetor
15     for ( int i = 0 ; i < (v.size()) ; i++)
16         soma = soma + v[i];
17
18     return media = soma/v.size();
19 }
20
21 double CEstatistica::DesvioPadrao(vector<double> v)
22 {
23     double soma = 0.0;
24     double vquadrado = 0.0;
25     desvio = 0.0;
26
27     // loop que faz a soma dos elementos do vetor elevados ao quadrado
28     for ( int i = 0 ; i < (v.size()) ; i++)
29     {
30         soma = soma + v[i];
31         vquadrado = vquadrado + (v[i]*v[i]);
32     }
33
34
35     return desvio = pow(((vquadrado - ((1.0/v.size())*soma*soma))/(v.
36         size()-1.0)) ,0.5);
}

```

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

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

```

1///Condicao para nao definir a classe mais de uma vez
2#ifndef CAjusteCurvaMinimosQuadrados_h
3#define CAjusteCurvaMinimosQuadrados_h
4
5///inclui o cabecalho da classe que sera utilizada
6#include "CEstatistica.h"
7///inclui vector pois ha parametros declarados que sao vetores
8#include <vector>
9
10class CAjusteCurvaMinimosQuadrados;
11
12///Classe que obtém os coeficiente da regressao linear por meio do
13///metodo dos minimos quadrados
13class CAjusteCurvaMinimosQuadrados
14{
15
16 //encapsulamento que permite o acesso para a classe e para a
17 //classe herdeira
17 protected:
18     ///coeficiente angular da reta do tipo y=ax+b
19     double coeficienteAngular;
20     ///coeficiente linear da reta do tipo y=ax+b
21     double coeficienteLinear;
22     ///coeficiente de correlacao da reta do tipo y=ax+b
23     double coeficienteCorrelacao;
24
25     ///cria um objeto da classe CEstatistica para ser utilizado
26     ///funcoes de calculo
26     CEstatistica estatistica;
27
28 protected:
29
30     ///Funcao que retorna o valor do coeficiente angular
31     double CoeficienteAngular (std::vector<double> vx, std::
32                             vector<double> vy);
32
33     ///Funao que retorna o valor do coeficiente angular
34     ///logaritimico
34     double CoeficienteAngularLog (std::vector<double> vx, std::
35                               vector<double> vy);
35

```

```

36     ///Funcao que retorna o valor do coeficiente linear
37     double CoeficienteLinear (std::vector<double> vx, std::
38         vector<double> vy);
39     ///Set do P1h
40     double CoeficienteLinearP1h (double _P1h);
41
42     ///Funcao que retorna o valor do coeficiente de correlacao
43     double CoeficienteCorrelacao (std::vector<double> vx, std::
44         vector<double> vy);
45
46 #endif

```

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

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

```

1 #include "CAjusteCurvaMinimosQuadrados.h"
2
3 //inclui a biblioteca cmath pois usa a funcao logaritmica
4 #include <cmath>
5
6 //inclui a biblioteca vector pois usa funcoes dos vetores:
7     push_back, resize
7 #include <vector>
8
9 //usa funcoes pertencentes ao namespace std
10 using namespace std;
11
12 //retorna o valor do coeficiente angular
13 double CAjusteCurvaMinimosQuadrados::CoeficienteAngular (vector<
14     double> vx, vector<double> vy)
14 {
15     double mnum = 0.0; //termo do denominador
16     double mden = 0.0; //termo do numerador
17
18     double mediax = estatistica.Media(vx);
19     double mediay = estatistica.Media(vy);
20
21     for (int j=0 ; j<vx.size() ; j++) //percorre todo o vetor
22     {
23         //metodo dos minimos quadrados

```

```
24         mnum = mnum + (vx[j] * (vy[j] - mediay));
25         mden = mden + (vx[j] * (vx[j] - mediax));
26     }
27
28     return coeficienteAngular = mnum/mden;
29
30 }
31
32 double CAjusteCurvaMinimosQuadrados::CoeficienteAngularLog(std::
33     vector<double> vx, std::vector<double> vy)
34 {
35     double mnum = 0.0; //termo do denominador
36     double mden = 0.0; //termo do numerador
37
38     double mediax = estatistica.Media(vx);
39     double mediay = estatistica.Media(vy);
40
41     mnum = log10(vx[vx.size() - 1]) - log10(vx[0]);
42     mden = vy[vy.size() - 1] - vy[0];
43
44     return coeficienteAngular = mden/mnum;
45 }
46 //retorna o valor do coeficiente linear
47 double CAjusteCurvaMinimosQuadrados::CoeficienteLinear (vector<
48     double> vx, vector<double> vy)
49 {
50     return coeficienteLinear = estatistica.Media(vy) - (
51             coeficienteAngular * estatistica.Media(vx));;
52 }
53
54 double CAjusteCurvaMinimosQuadrados::CoeficienteLinearP1h(double
55     _P1h)
56 {
57     return coeficienteLinear = _P1h;
58 }
59
60 double somar = 0.0;
```

```

61     double variax = 0.0;
62     double variay = 0.0;
63
64     double mediax = estatistica.Media(vx);
65     double mediay = estatistica.Media(vy);
66
67     for (int j=0 ; j<vx.size() ; j++)
68     {
69         somar = somar + ((vx[j] - mediax) * (vy[j] - mediay));
70         variax = variax + (pow ((vx[j] - mediax),2));
71         variay = variay + (pow ((vy[j] - mediay),2));
72     }
73
74     return coeficienteCorrelacao = -somar / pow ((variay *
75 }  


```

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

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

```

1///Condicao para nao definir a classe mais de uma vez
2#ifndef CAjusteCurva_h
3#define CAjusteCurva_h
4
5///inclui a biblioteca vector pois ha declaracao de vetor
6#include <vector>
7
8///inclui o cabecalho da classe pai
9#include "CAjusteCurvaMinimosQuadrados.h"
10#include "CMetodo.h"
11
12///Declaracao da Classe filha de CAjusteCurvaMinimosQuadrados
13class CAjusteCurva;
14
15///Classe que executa a regressao linear e ajusta ate a correlacao
16///satisfatoria
16class CAjusteCurva: public CAjusteCurvaMinimosQuadrados
17{
18    public:
19        ///coef. angular
20        double m;
21        ///coef. linear
22        double n;  


```

```

23         ///coef. de correlacao
24         double r;
25         ///indica que o coef. de correlacao nao foi satisfatorio (
26             ha estocagem)
27         bool efeitoEstocagem;
28         ///ajuste de variavel para logaritmica
29         std::vector<double> logt;
30
31     public:
32         ///Funcao que executa a regressao linear atraves do metodo
33             dos minimos quadrados
34         void Regressao(std::vector<double> vx, std::vector<double>
35             vy, double z, CMetodo _metodo, double _P1h);
36         ///Funcao que ajusta a regressao para o periodo correto,
37             removendo os pontos referentes a estocagem
38         void Ajuste(std::vector<double> vx, std::vector<double> vy,
39             double z, CMetodo _metodo, double _P1h);
40
41     };
42
43 //Fim da condicao de definicao da classe
44 #endif

```

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

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

```

1 #include "CAjusteCurva.h"
2
3 // inclui a biblioteca iostream pois usa funcoes de entrada e saida
4 // de dados para a tela
4 #include <iostream>
5
6 // inclui a biblioteca cmath pois usa a funcao logaritmica
7 #include <cmath>
8
9 // inclui a biblioteca vector pois usa funcoes dos vetores:
10 // push_back, resize
10 #include <vector>
11
12 // inclui biblioteca para usar modulo
13 #include <cstdlib>
14
15

```

```
16 // usa funcoes pertencentes ao namespace std
17 using namespace std;
18
19 // Funcao que cria a variavel logaritmica a partir dos parametros 1
20 // e 3 da funcao, executa a
21 // regressao linear atraves do metodo dos minimos quadrados.
22 void CAjusteCurva::Regressao(vector<double> vx, vector<double> vy,
23     double z, CMetodo _metodo, double _P1h = 0.0)
24 {
25     //limpa o vetor do eixo x para novo preenchimento
26     logt.resize(0);
27
28     //loop que percorre todo o vetor vx e preenche logt
29     switch (_metodo)
30     {
31         case CMetodo::METODO_HORNER:
32             for(int i=0 ; i<vx.size() ; i++)
33                 //transformacao da variavel em logaritmica
34                 logt.push_back (log10((z+vx[i]) / vx[i]));
35
36             m = CoeficienteAngular (logt,vy);
37             n = CoeficienteLinear (logt,vy);
38             r = CoeficienteCorrelacao (logt,vy);
39
40             break;
41         case CMetodo::METODO_MDH:
42             for(int i=0 ; i<vx.size() ; i++)
43                 //transformacao da variavel em logaritmica
44                 logt.push_back (vx[i]);
45
46             m = CoeficienteAngularLog (logt,vy);
47             n = CoeficienteLinearP1h(_P1h);
48             r = CoeficienteCorrelacao (logt,vy);
49
50             break;
51     default:
52         break;
53 }
```

```
56
57     cout << "EQUACAO: " << m << " * x + " << n << endl << " r = "
58     << r << endl;
59
60
61 //Funcao que ajusta a regressao para o periodo correto, removendo
62 //os pontos referentes a estocagem
62 void CAjusteCurva::Ajuste(vector<double> vx, vector<double> vy,
63                           double z, CMetodo _metodo, double _P1h = 0.0)
63 {
64     // variavel que contem os coef. de correlacao
65     vector<double> coef(vx.size()/1.2,r);
66
67     //variavel que ajusta o eixo y
68     vector<double> y;
69
70     //variavel que ajusta o eixo x
71     vector<double> t;
72
73     //loop principal que vai aumentando o valor de k e retirando as
74     //primeiras posicoes dos vetores (estocagem)
74     for (int k=1 ; k<(vx.size()/1.2) ; k++)
75     {
76         //repete o loop ate achar o coef. de correlacao aceitavel
77         if(abs(coef[k-1])<0.9900)
78         {
79             //ocorre estocagem se cair na condicao
80             efeitoEstocagem = true;
81             cout << "Necessario novo Ajuste." << endl;
82
83             //limpa os vetores
84             t.resize(0);
85             y.resize(0);
86
87             switch (_metodo)
88             {
89                 case CMetodo::METODO_HORNER :
90
91             {
92                 for(int l=0 ; l<vx.size() ; l++)
93                 {
```

```
94                     //define o eixo x
95                     t.push_back (log10((z + vx[1])/vx[1
96                                         ]));
97                     //define o eixo y
98                     y.push_back (vy[1]);
99                 }
100
101             for(int w=0 ; w<(vx.size()-k) ; w++)
102             {
103                 //retira o primeiro valor do vetor
104                 //estocagem
105                 t[w] = t[w+k];
106                 //se repetir o if, vai retirando
107                 y[w] = vy[w+k];
108                 //até terminar a estocagem (coef.
109                 //será bom)
110             }
111
112             //redefine o tamanho dos vetores
113             t.resize (vx.size()-k);
114             y.resize (vx.size()-k);
115
116             //nova regressão linear
117             m = CoeficienteAngular (t,y);
118             n = CoeficienteLinear (t,y);
119             //novo coeficiente de correlação
120             coef[k] = CoeficienteCorrelacao (t,y);
121
122             cout << "EQUAÇÃO:y=" << m << "x+" <<
123             n << endl << "r=" << coef[k] <<
124             endl;
125         }
126
127         case CMetodo::METODO_MDH:
128         {
129             for(int l=0; l<vx.size(); l++)
130             {
131                 //define o eixo x
132                 t.push_back(vx[l]);
```

```

131                         //define o eixo y
132                         y.push_back (vy[1]);
133                     }
134
135                 for(int w=0 ; w<(vx.size()-k) ; w++)
136                 {
137                     //retira o primeiro valor do vetor
138                     // (estocagem)
139                     t[w] = t[w+k];
140                     //se repetir o if, vai retirando
141                     cout<< "t[x]= " << t[w] << endl;
142                     y[w] = vy[w+k];
143                     //até terminar a estocagem (coef.
144                     //será bom)
145
146                     t.resize (vx.size()-k);
147                     y.resize (vx.size()-k);
148
149
150                     //nova regressão linear
151                     m = CoeficienteAngularLog (t,y);
152                     n = CoeficienteLinearP1h(_P1h);
153                     //novo coeficiente de correlação
154                     coef[k] = CoeficienteCorrelacao (t,y);
155
156                     cout << "EQUAÇÃO:y=" << m << "*x+" <<
157                     << n << endl << "r=" << coef[k] <<
158                     endl;
159
160                     break;
161
162             default:
163                 break;
164         }
165
166         if (abs(1.2*coef[k])<coef[0])
167             cout << "Regressão Linear não é perfeita,
168             Indicativo de Reservatório Heterogêneo"
169             << endl;

```

```

167         if (k>1) //apos a segunda regressao
168     {
169         if ((abs(1.2*coef [k]))<coef [k-1])
170     {
171         k = vx.size()/1.2; //para terminar
172         o loop
173         cout << "Maximo Coeficiente de"
174         Correlacao alcançado." << endl
175         ;
176     }
177 }
178 } // Fecha loop.
179 } // 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.

```

1 #ifndef CBUILDUP_H
2 #define CBUILDUP_H
3
4 /**
5 * Classe virtual para metodos aplicaveis
6 * Declaração dos calculos basicos de reservatorio
7 *
8 * @param m_Permeabilidade, m_Viscosidade
9 *
10 */
11
12 class CBuildUP {
13
14     public:
15
16     virtual ~CBuildUP() = default;
17
18     virtual void CalculoPermeabilidade(double _sistemaUnidade,
19                                         double _vazao, double _fatorVolumeFormacao, double
20                                         _viscosidade, double _m, double _altura);
21     virtual void CalculoFatorPelicula(double _m, double
22                                         _tempoProducao, double _n, double _pressao, double
23                                         _sistemaUnidade, double _porosidade, double _viscosidade,

```

```

    double _compressibilidade, double _raioPoco );
20
21     double getPermeabilidade() { return m_Permeabilidade; }
22     double getPelicula() {return m_Pelicula; }
23
24 protected:
25
26     double m_Permeabilidade;
27     double m_Pelicula;
28
29};
30
31#endif

```

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

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

```

1 #include "CBuildUp.h"
2
3 void CBuildUP::CalculoPermeabilidade(double _sistemaUnidade, double
4                                         _vazao, double _fatorVolumeFormacao, double _viscosidade, double
5                                         _m, double _altura)
6 {
7
8 void CBuildUP::CalculoFatorPelicula(double _m, double
9                                         _tempoProducao, double _n, double _pressao, double
10                                        _sistemaUnidade, double _porosidade, double _viscosidade, double
11                                         _compressibilidade, double _raioPoco)

```

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

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

```

1 ifndef CMETODO_H
2 define CMETODO_H
3
4 /**
5 *
6 * Metodos atualmente compatíveis
7 *

```

```

8 * @param none , HOPNER , MDH
9 *
10 */
11
12 enum class CMetodo {
13
14     none = 0, METODO_HOPNER, METODO_MDH
15
16 };
17
18 #endif

```

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

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

```

1 ifndef METODOMDH_H
2 define METODOMDH_H
3
4 include "CBuildUp.h"
5
6 class CMetodoMDH : public CBuildUP{
7
8     public:
9
10    CMetodoMDH() {};
11    ~CMetodoMDH() {};
12
13    virtual void CalculoPermeabilidade(double _sistemaUnidade,
14                                         double _vazao, double _fatorVolumeFormacao, double
15                                         _viscosidade, double _m, double _altura) override;
16    virtual void CalculoFatorPelicula(double _m, double
17                                         _tempoProducao, double _n, double _pressao, double
18                                         _sistemaUnidade, double _porosidade, double _viscosidade
19                                         , double _compressibilidade, double _raioPoco) override;
20
21};
22
23#endif

```

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

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

```

1 include "CMetodoMDH.h"
2

```

```

3 #include <iostream>
4 #include <cmath>
5
6 void CMetodoMDH::CalculoPermeabilidade(double _sistemaUnidade,
7   double _vazao, double _fatorVolumeFormacao, double _viscosidade,
8   double _m, double _altura)
9 {
10   m_Permeabilidade = (1.151 * _sistemaUnidade * _vazao *
11     _fatorVolumeFormacao *
12       _viscosidade) / (_m * _altura);
13
14   if ((_sistemaUnidade==141.2) || (_sistemaUnidade==19.03))
15     std::cout << "Permeabilidade:" << m_Permeabilidade;
16
17   if (_sistemaUnidade==0.3183)
18     std::cout << " milidarcy" << std::endl;
19
20 }
21
22 void CMetodoMDH::CalculoFatorPelicula(double _m, double
23   _tempoProducao, double _n, double _pressao, double
24   _sistemaUnidade, double _porosidade, double _viscosidade, double
25   _compressibilidade, double _raioPoco)
26 {
27   m_Pelicula = 1.151 * (((_m * log10(_tempoProducao)) + _n -
28     _pressao) /
29       _m) - log10((_sistemaUnidade *
30         m_Permeabilidade) /
31       (_porosidade * _viscosidade *
32         _compressibilidade
33         * _raioPoco * _raioPoco)) - 0.3514 + log10
34         (_tempoProducao+1));
35
36   std::cout << "Fator de Pelicula:" << m_Pelicula << std::
37   endl;
38
39 }

```

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

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

```
1 #ifndef METODOHORNER_H
2 #define METODOHORNER_H
3
4 #include "CBuildUp.h"
5
6 class CMetodoHorner : public CBuildUP {
7
8     public:
9
10    CMetodoHorner() {};
11    ~CMetodoHorner() {};
12
13    virtual void CalculoPermeabilidade(double _sistemaUnidade,
14                                         double _vazao, double _fatorVolumeFormacao, double
15                                         _viscosidade, double _m, double _altura) override;
16    virtual void CalculoFatorPelicula(double _m, double
17                                         _tempoProducao, double _n, double _pressao, double
18                                         _sistemaUnidade, double _porosidade, double _viscosidade
19                                         , double _compressibilidade, double _raioPoco) override;
20
21
22 };
23
24#endif
```

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

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

```
1 #include "CMetodoHorner.h"
2
3 #include <iostream>
4 #include <cmath>
5
6 void CMetodoHorner::CalculoPermeabilidade(double _sistemaUnidade,
7                                             double _vazao, double _fatorVolumeFormacao, double _viscosidade,
8                                             double _m, double _altura)
9 {
10    m_Permeabilidade = (1.151 * _sistemaUnidade * _vazao *
11                           _fatorVolumeFormacao *
12                           _viscosidade) / (-_m * _altura);
```

```

12     std::cout << "Permeabilidade:" << m_Permeabilidade;
13
14     if (_sistemaUnidade==141.2) || (_sistemaUnidade==19.03))
15         std::cout << " milidarcy" << std::endl;
16
17     if (_sistemaUnidade==0.3183)
18         std::cout << " metros quadrados" << std::endl;
19
20 }
21
22 void CMetodoHorner::CalculoFatorPelicula(double _m, double
23     _tempoProducao, double _n, double _pressao, double
24     _sistemaUnidade, double _porosidade, double _viscosidade, double
25     _compressibilidade, double _raioPoco)
26 {
27     m_Pelicula = 1.151 * (((_m * log10(_tempoProducao)) + _n -
28     _pressao) /
29             -_m) - log10((_sistemaUnidade *
30             m_Permeabilidade) /
31             (_porosidade * _viscosidade *
32             _compressibilidade
33             * _raioPoco * _raioPoco)) - 0.3514 + log10
34             (_tempoProducao+1));
35
36
37     std::cout << "Fator de Pelicula:" << m_Pelicula << std::
38             endl;
39 }

```

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

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

```

1 /// Condicao para nao definir a classe mais de uma vez
2 #ifndef CCaracterizacaoReservatorio_h
3 #define CCaracterizacaoReservatorio_h
4
5 class CCaracterizacaoReservatorio;
6
7
8 /// Classe que caracteriza o reservatorio.
9 class CCaracterizacaoReservatorio
10 {

```

```

11
12 public:
13     ///Funcao que analisa os resultados e caracteriza o
14     ///reservatorio.
15     void Caracterizacao(double permeabilidade, double
16         fatorpelicula, double indiceprodutividade, double
17         raiopoco, double raioefetivo);
18
19 };
20
21 #endif

```

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

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

```

1 #include "CCaracterizacaoReservatorio.h"
2
3 //inclui a biblioteca iostrem pois usa funcoes de entrada e saida
4 //de dados para a tela
5
6 //usa funcoes pertencentes ao namespace std
7 using namespace std;
8
9 //Funcao que caracteriza o reservatorio, dado os parametros
10 //calculados
11 void CCaracterizacaoReservatorio::Caracterizacao (double
12     permeabilidade, double fatorpelicula, double indiceprodutividade
13     , double raiopoco, double raioefetivo)
14 {
15     //condicoes que se satisfeitas, escrevem na tela
16     //caracteristicas do reservatorio.
17
18     if (permeabilidade<=10)
19         cout << "1-Reservatoriocompermeabilidaderuim." << endl;
20
21     if ((permeabilidade<100)&&(permeabilidade>10))
22         cout << "1-Reservatoriocompermeabilidadeboa." << endl;
23
24     if (permeabilidade>=100)
25         cout << "1-Reservatoriocompermeabilidadeexcelente." <<
26             endl;
27
28 }

```

```

24  if (fatorpelicula==0)
25      cout << "2-uReservatorio sem dano e sem estímulo." << endl;
26
27  if (fatorpelicula<0)
28      cout << "2-uReservatorio estimulado" << endl;
29
30  if ((fatorpelicula>0)&&(fatorpelicula<5))
31      cout << "2-uReservatorio com dano baixo , uao necessita de
32          processos de acidificacao ou fraturamento hidraulico." <<
33          endl;
34
35  if ((fatorpelicula>5)&&(fatorpelicula<10))
36      cout << "2-uReservatorio com dano intermediario , pode ser
37          usado processos de acidificacao ou fraturamento
38          hidraulico." << endl;
39
40  if (indiceprodutividade<=0.01)
41      cout << "3-uReservatorio com produtividade baixo , considerar
42          uso de tecnicas de recuperacao secundaria." << endl;
43
44  if ((indiceprodutividade>0.01)&&(indiceprodutividade<0.1))
45      cout << "3-uReservatorio com produtividade regular , u
46          considerar uso de tecnicas de recuperacao secundaria." <<
47          endl;
48
49  if ((raioefetivo/raiopoco)<=0.0001)
50      cout << "4-uRazao de dano alto , pois o raiio efetivo e muito
51          menor que o raiio do poco real , afetando a produtividade ."
          << endl;
}

```

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

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

```
1#ifndef CSimuladorAnaliseTestePressao_h
2#define CSimuladorAnaliseTestePressao_h
3
4#include <fstream>
5#include <iostream>
6///inclusao de arquivos de classe necessarios
7
8#include "cgnuplot.h"
9#include "CReservatorio.h"
10#include "CFluido.h"
11#include "CPoco.h"
12#include "CDadosRegistradorPressao.h"
13#include "CEstatistica.h"
14#include "CAjusteCurva.h"
15#include "CCaracterizacaoReservatorio.h"
16
17#include "CMetodo.h"
18#include "CBuildUp.h"
19#include "CMetodoHorner.h"
20#include "CMetodoMDH.h"
21
22
23
24///criacao do objeto caracterizar da classe CCaracterizacao.h"
25
26class CSimuladorAnaliseTestePressao;
27
28///Classe que faz a analise do teste de pressao realizado no campo
29///e infere as propriedades do reservatorio
30class CSimuladorAnaliseTestePressao
31{
32    public:
33        ///Nome do Arquivo exportado
34        std::string nome;
35        ///pressao inicial que se encontrava o reservatorio
36        double pressaoInicial;
37        ///queda de pressao referente ao fator de pelicula
38        double pressaoDano;
39        ///indice de produtividade do reservatorio
40        double indiceProdutividade;
```

```
41     ///eficiencia de fluxo do reservatorio
42     double eficienciaFluxo;
43     ///raio efetivo do poco
44     double raioEfetivo;
45     ///coeficiente de estocagem do poco
46     double coeficienteEstocagem;
47     ///tempo de duracao do efeito de estocagem
48     double tempoEstocagem;
49     ///cria objeto de armazenamento de dados
50     std::ofstream fout;
51     ///criacao do objeto poco da classe CPoco
52     CPoco poco;
53     ///criacao do objeto fluido da classe CFluido
54     CFluido fluido;
55     ///criacao do objeto reservatorio da classe CReservatorio
56     CReservatorio reservatorio;
57     ///criacao do objeto registrador da classe CRegistrador
58     CDadosRegistradorPressao registrador;
59     ///criacao do objeto ajuste da classe CAjuste
60     CAjusteCurva ajuste;
61     ///criacao do objeto caracterizar da classe
62     CCaracterizacao
63     CCaracterizacaoReservatorio caracterizar;
64     ///criacao do objeto buildup
65     CBuildUP* buildup;
66     ///criacao do objeto plot da classe CGnuplot
67     //CGnuplot plot;

68 public:
69
70     ///Funcao que calcula e preenche o atributo permeabilidade
71     void CalculoPermeabilidade ();
72     ///Funcao que calcula e preenche o atributo pressao inicial
73     void CalculoPressaoInicial ();
74     ///Funcao que calcula e preenche o atributo fator pelicula
75     void CalculoFatorPelicula ();
76     ///Funcao que calcula e preenche o atributo raioefetivo
77     void CalculoRaioEfetivo ();
78     ///Funcao que calcula e preenche os atributos
79     indiceprodutividade, eficienciafluxo, pressaodano
80     void CalculoIndices ();
81     ///Funcao que calcula e preenche os atributos
```

```

        coeficienteestocagem e tempoestocagem
81     void CalculoEstocagem ();
82     ///Funcao que chama as entradas de dados necessarias
83     void EntradaDados ();
84     ///Funcao principal que executa a simulacao do teste
85     void Executar ();
86
87         ///Varia parametros de reservatorio
88         void Variacao ();
89         ///Funcao que exporta os dados para um arquivo.dat
90     void Exporta ();
91
92
93 };
94
95
96
97 #endif

```

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

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

```

1 #include "CSimuladorAnaliseTestePressao.h"
2
3 // inclui a biblioteca iostream pois usa funcoes de entrada e saida
4 // de dados para a tela
4 #include <iostream>
5
6 // inclui a biblioteca cmath pois usa a funcao logaritmica
7 #include <cmath>
8
9 // inclui a biblioteca vector pois usa funcoes dos vetores:
10 // push_back, resize
10 #include <vector>
11
12 // inclui a biblioteca que guarda os dados
13 #include <fstream>
14
15 // inclui biblioteca string que le caracteres
16 #include <string>
17
18 // biblioteca que permite manipulacao de variaveis

```

```

19 #include <sstream>
20
21 //usa funcoes pertencentes ao namespace std
22 using namespace std;
23
24 ///sobrecarga operador >>
25 std::istream &operator>>(std::istream &is, CMetodo &metodo)
26 {
27     unsigned int a;
28     is >> a;
29     metodo = static_cast<CMetodo>(a);
30
31     return is;
32 }
33
34 //Funcao principal que executa a simulacao do teste
35 void CSimuladorAnaliseTestePressao::Executar()
36 {
37     cout << endl << "PROGRAMA PARA CALCULO DE PARAMETROS DE "
38             "RESERVATORIO POR TESTES DE PRESSAO" << endl
39             << endl << "1-Rodar o Programa" << endl << "2-Sair" <<
40             endl << endl;
41
42     int i = 0;
43     cin >> i;
44
45     while (i==1) //quando terminar a execucao do programa, se o
46             usuario quiser, o programa roda novamente
47     {
48         CMetodo ans;
49         do
50         {
51             cout << "Escolha o metodo a ser aplicado: 1-Horner
52             , 2-MDH" << endl;
53             cin >> ans;
54             cin.get();
55
56             switch (ans)
57             {
58                 case CMetodo::METODO_HORNER:
59                     buildup = new CMetodoHorner();
60                     break;
61                 case CMetodo::METODO_MDH:
62
63             }
64         }
65     }
66 }
```

```
57             buildup = new CMetodoMDH();
58             break;
59         default:
60             cout << "Opção invalida!!" << endl;
61             ans = CMetodo::none;
62             break;
63         }
64
65     }while (ans == CMetodo::none);
66
67     cout << "Entrada de Dados do teste de pressão realizado"
68         << endl
69         << "
-----"
70         << endl;
71
72     EntradaDados();
73     double P1h = 0.0;
74     if (ans == CMetodo::METODO_MDH)
75     {
76         cout << "entre com valor P1h" << endl;
77         cin >> P1h;
78         cin.get();
79     }
80
81     cout << "Entrada de Dados Finalizada" << endl
82         << "
-----"
83         << endl << endl;
84     cout << "Importação dos dados do registrador de"
85         " Pressão" << endl
86         << "
-----"
87         << endl << endl;
88
89     registrador.Importa();
90     cout << "Dados do registrador importados com sucesso"
91         << endl << endl;
92
93     cout << "Regressão Linear dos Dados" << endl
94         << "
-----"
```

```
    " << endl << endl;

89
90     ajuste.Regressao (registrador.TempoSemProducao() ,
91                     registrador.PressaoMedida(), poco.TempoProducao() ,
92                     ans , P1h);

93     cin.get ();
94     cout << "Localizando o Periodo Transiente" << endl
95             << "
96
97             -----"
98             " << endl << endl;

99
100            ajuste.Ajuste(registrador.TempoSemProducao() ,
101                      registrador.PressaoMedida(), poco.TempoProducao() ,
102                      ans , P1h);

103
104            cout << "Regressao linear feita com sucesso" << endl
105                  << "
106
107                  -----"
108                  " << endl;
109
110            cout << "Ajuste N = " << ajuste.n << endl;
111
112            //GERA O GRAFICO CASO USUARIO QUEIRA
113
114            cout << "Deseja gerar o grafico: " << endl << "1-Sim"
115                  << endl << "2-Nao" << endl << endl << endl;
116
117            int j;
118            cin >> j;
119
120
121            if (j==1)
122            {
123
124                CGnuplot plot;
125                //gera o grafico com a reta perfeita obtida
126                if(ans == CMetodo::METODO_HORNER)
127                {
128                    plot.plot_slope (ajuste.m,ajuste.n);
129                    cin.get();
130                    if (ajuste.efeitoEstocagem==1)
131                        //compara com os pontos originais
132                        plot.plot_xy (ajuste.logt,registrador.
133                                      PressaoMedida());
```

```
120                     cin.get();
121             } else if (ans == CMetodo::METODO_MDH)
122             {
123                 plot.XLogscale();
124                 plot.plot_logSlope (ajuste.m,ajuste.n);
125                 cin.get();
126                 if (ajuste.efeitoEstocagem==1)
127                     //compara com os pontos originais
128                     plot.plot_xy (ajuste.logt,registrador.
129                         PressaoMedida());
130                     cin.get();
131             }
132
133
134         cout << "Parametros do Reservatorio" << endl
135             << "
136                                         -----
137             " << endl << endl;
138             //CALCULOS
139             CalculoPermeabilidade();
140             CalculoPressaoInicial();
141             CalculoFatorPelicula(); //guardar
142             CalculoRaioEfetivo();
143             CalculoIndices();
144             Exporta();
145             cin.get();
146
147             //se nao ocorreu estocagem
148             if (ajuste.efeitoEstocagem==false)
149                 cout << "Reservatorio sem ou periodo de estocagem"
150                     << endl;
151             else
152                 {
153                     CalculoEstocagem ();
154                     //zera o valor em caso de novo calculo
155                     ajuste.efeitoEstocagem = false;
156                 }
157
158             cout << "Caracterizacao do Reservatorio." << endl
159                 << "
160                                         -----
```

```
    " << endl;

157
158     caracterizar.Caracterizacao(buildup->getPermeabilidade
159         (), buildup->getPelicula(), indiceProdutividade,
160         poco.RaioPoco(), raioEfetivo);
161     cin.get();
162
163     //Nova Escolha
164     cout << "\n1-Rodar o Programa" << endl << "2-Sair" <<
165         endl << endl;
166     cin >> i;
167 }
168
169 //chama as outras entradas de dados
170 void CSimuladorAnaliseTestePressao::EntradaDados()
171 {
172     reservatorio.EntradaSistemaUnidades();
173     reservatorio.EntradaDados();
174     reservatorio.Erro();
175     fluido.EntradaDados();
176     fluido.Erro();
177     poco.EntradaDados();
178     poco.Erro();
179 }
180
181 //Calcula a permeabilidade
182 void CSimuladorAnaliseTestePressao::CalculoPermeabilidade ()
183 {
184     buildup->CalculoPermeabilidade(reservatorio.SistemaUnidade(),
185         poco.Vazao(), fluido.FatorVolumeFormacao(), fluido.
186         Viscosidade(), ajuste.m, reservatorio.Altura());
187
188 //Calcula a pressao inicial
189 void CSimuladorAnaliseTestePressao::CalculoPressaoInicial()
190 {
191     pressaoInicial = ajuste.n;
192     cout << "Pressao Inicial:" << pressaoInicial;
```

```
193
194     if (reservatorio.SistemaUnidade() == 141.2)
195         cout << "psi" << endl;
196
197     if (reservatorio.SistemaUnidade() == 19.03)
198         cout << "kgf/cm²" << endl;
199
200     if (reservatorio.SistemaUnidade() == 0.3183)
201         cout << "Pascal" << endl;
202
203
204 }
205
206
207 // Calcula fator pelicula
208 void CSimuladorAnaliseTestePressao::CalculoFatorPelicula ()
209 {
210     buildup->CalculoFatorPelicula(ajuste.m, poco.TempoProducao()
211         , ajuste.n, poco.PressaoPoco(), reservatorio.
212         SistemaUnidade(), reservatorio.Porosidade(), fluido.
213         Viscosidade(), fluido.Compressibilidade(), poco.RaioPoco
214         ());
215 }
216
217 // Calcula raio efetivo
218
219 void CSimuladorAnaliseTestePressao::CalculoRaioEfetivo()
220 {
221     raioEfetivo = poco.RaioPoco() * exp(-buildup->getPelicula())
222         ;
223     cout << "RaioEfetivo:" << raioEfetivo;
224
225
226     if ((reservatorio.SistemaUnidade() == 0.3183) || (reservatorio
227         .SistemaUnidade() == 19.03))
228         cout << "metros" << endl;
229
230     if (reservatorio.SistemaUnidade() == 141.2)
231         cout << "ft" << endl;
232
233
234 // Calcula do indice de produtividade, eficiencia de fluxo e queda
235 // de pressao referente ao dano
236
237 void CSimuladorAnaliseTestePressao::CalculoIndices ()
```

```

228 {
229     pressaoDano = 0.869 * (-ajuste.m) * buildup->getPelicula();
230
231     indiceProdutividade = poco.Vazao() / (pressaoInicial - poco.
232         PressaoPoco());
233
234     eficienciaFluxo = (pressaoInicial - poco.PressaoPoco() -
235         pressaoDano) / (pressaoInicial - poco.PressaoPoco());
236
237     cout << "Queda de Pressão devido ao dano:" << pressaoDano
238         << endl;
239
240     //Exibir em porcentagens
241     cout << "Indice de Produtividade:" << indiceProdutividade
242         *100.0 << "%" << endl <<
243     "Eficiencia de Fluxo:" << eficienciaFluxo*100.0 << "%"
244         << endl;
245
246 }
247
248 void CSimuladorAnaliseTestePressao::Variacao()
249 {
250     cout << "-----" <<
251         endl;
252     cout << "Deseja variar algum parametro?" << endl;
253     cout << "1-Sim|2-Nao" << endl;
254     int resp;
255     cin >> resp;
256     cin.get();
257     do
258     {
259         if (resp!=1 && resp!=2)
260         {
261             cout << "Alternativa Invalida" << endl;
262             cout << "1-Sim|2-Nao" << endl;
263             cin >> resp;
264             cin.get();
265         }
266     }
267     while (resp!=1 && resp!=2);
268     if (resp == 1)
269     {

```

```
264     cout << "\nQual parametro deseja variar?" << endl;
265     cout << "| 1 - Porosidade | 2 - Fator Volume de Formacao | 3 -
266         Compressibilidade | 4 - Viscosidade" << endl;
267     cin >> resp;
268     cin.get();
269
270     do
271     {
272         if (resp!=1 && resp!=2 && resp!=3 && resp!=4)
273         {
274             cout << "Alternativa Invalida" << endl;
275             cout << "| 1 - Porosidade | 2 - Fator Volume de Formacao | 3 -
276                 Compressibilidade | 4 - Viscosidade" << endl;
277             cin >> resp;
278             cin.get();
279         }
280     }
281     while (resp!=1 && resp!=2 && resp!=3 && resp!=4);
282     int intervalo;
283     double dp;
284     switch (resp)
285     {
286         case 1:
287             cout << "Digite um valor inicial para porosidade" << endl;
288             double porosidadeInicial;
289             cin >> porosidadeInicial;
290             cin.get();
291             while ((porosidadeInicial < 0.00) || (porosidadeInicial >
292                 1.00))
293             {
294                 cout << "Reinforme a porosidade Inicial:" << endl;
295                 cin >> porosidadeInicial;
296                 cin.get();
297             }
298             cout << "Digite um valor Final para porosidade" << endl;
299             double porosidadeFinal;
300             cin >> porosidadeFinal;
301             cin.get();
302             while ((porosidadeFinal < 0.00) || (porosidadeFinal > 1.00)
303                 || (porosidadeInicial > porosidadeFinal))
304             {
305                 cout << "Reinforme a porosidade Final:" << endl;
306                 cin >> porosidadeFinal;
```

```
302         cin.get();
303     }
304     cout << "Em quantos intervalos deseja dividir a porosidade?
305         " << endl;
306     cin >> intervalo;
307     cin.get();
308     dp=(porosidadeFinal - porosidadeInicial)/intervalo;
309     for (int x = porosidadeInicial; porosidadeInicial <=
310          porosidadeFinal; porosidadeInicial=porosidadeInicial+dp)
311     {
312         reservatorio.Porosidade(porosidadeInicial);
313         cout << "\nCalculo para porosidade:" << porosidadeInicial <<
314             endl;
315         cout << "-----" <<
316             endl;
317         CalculoPermeabilidade();
318         CalculoPressaoInicial();
319         CalculoFatorPelicula(); //guardar
320         CalculoRaioEfetivo();
321         CalculoIndices();
322         cout << "-----" <<
323             endl;
324         ostringstream strs;
325         strs << porosidadeInicial;
326         string porosidadeString = strs.str();
327         string formato = ".dat";
328         string Saida = nome+"porosidade"+porosidadeString+formato;
329         // abre arquivo
330         fout.open (Saida.c_str());
331         fout << "Permeabilidade:" << buildup->getPermeabilidade();
332         if ((reservatorio.SistemaUnidade()==141.2) || (reservatorio.
333             SistemaUnidade()==19.03))
334         fout << " milidarcy" << endl;
335         if (reservatorio.SistemaUnidade()==0.3183)
336             fout << " metros quadrados" << endl; //criacao do objeto
337             armazena da classe CSimuladorAnaliseTestePressao
338         fout << "Pressao Inicial:" << pressaoInicial;
339         if (reservatorio.SistemaUnidade()==141.2)
```

```
337         fout << "psi" << endl;
338
339     if (reservatorio.SistemaUnidade() == 19.03)
340         fout << "kgf/cm²" << endl;
341
342     if (reservatorio.SistemaUnidade() == 0.3183)
343         fout << "Pascal" << endl;
344     fout << "Fator de Pelicula:" << buildup->getPelicula() <<
345         endl;
346     fout << "Raio Efetivo:" << raioEfetivo;
347
348     if ((reservatorio.SistemaUnidade() == 0.3183) || (reservatorio.
349         SistemaUnidade() == 19.03))
350         fout << "metros" << endl;
351
352     if (reservatorio.SistemaUnidade() == 141.2)
353         fout << "ft" << endl;
354     fout << "Queda de Pressão devido ao dano:" << pressaoDano
355         << endl;
356
357     //Exibir em porcentagens
358     fout << "Indice de Produtividade:" << indiceProdutividade * 100.0
359         << "%" << endl << "Eficiencia de Fluxo:" << eficienciaFluxo
360         * 100.0 << "%" << endl;
361     fout.close();
362     cout << "Dados Salvos com sucesso!" << endl;
363     cout << "-----" <<
364         endl;
365     cin.get();
366     }
367
368     break;
369
370     case 2:
371         cout << "Digite um valor inicial para fator volume de"
372             "formacao" << endl;
373         double FatorVolformacaoInicial;
374         cin >> FatorVolformacaoInicial;
375         cin.get();
376         while ((FatorVolformacaoInicial < 0.00))
377         {
378             cout << "Reinforme o fator volume de formacao Inicial:"
379                 << endl;
```

```
371         cin >> FatorVolformacaoInicial;
372         cin.get();
373     }
374     cout << "Digite um valor Final para fator volume de
375         formacao" << endl;
376     double FatorVolformacaoFinal;
377     cin >> FatorVolformacaoFinal;
378     cin.get();
379     while ((FatorVolformacaoFinal < 0.00) || (
380             FatorVolformacaoInicial > FatorVolformacaoFinal))
381     {
382         cout << "Reinforme o fator volume de formacao Final:" <<
383             endl;
384         cin >> FatorVolformacaoFinal;
385         cin.get();
386     }
387     cout << "Em quantos intervalos deseja dividir o fator
388         volume de formacao?" << endl;
389     cin >> intervalo;
390     cin.get();
391     dp=(FatorVolformacaoFinal - FatorVolformacaoInicial)/
392         intervalo;
393     for (int x = FatorVolformacaoInicial; FatorVolformacaoInicial <=
394             FatorVolformacaoFinal; FatorVolformacaoInicial+=dp)
395     {
396         fluido.FatorVolumeFormacao(FatorVolformacaoInicial);
397         cout << "\nCalculo para fator volume de formacao:" <<
398             FatorVolformacaoInicial << endl;
399         cout << "-----"
400             << endl;
401         CalculoPermeabilidade();
402         CalculoPressaoInicial();
403         CalculoFatorPelicula(); //guardar
404         CalculoRaioEfetivo();
405         CalculoIndices();
406         cout << "-----"
407             << endl;
408
409         ostringstream strs;
410         strs << FatorVolformacaoInicial;
411         string FatorVolformacaoString = strs.str();
412
413
```

```
404     string formato = ".dat";
405     string Saida = nome+FatorVolF(+FatorVolformacaoString+formato;
406                                     // abre aquivo
407     fout.open (Saida.c_str());
408
409     fout << "Permeabilidade:" << buildup->getPermeabilidade();
410     if ((reservatorio.SistemaUnidade()==141.2) || (reservatorio.
411           SistemaUnidade()==19.03))
412         fout << " milidarcy" << endl;
413         if (reservatorio.SistemaUnidade()==0.3183)
414             fout << " metros quadrados" << endl; //criacao do objeto
415                                         armazena da classe CSimuladorAnaliseTestePressao
416     fout << "Pressao Inicial:" << pressaoInicial;
417
418
419     if (reservatorio.SistemaUnidade()==141.2)
420         fout << " psi" << endl;
421
422     if (reservatorio.SistemaUnidade()==19.03)
423         fout << " kgf/cm2" << endl;
424
425     if (reservatorio.SistemaUnidade()==0.3183)
426         fout << " Pascal" << endl;
427         fout << " Fator de Pelicula:" << buildup->getPelicula() <<
428             endl;
429         fout << " Raio Efetivo:" << raioEfetivo;
430
431     if ((reservatorio.SistemaUnidade()==0.3183) || (reservatorio.
432           SistemaUnidade()==19.03))
433         fout << " metros" << endl;
434
435     if (reservatorio.SistemaUnidade()==141.2)
436         fout << " ft" << endl;
437         fout << " Queda de Pressao devido ao dano:" << pressaoDano
438             << endl;
439
440         //Exibir em porcentagens
441         fout << " Indice de Produtividade:" << indiceProdutividade*100.0
442             << " %" << endl << " Eficiencia de Fluxo:" << eficienciaFluxo
443                 *100.0 << " %" << endl;
444         fout.close();
445         cout << " Dados Salvos com sucesso!" << endl;
446         cout << " -----" <<
```

```
        endl;
439    cin.get();
440    }
441
442    break;
443
444    case 3:
445        cout << "Digite um valor inicial para Compressibilidade" <<
446            endl;
447        double CompressibilidadeInicial;
448        cin >> CompressibilidadeInicial;
449        cin.get();
450        while ((CompressibilidadeInicial < 0.00))
451        {
452            cout << "Reinforme a compressibilidade:" << endl;
453            cin >> CompressibilidadeInicial;
454            cin.get();
455            cout << "Digite um valor Final para Compressibilidade" <<
456                endl;
457            double CompressibilidadeFinal;
458            cin >> CompressibilidadeFinal;
459            cin.get();
460            while ((CompressibilidadeFinal < 0.00) || (
461                CompressibilidadeInicial > CompressibilidadeFinal))
462            {
463                cout << "Reinforme a Compressibilidade:" << endl;
464                cin >> CompressibilidadeFinal;
465                cin.get();
466                cout << "Em quantos intervalos deseja dividir o"
467                    " Compressibilidade?" << endl;
468                cin >> intervalo;
469                cin.get();
470                dp=(CompressibilidadeFinal - CompressibilidadeInicial)/
471                    intervalo;
472                for (int x = CompressibilidadeInicial; CompressibilidadeInicial
473                    <= CompressibilidadeFinal; CompressibilidadeInicial+=dp)
474                {
475                    fluido.Compressibilidade(CompressibilidadeInicial);
476                    cout << "\nCalculo para Compressibilidade:" <<
477                        CompressibilidadeInicial << endl;
```

```
473     cout << "-----" <<
474         endl;
475             CalculoPermeabilidade();
476             CalculoPressaoInicial();
477             CalculoFatorPelicula(); //guardar
478             CalculoRaioEfetivo();
479             CalculoIndices();
480
481     cout << "-----" <<
482         endl;
483
484
485     ostringstream strs;
486     strs << CompressibilidadeInicial;
487     string CompressibilidadeString = strs.str();
488
489
490     string formato = ".dat";
491     string Saida = nome+Compressibilidade+"+"+CompressibilidadeString
492         +formato;
493             // abre arquivo
494     fout.open (Saida.c_str());
495
496
497     fout << "Permeabilidade:" << buildup->getPermeabilidade();
498     if ((reservatorio.SistemaUnidade()==141.2) || (reservatorio.
499         SistemaUnidade()==19.03))
500         fout << " milidarcy" << endl;
501     if (reservatorio.SistemaUnidade()==0.3183)
502         fout << " metros quadrados" << endl; //criacao do objeto
503             armazena da classe CSimuladorAnaliseTestePressao
504     fout << "Pressao Inicial:" << pressaoInicial;
505
506
507     if (reservatorio.SistemaUnidade()==141.2)
508         fout << " psi" << endl;
509
510
511     if (reservatorio.SistemaUnidade()==19.03)
512         fout << " kgf/cm2" << endl;
513
514
515     if (reservatorio.SistemaUnidade()==0.3183)
516         fout << " Pascal" << endl;
517     fout << " Fator de Pelicula:" << buildup->getPelicula() <<
518         endl;
519     fout << " Raio Efetivo:" << raioEfetivo;
520
521
522     if ((reservatorio.SistemaUnidade()==0.3183) || (reservatorio.
```

```
        SistemaUnidade() == 19.03))
509      fout << "metros" << endl;
510
511    if (reservatorio.SistemaUnidade() == 141.2)
512      fout << "ft" << endl;
513      fout << "Queda de Pressão devido ao dano:" << pressaoDano
514      << endl;
515
516      //Exibir em porcentagens
517      fout << "Índice de Produtividade:" << indiceProdutividade * 100.0
518      << "%" << endl << "Eficiência de Fluxo:" << eficienciaFluxo
519      * 100.0 << "%" << endl;
520      fout.close();
521      cout << "Dados salvos com sucesso!" << endl;
522      cout << "-----" <<
523          endl;
524      cin.get();
525      }
526
527      break;
528
529
530      case 4:
531      cout << "Digite um valor inicial para Viscosidade" << endl;
532      double ViscosidadeInicial;
533      cin >> ViscosidadeInicial;
534      cin.get();
535      while ((ViscosidadeInicial < 0.00))
536      {
537          cout << "Reinforme a compressibilidade:" << endl;
538          cin >> ViscosidadeInicial;
539          cin.get();
540      }
541      cout << "Digite um valor Final para Viscosidade" << endl;
542      double ViscosidadeFinal;
543      cin >> ViscosidadeFinal;
544      cin.get();
545      while ((ViscosidadeFinal < 0.00) || (ViscosidadeInicial >
546          ViscosidadeFinal))
547      {
548          cout << "Reinforme a Viscosidade:" << endl;
549          cin >> ViscosidadeFinal;
550          cin.get();
551      }
```

```
545     cout << "Em quantos intervalos deseja dividir o Viscosidade  
546         ?" << endl;  
547     cin >> intervalo;  
548     cin.get();  
549     dp=(ViscosidadeFinal - ViscosidadeInicial)/intervalo;  
550     for (int x = ViscosidadeInicial; ViscosidadeInicial <=  
551             ViscosidadeFinal; ViscosidadeInicial+=dp)  
552     {  
553         fluido.Viscosidade(ViscosidadeInicial);  
554         cout << "\nCalculo para Viscosidade:" << ViscosidadeInicial <<  
555             endl;  
556         cout << "-----" <<  
557             endl;  
558         CalculoPermeabilidade();  
559         CalculoPressaoInicial();  
560         CalculoFatorPelicula(); //guardar  
561         CalculoRaioEfetivo();  
562         CalculoIndices();  
563         cout << "-----" <<  
564             endl;  
565         ostringstream strs;  
566         strs << ViscosidadeInicial;  
567         string ViscosidadeString = strs.str();  
568         string formato = ".dat";  
569         string Saida = nome+"Viscosidade"+ViscosidadeString+formato;  
570         // abre arquivo  
571         fout.open (Saida.c_str());  
572         fout << "Permeabilidade:" << buildup->getPermeabilidade();  
573         if ((reservatorio.SistemaUnidade()==141.2) || (reservatorio.  
574             SistemaUnidade()==19.03))  
575             fout << " milidarcy" << endl;  
576         if (reservatorio.SistemaUnidade()==0.3183)  
577             fout << " metros quadrados" << endl; //criacao do objeto  
578             armazena da classe CSimuladorAnaliseTestePressao  
579         fout << "Pressao Inicial:" << pressaoInicial;
```

```
580     if (reservatorio.SistemaUnidade() == 19.03)
581         fout << "kgf/cm2" << endl;
582
583     if (reservatorio.SistemaUnidade() == 0.3183)
584         fout << "Pascal" << endl;
585     fout << "Fator de Pelicula:" << buildup->getPelicula() <<
586         endl;
587     fout << "Raio Efetivo:" << raioEfetivo;
588
589     if ((reservatorio.SistemaUnidade() == 0.3183) || (reservatorio.
590         SistemaUnidade() == 19.03))
591         fout << "metros" << endl;
592
593     if (reservatorio.SistemaUnidade() == 141.2)
594         fout << "ft" << endl;
595     fout << "Queda de Pressao devido ao dano:" << pressaoDano
596         << endl;
597
598     //Exibir em porcentagens
599     fout << "Indice de Produtividade:" << indiceProdutividade * 100.0
600         << "%" << endl << "Eficiencia de Fluxo:" << eficienciaFluxo
601         * 100.0 << "%" << endl;
602     fout.close();
603     cout << "Dados salvos com sucesso!" << endl;
604     cout << "-----" <<
605         endl;
606     cin.get();
607 }
608 void CSimuladorAnaliseTestePressao::Exporta()
609 {
610     //armazena a string digitada em nomeSaida
611
612     cout << "\nInforme o nome do arquivo de saida com os parametros
613         calculados pelo simulador:" << endl;
614     cin >> nome;
615     cin.get();
```

```

615 //      getline (cin, nome);
616     string formato = ".dat";
617     string Saída = nome+formato;
618             // abre arquivo
619     fout.open (Saída.c_str());
620
621     fout << "Permeabilidade:" << buildup->getPermeabilidade();
622     if ((reservatorio.SistemaUnidade()==141.2) || (reservatorio.
623           SistemaUnidade()==19.03))
624       fout << " milidarcy" << endl;
625     if (reservatorio.SistemaUnidade()==0.3183)
626       fout << " metros quadrados" << endl; //criacao do objeto
627           armazena da classe CSimuladorAnaliseTestePressao
628     fout << "Pressao Inicial:" << pressaoInicial;
629
630
631     if (reservatorio.SistemaUnidade()==141.2)
632       fout << " psi" << endl;
633
634     if (reservatorio.SistemaUnidade()==19.03)
635       fout << " kgf/cm²" << endl;
636
637     if (reservatorio.SistemaUnidade()==0.3183)
638       fout << " Pascal" << endl;
639     fout << "Fator de Pelicula:" << buildup->getPelicula() <<
640           endl;
641     fout << "Raio Efetivo:" << raioEfetivo;
642
643 //      if ((reservatorio.SistemaUnidade()==ESistemaUnidade::SI) ||
644 // (reservatorio.SistemaUnidade()==19.03))
645     if ((reservatorio.SistemaUnidade()==0.3183) || (reservatorio.
646           SistemaUnidade()==19.03))
647       fout << " metros" << endl;
648
649     if (reservatorio.SistemaUnidade()==141.2)
650       fout << " ft" << endl;
651     fout << "Queda de Pressao devido ao dano:" << pressaoDano
652           << endl;
653
654 //Exibir em porcentagens
655     fout << "Indice de Produtividade:" << indiceProdutividade*100.0
656           << "% " << endl << "Eficiencia de Fluxo:" << eficienciaFluxo
657           *100.0 << "% " << endl;

```

```

649     fout.close();
650     cout << "Dados salvos com sucesso!" << endl;
651     cout << "-----" <<
652         endl;
653 //Calcula os parametros da estocagem, se houver
654 void CSimuladorAnaliseTestePressao::CalculoEstocagem()
655 {
656     coeficienteEstocagem = (poco.Vazao() * fluido.
657                             FatorVolumeFormacao() * registrador.TempoSemProducao(0))
658                         / (24.0 * (registrador.PressaoMedida(0)
659                           - poco.PressaoPoco())));
660
661     tempoEstocagem = ((60.0 + 3.5 * buildup->getPelicula())/
662                         buildup->getPermeabilidade() * reservatorio.Altura())
663                         * reservatorio.SistemaUnidade() * 24.0 *
664                         coeficienteEstocagem * fluido.Viscosidade();
665
666     cout << "Coeficiente de Estocagem:" << coeficienteEstocagem
667         << endl
668         << "Tempo de Estocagem:" << tempoEstocagem;
669
670
671 }

```

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

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

```

1 #include "CSimuladorAnaliseTestePressao.h"
2
3 int main()
4 {
5     // cria o objeto simulador da classe
6     // CSimuladorAnaliseTestePressao
7     CSimuladorAnaliseTestePressao simulador;
8
9     //executa a funcao de analise do teste de pressao

```

```
9     simulador.Executar();  
10  
11     // retorna 0 se o programa rodou normalmente  
12     return 0;  
13 }
```

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 auxílio 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

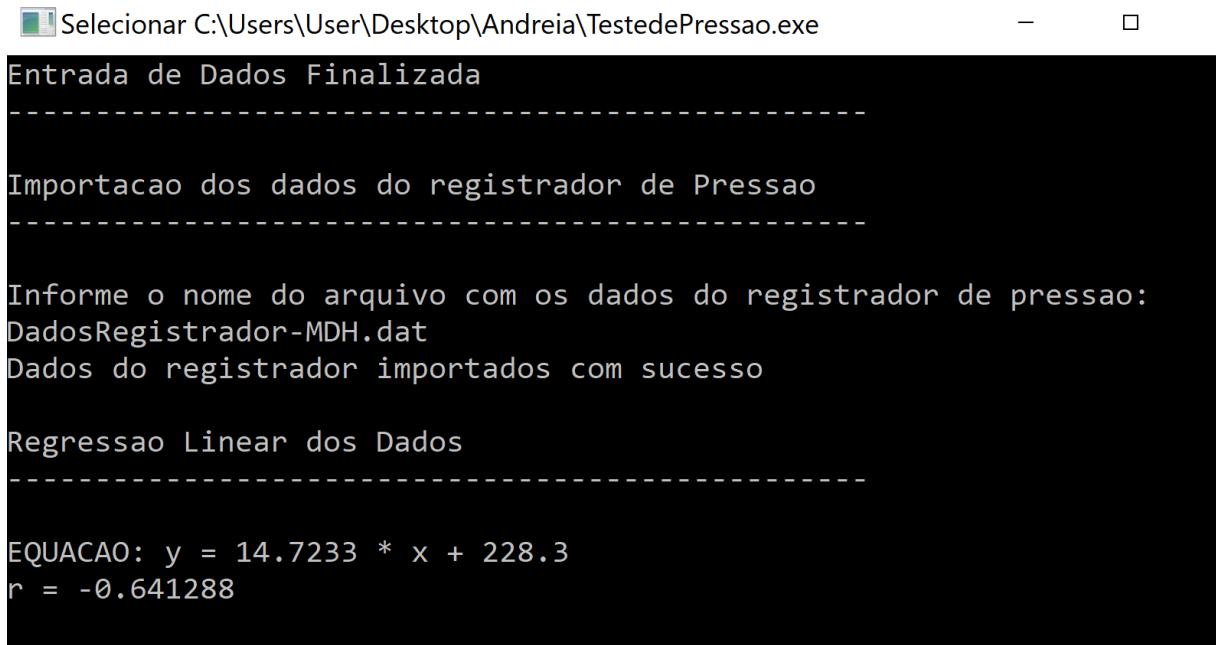
PROGRAMA PARA CALCULO DE PARAMETROS DE RESERVATORIO POR TESTES DE PRESSAO

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.

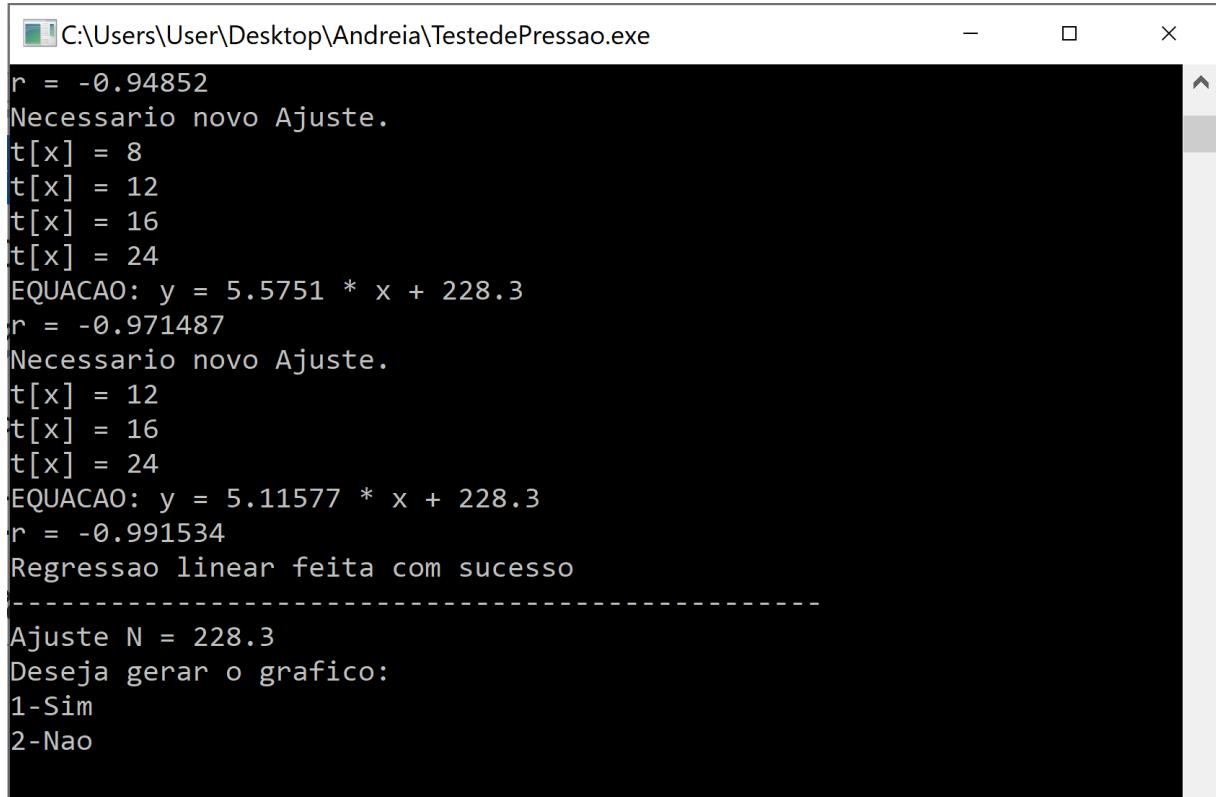


```
Selecionar C:\Users\User\Desktop\Andreia\TestedePressao.exe
Entrada de Dados Finalizada
-----
Importacao dos dados do registrador de Pressao
-----
Informe o nome do arquivo com os dados do registrador de pressao:
DadosRegistrador-MDH.dat
Dados do registrador importados com sucesso

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.

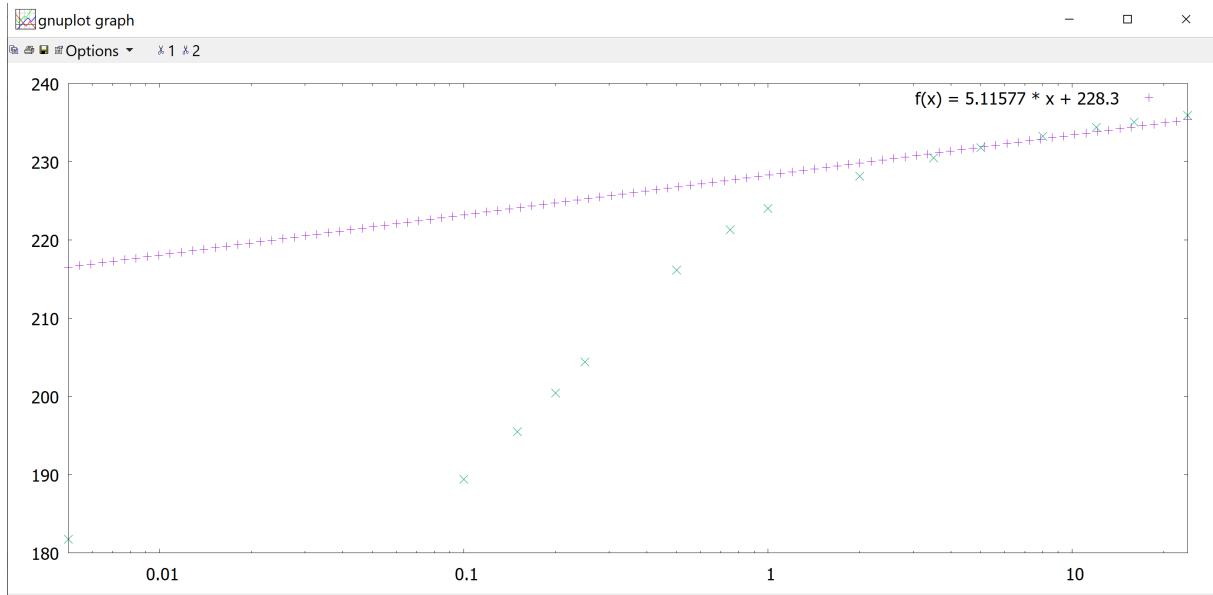


Figura 7.4: Gráfico de P_{ws} vs. $\log\Delta t$ gerado pelo Gnuplot.

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.

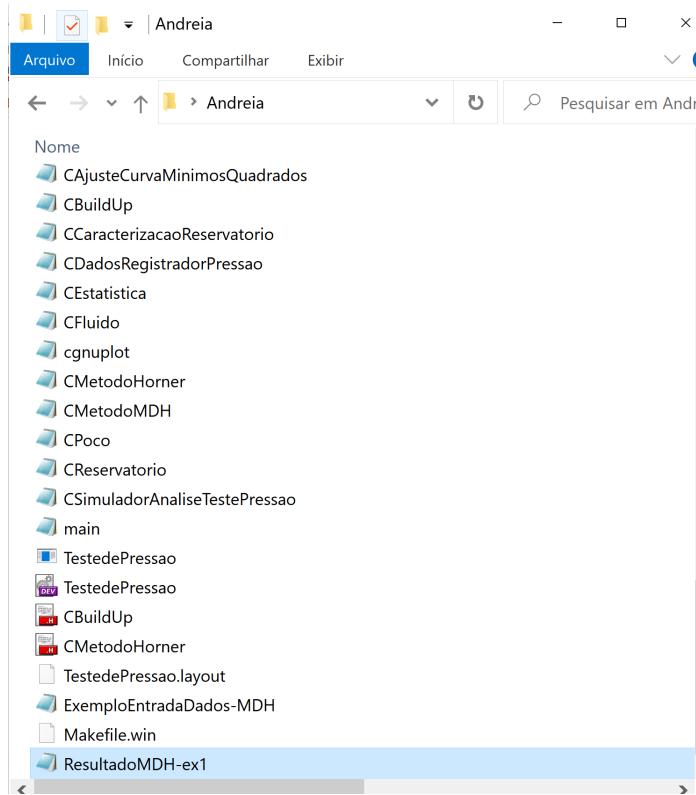
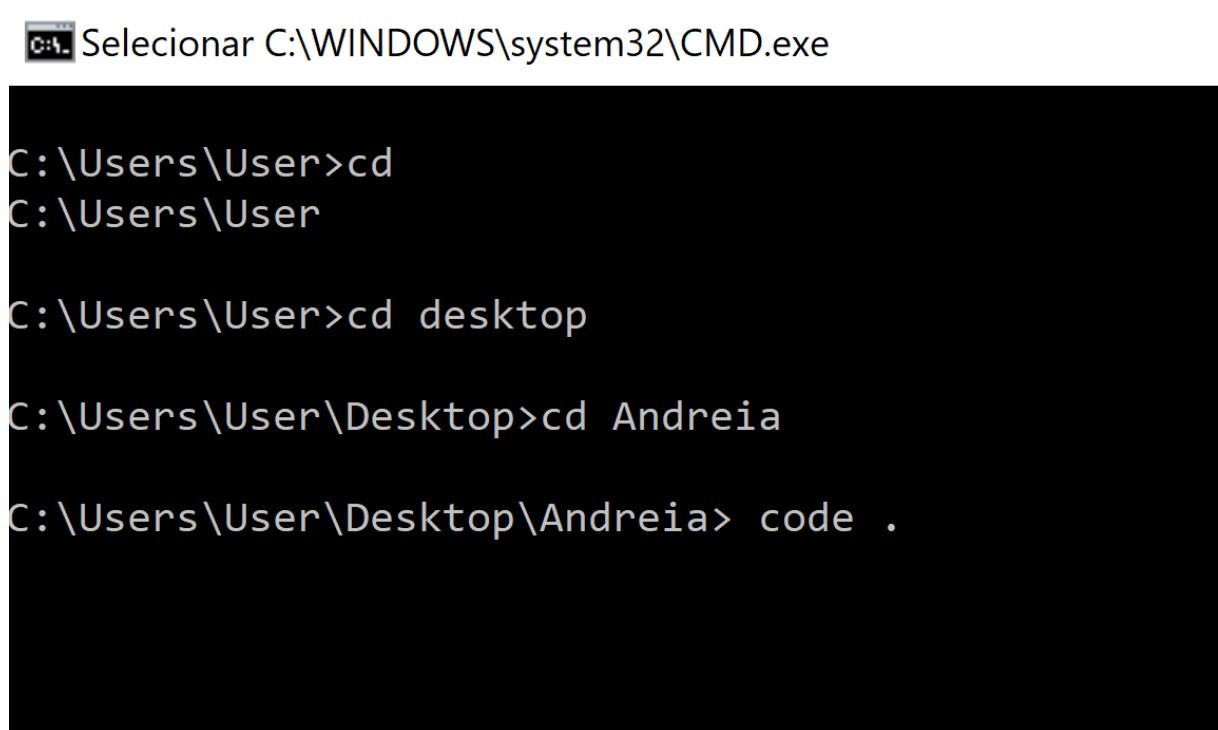


Figura 7.6: Diretório onde se encontram as listagens e onde são salvos os arquivos de saída (em destaque).

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.



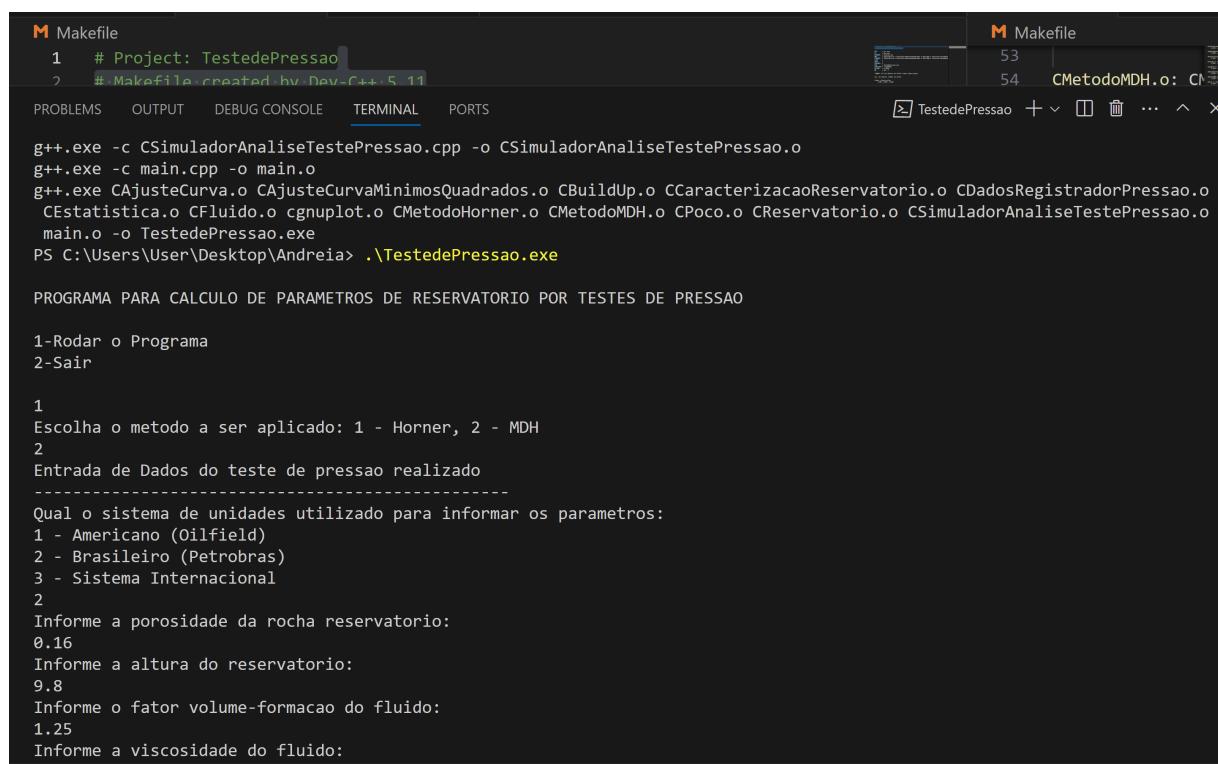
```
C:\Users\User>cd
C:\Users\User

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.



```
M Makefile
1 # Project: TestedePressao
2 # Makefile created by Dev-C++ 5.11

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
TestedePressao + × ⌂ ... ^ ×

g++.exe -c CSimuladorAnaliseTestePressao.cpp -o CSimuladorAnaliseTestePressao.o
g++.exe -c main.cpp -o main.o
g++.exe CAjusteCurva.o CAjusteCurvaMimosQuadrados.o CBuildUp.o CCaracterizacaoReservatorio.o CDadosRegistradorPressao.o
CEstatistica.o CFluido.o cgnuplot.o CMetodoHorner.o CMetodoMDH.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 Makefile Untitled-1
M Makefile
1 # Project: TestedePressao
2 #. Makefile created by Dev-C++ 5.11

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
[ ] Terminal

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.

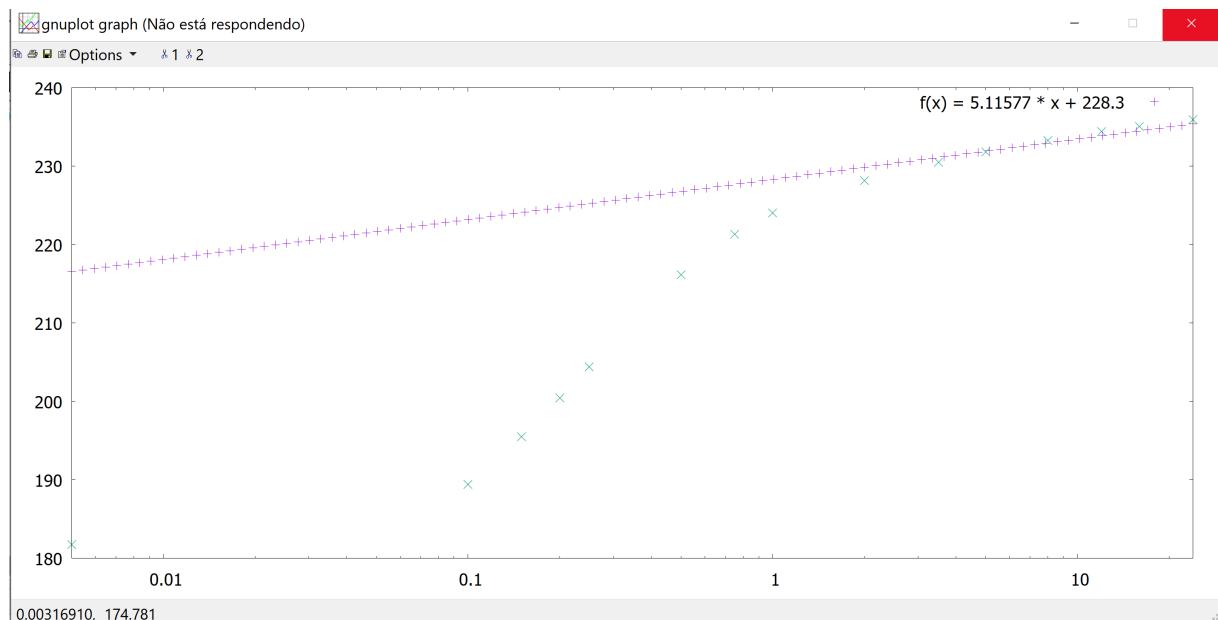


Figura 7.10: Gráfico de P_{ws} vs. $\log \Delta t$ gerado pelo Gnuplot.

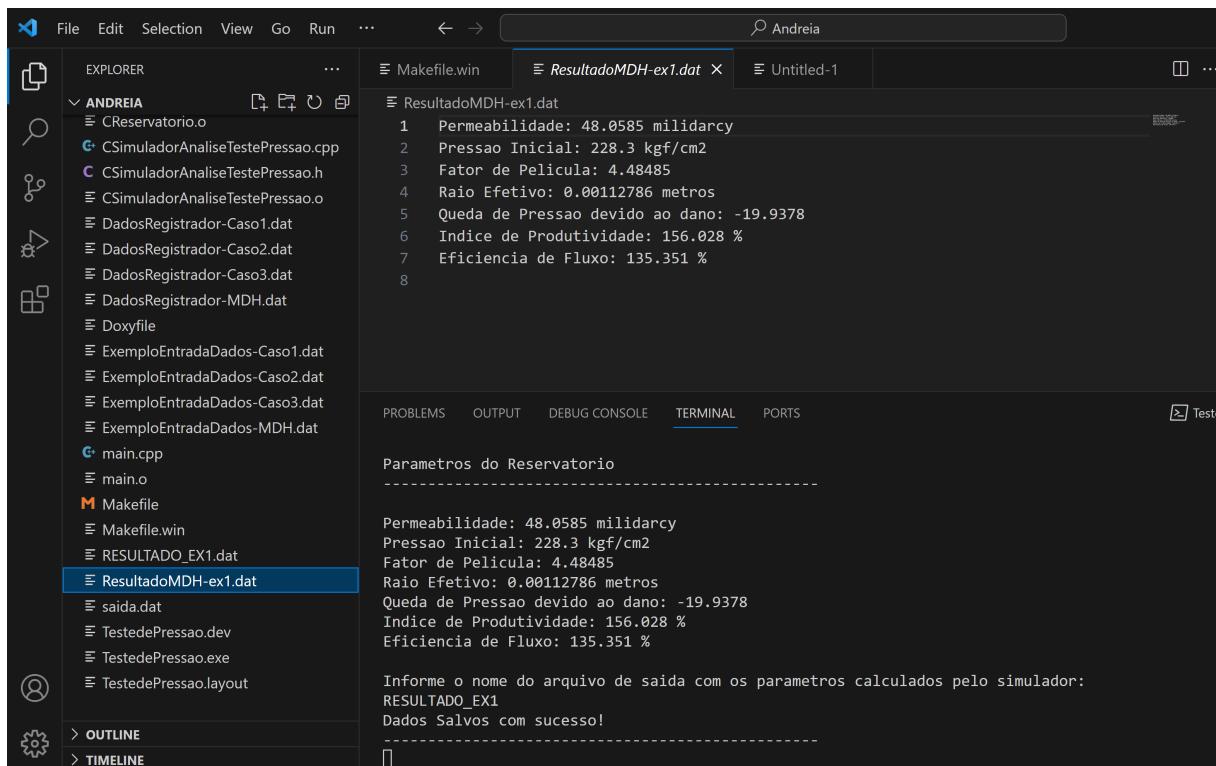


Figura 7.11: Tela do programa mostrando o nome do arquivo de saída a ser salvo, onde ele está e o seu resultado.

```

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

```

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.

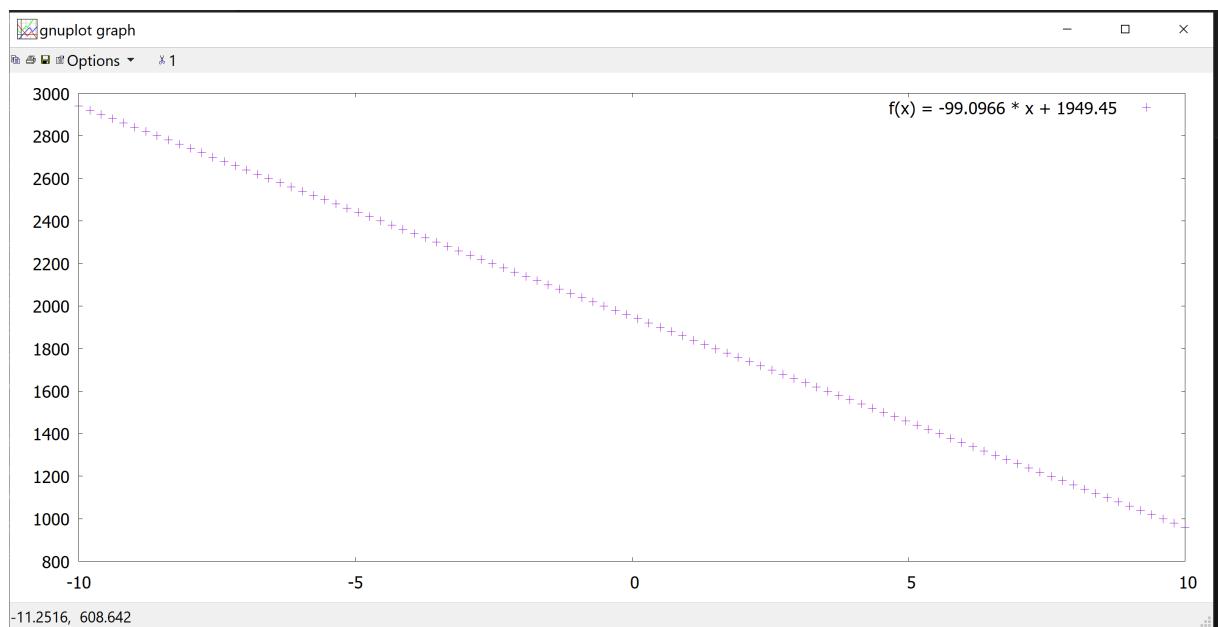


Figura 7.14: Gráfico de P_{ws} vs. $\log[(t + \Delta t)/\Delta t]$ gerado pelo Gnuplot.

```
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:  
ResultadoCaso1  
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 documentation interface. At the top, there's a header with the project name 'Projeto_engenhariaAndreia', a subtitle 'Avaliação de Formações- Teste de pressão - MDH', and navigation links for 'Página Principal', 'Classes', and 'Arquivos'. A search bar is also present. Below the header, the title 'Lista de Classes' is displayed. A note below it says 'Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:'. The main content is a table listing various classes with their descriptions:

Class	Description
<code>C AJusteCurva</code>	Classe que executa a regressao linear e ajusta ate a correlacao satisfatoria
<code>C AJusteCurvaMinimosQuadrados</code>	Classe que obtém os coeficiente da regressao linear por meio do metodo dos minimos quadrados
<code>C BuildUP</code>	
<code>C CCaracterizacaoReservatorio</code>	Classe que caracteriza o reservatorio
<code>C CDadosRegistradorPressao</code>	Classe que contem dados registrados do registrador de pressao
<code>C CEstatistica</code>	Classe que calcula estatisticas do vetor, como media e desvio padrao, util para regressao
<code>C CFluido</code>	Classe contendo as caracteristicas do fluido produzido
<code>C CMetodoHorner</code>	
<code>C CMetodoMDH</code>	
<code>C CPoco</code>	Classe contendo as caracteristicas do poco
<code>C CReservatorio</code>	Classe contendo as caracteristicas do reservatorio
<code>C CSimuladorAnaliseTestePressao</code>	Classe que faz a analise do teste de pressao realizado no campo e infere as propriedades do reservatorio
<code>C Gnuplot</code>	Classe de interface para acesso ao programa gnuplot

Figura 8.1: Lista de classes pelo doxygen.

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. 17
- [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. 17
- [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. LENEPE-UENF. 17
- [Rosa, 2006] Rosa, A. J. (2006). *Engenharia de Reservatório de Petróleo*. Interciência, Rio de Janeiro. 17
- [Rosa and Correa, 1987] Rosa, J. and Correa, A. (1987). Análise de testes de pressão em poços. *Apostila Petrobras–março*. 17
- [Steding-Jessen, 2000] Steding-Jessen, K. (2000). *Latex demo: Exemplo com Latex 2e*.

Índice Remissivo

- Análise orientada a objeto, 20
- AOO, 20
- Associações, 33
- atributos, 33
- Casos de uso, 5
- Ciclo construção, 36
- colaboração, 28
- comunicação, 28
- Concepção, 4
- Controle, 32
- Diagrama de colaboração, 28
- Diagrama de componentes, 34
- Diagrama de execução, 34
- Diagrama de máquina de estado, 29
- Diagrama de sequência, 27
- 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
- Heranças, 33
- heranças, 33
- Implementação, 36
- métodos, 33
- Mensagens, 27
- Metodologia utilizada, 2
- modelo, 33
- Plataformas, 32
- POO, 32
- Projeto do sistema, 31
- Projeto orientado a objeto, 32
- Protocolos, 31
- Recursos, 32
- Requisitos, 5
- Requisitos funcionais, 5
- Requisitos não funcionais, 5