

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

PROJETO ENGENHARIA: DESENVOLVIMENTO DO SOFTWARE
CARACTERIZAÇÃO DE RESERVATÓRIOS INTEGRADO À ANÁLISE
PETROFÍSICA
TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA

MAXIMIANO KANDA FERRAZ - VERSÃO 1 (2012/2)
GABRIEL VASCONCELOS DE SOUSA - VERSÃO 1 (2012/2)
ALEXANDRE LISBOA BASTOS JUNIOR - VERSÃO 2 (2014/2)
JOÃO VITOR MARINHO DA COSTA NEVES - VERSÃO 2 (2014/2)

Prof.: André Duarte Bueno

MACAÉ - RJ
MARÇO - 2015

Sumário

1	Introdução	1
1.1	Escopo do problema	1
1.2	Objetivos	2
2	Especificação	4
2.1	Especificação do software - requisitos	4
2.1.1	Nome do produto e componentes	4
2.1.2	Especificação	4
2.1.3	Requisitos funcionais	5
2.1.4	Requisitos não funcionais	5
2.2	Casos de uso do programa	5
2.3	Diagrama de caso de uso geral do programa	5
2.4	Diagrama de caso de uso específico do programa	7
3	Elaboração	8
3.1	Análise de domínio	8
3.1.1	Subárea da Engenharia de Reservatórios	8
3.1.2	Modelagem dos testes de pressão para reservatório	10
3.1.3	Correlação para cálculo da viscosidade	12
3.1.4	Sub-área Petrofísica	13
3.2	Identificação de pacotes – assuntos	14
3.3	Diagrama de pacotes – assuntos	14
4	AOO – Análise Orientada a Objeto	16
4.1	Diagramas de classes	16
4.1.1	Dicionário de classes	16
4.2	Diagrama de seqüência – eventos e mensagens	25
4.2.1	Diagrama de seqüência geral	25
4.2.2	Diagrama de seqüência específico	25
4.3	Diagrama de comunicação – colaboração	25
4.4	Diagrama de máquina de estado	28
4.5	Diagrama de atividades	28

5	Projeto	31
5.1	Projeto do sistema	31
5.2	Projeto orientado a objeto – POO	32
5.3	Diagrama de componentes	32
5.4	Diagrama de implantação	33
6	Implementação	34
6.1	Código fonte	34
7	Teste	79
7.1	Dados entrada Teste na plataforma GNU/Linux	79
7.2	Dados entrada Teste na plataforma Windows	81
8	Documentação	87
8.1	Documentação para Usuário	87
8.2	Documentação para desenvolvedor	89
8.2.1	Dependências	89
8.2.2	Documentação usando doxygen	90

Capítulo 1

Introdução

Existem inúmeros softwares desenvolvidos na indústria do petróleo objetivando a caracterização de um reservatório a partir de dados de poço, análises de pressão de um dado reservatório ao longo do seu tempo produtivo, sob diferentes condições de produção e vazões.

Este programa foi desenvolvido baseado em um software desenvolvido por [Ferraz, 2012]. Para tornar o programa anterior mais eficaz, o programa deve gerir melhor os dados de entrada (reduzindo-os) ou os dados de saída (expandindo-os), ambos, ou a eficiência e clareza do código em si.

O presente trabalho pretende simplificar, de forma eficiente, o *input* do programa, fazendo com que o usuário possa dispor de maior diversidade nos parâmetros de entrada, tornando as informações obtidas mais realísticas e utilizando-se de correlações comprovadamente eficientes pela literatura.

O grande diferencial do programa é permitir que o usuário avalie a formação à partir de dados do poço e que calcule a permeabilidade diretamente da amostragem de testemunho, podendo, ao final, comparar os resultados obtidos para os testes de poço e a testemunhagem. O programa também permitirá maior flexibilidade nas entradas de composição do fluido, permitindo que o usuário utilize correlações encontradas na literatura, caso necessário.

1.1 Escopo do problema

Avaliações de formações desempenham um papel fundamental na indústria do petróleo. A necessidade de determinar características de reservatórios, avaliando a disponibilidade de hidrocarbonetos e futuras previsões para determinado tempo de produção são premissas básicas para um Engenheiro de Reservatórios. Uma avaliação correta, utilizando as técnicas da análise de dados de pressão, permite que investimentos sejam corretamente alocados.

Uma outra frente importante a ser trabalhada na indústria do petróleo é a estimativa

dos *parâmetros petrofísicos*. Parâmetros como permeabilidade e porosidade são necessários para a caracterização de reservatórios de hidrocarbonetos. Um Engenheiro de Reservatórios, ao avaliar a potencialidade do reservatório, geralmente tem à sua disposição os dados permoporosos da formação. Esses dados, muitas vezes, são obtidos através de testemunhagem, com amostras submetidas à análises petrofísicas.

Na testemunhagem, para determinar os parâmetros necessários, é aplicada a lei de Boyle-Mariotte para a determinação de porosidade. Esta lei relaciona a variação do volume e pressão de um gás a uma temperatura constante, regendo o princípio do porosímetro, que determina o volume poroso pela expansão do gás dentro da amostra rochosa. A porosidade é então determinada pela razão do volume de vazios (também conhecido por volume poroso) e o volume total da amostra.

Já o permeâmetro à gás é um equipamento petrofísico usado para determinar a permeabilidade da amostra. O permeâmetro determina o fluxo de gás que atravessa a amostra, a partir do quadrado da diferença entre a pressão de entrada e a pressão de saída, que depende da área do testemunho e da viscosidade do gás que a atravessa.

A partir de entradas de parâmetros pré estabelecidos ou calculados experimentalmente pelos equipamentos supracitados, é possível, utilizando conceitos básicos de fluxo através de formações porosas, aplicar técnicas de análise de dados de pressão em poços.

1.2 Objetivos

Os objetivos deste trabalho são:

- Objetivo geral:
 - Criar um software capaz de calcular parâmetros e inferir características de um reservatório, através da análise dos dados obtidos em testes de pressão em poços e na testemunhagem do reservatório, possibilitando estimar as dimensões do campo e sua potencialidade econômica. Um diferencial importante nesta versão é a flexibilidade na entrada de dados do usuário.
- Objetivos específicos:
 - Cálculo da permeabilidade efetiva, dano de formação, índice de produtividade, efeito de estocagem.
 - Cálculo de propriedades desconhecidas dos fluidos, quando estas são desconhecidas, a partir de correlações empíricas aceitas na literatura
 - Permitir que o usuário obtenha a permeabilidade através de dados obtidos em procedimentos petrofísicos de testemunhagem
 - Permitir comparação entre a permeabilidade obtida no teste de formação com a obtida por testemunhagem

- Fornecer curvas de pressão ao longo do tempo através de um programa para plotar gráficos.

Capítulo 2

Especificação

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

2.1 Especificação do software - requisitos

2.1.1 Nome do produto e componentes

Nome	Simulador para caracterização de reservatórios integrado à análise Petrofísica
Componentes principais	Cálculo de parâmetros e caracterização de um reservatório.
Missão	Auxiliar Engenheiros de reservatório a caracterizar um reservatório e possibilitar comparação com dados petrofísicos.

2.1.2 Especificação

O projeto a ser desenvolvido consiste em um software que importe dados obtidos através de testes de pressão e de testemunhagem (somente caso o usuário opte por registrar dados de permeabilidade a partir dos testes petrofísicos) à partir de arquivos .dat e de parâmetros de *input* do usuário. O programa realizará uma regressão linear semi-logaritmica entre a pressão medida no poço versus o tempo, gerando gráficos com auxílio da classe externa Gnuplot. O usuário fornecerá através do arquivo .dat, valores iniciais de porosidade, espessura do reservatório, viscosidade (ou correlação utilizada, caso o usuário não disponha de dados de viscosidade) e fator volume de formação do fluido.

O programa será realizado utilizando a linguagem de programação orientada ao objeto C++, uma linguagem caracterizada por sua eficiência e reaproveitamento de códigos previamente desenvolvidos. A interface do programa será em modo texto, de forma a simplificar a entrada e saída de dados. Através da visualização e interpretação dos gráficos

gerados, o programa terá como saída os valores de permeabilidade efetiva, pressão inicial, raio externo do reservatório, índice de produtividade, fator película de formação e efeito de estocagem (caso estes ocorram), além dos gráficos da variação da pressão no poço versus tempo, para reservatórios de óleo e gás. Permitirá também cálculos de correlação para determinar a viscosidade de um fluido caso esta não seja determinada, além de permitir que o usuário determine se deseja obter permeabilidade a partir da testemunhagem, para efeitos de comparação.

O presente código adota licença de software livre, GPLv2, podendo ser livremente distribuído.

2.1.3 Requisitos funcionais

RF-01	O usuário deverá ter liberdade para optar se deseja comparar a permeabilidade obtida por teste de pressão com a obtida por testemunhagem.
RF-02	Deve permitir o carregamento de arquivos criados pelo software.
RF-03	Deve permitir a escolha de utilizar a correlação de Petrosky-Farshad para cálculo de viscosidade.
RF-04	O usuário poderá plotar seus resultados em um gráfico.

2.1.4 Requisitos não funcionais

RNF-01	O programa deverá ser multi-plataforma, podendo ser executado em <i>Windows</i> , <i>GNU/Linux</i> ou <i>Mac</i> .
---------------	--

2.2 Casos de uso do programa

A Tabela 2.1 apresenta um caso de uso que descreve um ou mais cenários de uso do software, exemplos de uso, como o sistema interage com usuários externos (atores) e algumas das etapas a serem seguidas, representando uma sequência típica de uso do programa. As exceções (ou erros) também devem ser apresentados, possíveis cenários onde o programa tem sua funcionalidade prejudicada.

2.3 Diagrama de caso de uso geral do programa

A Figura 2.1 demonstra como o usuário vai interagir com o programa, através de ações simples. Esse diagrama, conhecido como diagrama de caso de uso geral, mostra as tarefas executadas pelo usuário para que o programa seja executado.

Tabela 2.1: Caso de uso do programa

Nome do caso de uso:	Cálculo de parâmetros do reservatório
Resumo/descrição:	Determinação parâmetros do reservatório através de uma regressão linear com os dados do teste de pressão.
Etapas:	<ol style="list-style-type: none"> 1. Fornecer dados do reservatório 2. Fornecer dados do fluido 3. Calcular viscosidade do fluido caso não seja informada 4. Fornecer dados do poço 5. Importar dados do DadosRegistrador.dat para dados do poço 6. Gerar gráfico Variação da Pressão x Tempo 7. Optar por fazer comparação com testemunhagem 8. Caso positivo, importar arquivos para cálculo da permeabilidade 9. Exportar resultados para disco.
Cenários alternativos:	Entrada errada do usuário (por exemplo, valores conhecidos positivos entrados como negativos). O programa apresentará uma mensagem de erro neste caso, solicitando uma nova entrada.

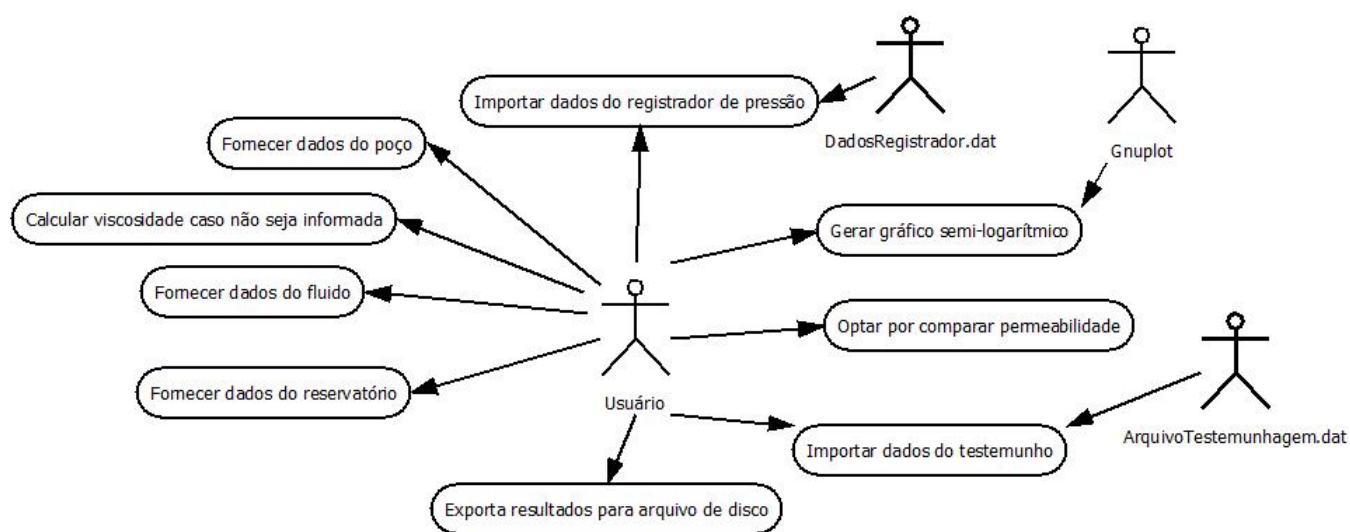


Figura 2.1: Diagrama de caso de uso geral

2.4 Diagrama de caso de uso específico do programa

A Figura 2.2 exibe a interação do usuário com o programa, porém agora detalhando a solicitação de variação de um determinado parâmetro, mostrando que o usuário deve inserir os dados necessários nos diretórios para cálculos de permeabilidade e porosidade, caso a opção de testemunhagem seja escolhida.

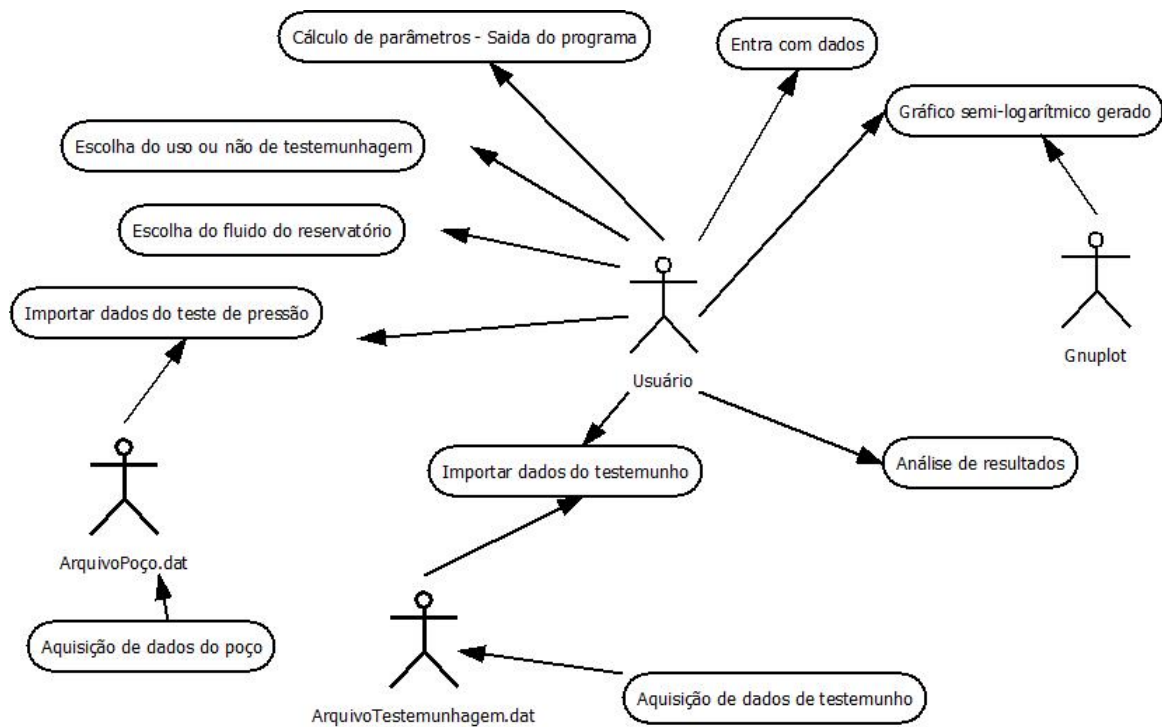


Figura 2.2: Diagrama de caso de uso específico

Capítulo 3

Elaboração

Nesta etapa será feita uma análise dos requisitos do programa, isto é, o que é necessário para que o programa seja desenvolvido. Assim, o programa é ajustado de forma a permitir que o usuário possa variar parâmetros e verificar as consequências geradas na análise de comportamento do reservatório. Permite, também, expansões futuras do programa, como exemplo incluir testes de formação específicos para reservatórios de gás, atendendo à necessidades posteriores.

3.1 Análise de domínio

O objetivo desta análise é compreender a abrangência do sistema desenvolvido pois o diagrama de pacotes representam as áreas abordadas pelo programa.

O programa a ser desenvolvido, trabalha em duas frentes principais relacionadas à área de Engenharia de Exploração e Produção de Petróleo:

- Subárea da Petrofísica: conteúdo que trata das propriedades físicas de minerais e rochas, incluindo composição matricial e espaço poroso, além dos fluidos que a percorrem;
- Subárea da Engenharia de Reservatórios: que compreende atividades como a utilização de métodos para teste de pressão, acompanhando dados de pressão em poços em função da vazão de produção. Esse acompanhamento permite determinar parâmetros de drenagem e outros fatores que interferem na produção do reservatório.

3.1.1 Subárea da Engenharia de Reservatórios

De acordo com [Adalberto José Rosa, 1987], a avaliação de formação consiste em um conjunto de atividades e estudos que têm como objetivo estimar as propriedades de uma formação (permeabilidade, fator de película, índice de produtividade, etc). Para executar o teste, o poço é completado temporariamente para permitir a produção dos fluidos contidos na formação de forma segura, em uma vazão controlada. Os passo a passo do

teste são listados a seguir e a Figura 3.1 detalha um arranjo convencional de um teste de pressão.

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.
- analisar resultados.

Segundo [Ferraz, 2012], um teste de formação tem como objetivos:

- 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.

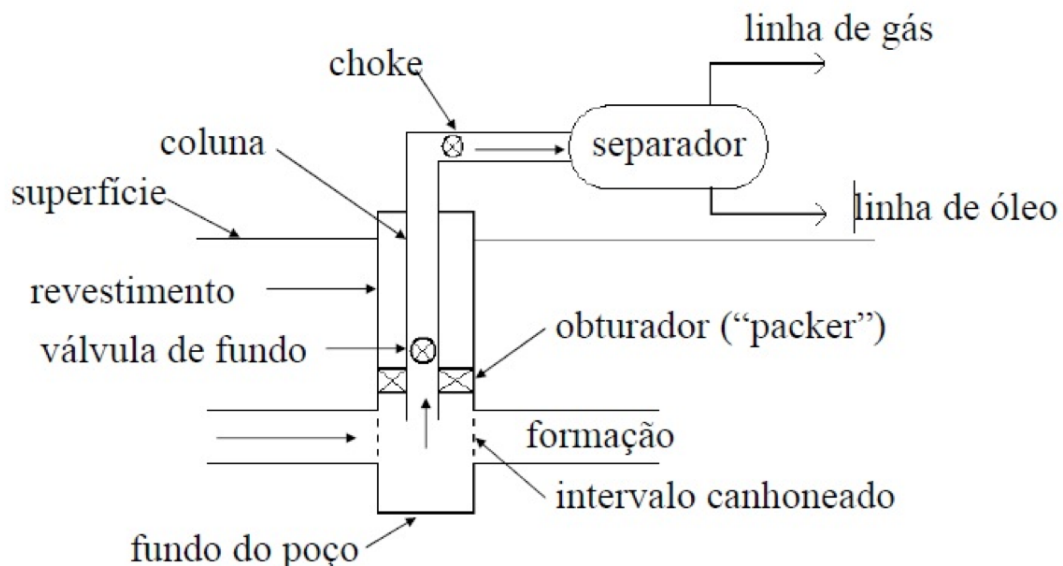


Figure 3.1: Arranjo de um teste de pressão convencional

O registrador de pressão externo registra as pressões externas à coluna, os tubos perfurados permitem a passagem dos fluidos para o interior da coluna, o obturador (*packer*) veda o espaço anular, isolando a formação da pressão hidrostática exercida pelo fluido

de amortecimento. O Registrador interno indica entupimento dos tubos perfurados, a válvula de fundo permite a abertura ou fechamento do poço e o registrador acima da válvula é utilizado para verificar vazamento da válvula durante a estática. A válvula de circulação reversa permite passagem do anular para o interior da coluna de teste. Já o estrangulador (*choke*) é um redutor de diâmetro utilizado na linha de superfície à montante do separador para limitar/controlar a vazão do poço. Os diâmetros mais utilizados são: 1/8", 1/4", 3/8" e 1/2", dados contidos em [Álvaro M. M. Peres, 2008].

3.1.2 Modelagem dos testes de pressão para reservatório

A Equação da difusividade para um fluxo monofásico e uma geometria radial é a base para explicação do comportamento apresentado por um reservatório, a mesma é dada por:

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

A equação admite inúmeras soluções. Para obter a solução para um caso particular é necessário especificar as condições iniciais e de contorno de acordo com o tipo de reservatório.

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, permitindo estimar propriedades pelo método de Horner [Adalberto Rosa, 2006], ilustrado pela 3.2.

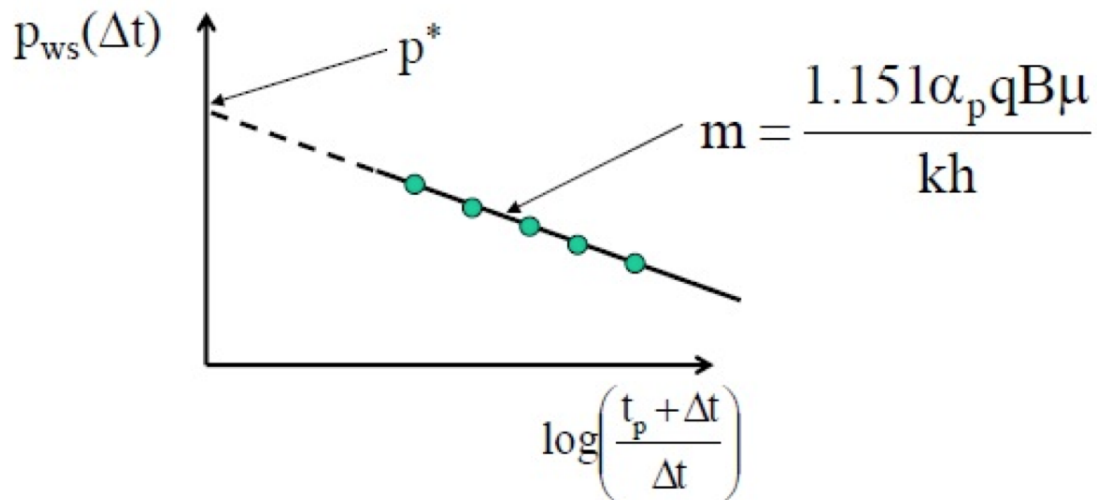


Figura 3.2: Gráfico semi-logaritimico obtido pelo Método de Horner, que fornece parâmetros referentes ao reservatório.

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 fluidos, 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, m [adimensional] é o coeficiente angular, o coeficiente α_p [adimensional] define o sistema de unidades utilizado, variando do sistema de unidades Americano ($\alpha_p = 141.2$) e Brasileiro ($\alpha_p = 19.03$). A implementação desse coeficiente no programa permite uma maior abrangência de uso ao programa. As unidades utilizadas por cada sistema estão mostradas na tabela 3.1.

Tabela 3.1: Sistemas de Unidades

Parâmetro	Americano	Petrobras
Comprimento (r_w, h)	pés (ft)	metros (m)
Vazão (q)	barris por dia ($\frac{bbl}{d}$)	metros cúbicos por dia ($\frac{m^3}{d}$)
Permeabilidade (k)	milidarcy (md)	milidarcy (md)
Fator Volume-Formação (B)	$\frac{bbl}{STB}$	$\frac{m^3}{m^3}$
Tempo ($t_p, twbs, \Delta t$)	horas (h)	horas (h)
Viscosidade (μ)	centipoises (cp)	centipoises (cp)
Pressão ($P_{wf}, P_i, \Delta P_{skin}$)	psi	$\frac{kgf}{cm^2}$
Temperatura (T)	rankine (R)	kelvin (K)

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

O fator de película S [adimensional] é definido como uma região ao redor do poço cuja permeabilidade foi alterada, reduzida (por fluido de perfuração/completação, inchamento de argilas, inversão de molhabilidade, etc) ou melhorada (através de processos de acidificação, fraturamento hidráulico, etc.) e se dá pela equação 3.3. É uma equação adimensional, onde c_t [1/psi] é a compressibilidade total, r_w [pés] é o raio do poço, t_p [dias] o tempo de produção do poço, P_{wf} [psi] é a pressão registrada no fechamento do poço, ΔP_{skin} [psi] é a queda de pressão devido ao dano 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.

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

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

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 maior efetividade 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 3.4 e 3.5, sendo P_i [psi] a pressão inicial do reservatório, indicada pelo gráfico. Valores maiores que 10 para o índice de produtividade são considerados bons.

$$IP = \frac{q}{P_i - P_{wf}} \quad (3.4)$$

$$EF = \frac{P_i - P_{wf} - \Delta P_{skin}}{P_i - P_{wf}} \quad (3.5)$$

O raio efetivo do poço r_{we} é definido como o tamanho teórico do poço incluindo o dano, calculado pela equação 3.6. Quanto maior o dano, menor o raio efetivo, pois o poço produz menos do que deveria.

$$r_{we} = r_w * e^{-s} \quad (3.6)$$

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 3.7, e sua duração, t_{wbs} [horas], pela equação 3.8.

$$C = \frac{qB\Delta t}{24(P - P_{wf})} \quad (3.7)$$

$$t_{wbs} = \frac{60.0 + 3.5 * S}{(24C * \alpha_p k h \mu)} \quad (3.8)$$

3.1.3 Correlação para cálculo da viscosidade

É comum, em muitas situações na indústria do petróleo, o usuário não dispor de todos os parâmetros de entrada. Ao explorar uma nova formação, muitas vezes há incertezas acerca dos parâmetros do fluido em questão. Para que o teste de poço apresente resultados mais realísticos (acurados), é possível utilizar correlações empíricas obtidas na literatura.

Segundo definido por [e Farshad, 1993], é possível correlacionar a viscosidade do óleo (μ) com sua temperatura (T) e sua densidade (ρ). Para efeitos de simplificação, este software supõe que o óleo a ser estudado trata-se de óleo morto, sob temperatura entre 105 F e 295 F e pressão entre 1600 a 10250 psi, o que é compatível com a finalidade do programa. A equação 3.9 de Petrosky & Farshad é apresentada à seguir:

$$\mu = (2.3511 \cdot 10^7) * (T^{-2.10255}) * (\log_{10} API)^X \quad (3.9)$$

Onde o parâmetro X é definido na equação à seguir:

$$X = (4.59388 \cdot \log_{10} T) - 22.82792 \quad (3.10)$$

As unidades utilizadas nesta correlação são: Temperatura (Fahrenheit), densidade (API) e a viscosidade é resultada em cp.

3.1.4 Sub-área Petrofísica

A petrofísica estuda aspectos teóricos e experimentais referentes à determinação das propriedades físicas das rochas, bem como as causas de suas variações no tempo e no espaço. O trabalho foca na obtenção do parâmetro de permeabilidade através da utilização de um permeâmetro.

A permeabilidade obtida através de um teste petrofísico é pontual, para um intervalo de uma amostra coletada por um processo conhecido por testemunhagem. O software fará uma simulação direta do funcionamento de um permeâmetro. O objetivo será comparar a permeabilidade obtida na testemunhagem com àquela obtida no teste de formação.

O permeâmetro a gás (ilustrado na Figura 3.3) é utilizado para determinar o fluxo de gás que passa pela amostra em função da diferença quadrática entre a pressão de entrada (P_u) e a pressão de saída (P_d). A equação 3.11 relaciona a área da amostra (A) e a viscosidade (μ) do gás. O funcionamento deste equipamento pode ser comparado com os do experimento de Darcy, com a única diferença que no permeâmetro se utiliza um fluxo de gás, não de água.

$$Q = \frac{Ak}{2000\mu P_{atm}} = (P_u^2 - P_d^2) \quad (3.11)$$

Onde: Q [cm³/s] é a vazão; A [m²] a área; μ [cp] a viscosidade; P_{atm} [atm] é a pressão atmosférica; k [mD] é a permeabilidade da amostra; P_u [atm] a pressão de entrada e P_d [atm] a pressão de saída no permeâmetro.

O porosímetro é um equipamento petrofísico utilizado para medir a porosidade em laboratório, a partir de amostras de rochas obtidas em testemunhagem. Como o programa não simula diretamente a presença de um porosímetro, o usuário deve entrar com o dado da porosidade da zona de interesse (reservatório) para que o programa simule o teste da formação.

O Engenheiro de Reservatórios deve ter acesso à porosidade para que possa executar os testes, mostrando mais uma vez a complexidade desta área e a necessidade de interação entre os diferentes setores da Engenharia de Petróleo.

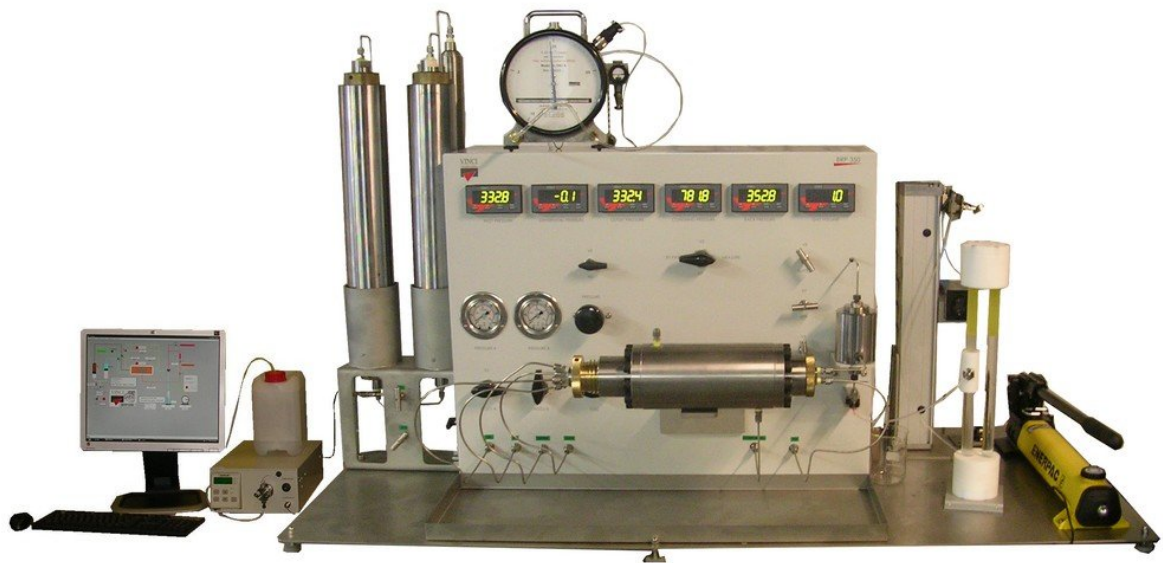


Figura 3.3: Exemplo de um permeômetro a gás

3.2 Identificação de pacotes – assuntos

- Pacote DadosPoço: importa os dados do teste de pressão de um arquivo .dat e apresenta conceitos utilizados para estimar parâmetros do reservatório.
- Pacote DadosTestemunho: importa os dados do testemunho de um arquivo .dat e apresenta conceitos utilizados para estimar parâmetros do reservatório.
- Pacote AjusteCurva: realiza a regressão linear dos dados passados no arquivo .dat. Deve apresentar conexão com o Pacote DadosPoço, uma vez que recebe os dados contidos nele. Pode receber dados contidos no Pacote DadosTestemunho caso as informações sejam inseridas no arquivo .dat correspondente.
- Pacote Caracterizacao: gera os gráficos com a curva semi-logaritmica dos dados da regressão linear feita, plota os dados de entrada da pressão versus tempo, avalia os resultados através de um modelo para caracterização.
- Pacote SimuladorPropriedades: composto de diversas equações, calcula propriedades que podem ser obtidas pela curva semi-logaritmica. Os resultados dessas equações são parte da saída do programa.

3.3 Diagrama de pacotes – assuntos

Um diagrama de pacotes é útil para mostrar as dependências entre as diversas partes do sistema. Pode incluir: sistemas, subsistemas, colaborações, casos de uso e componentes.

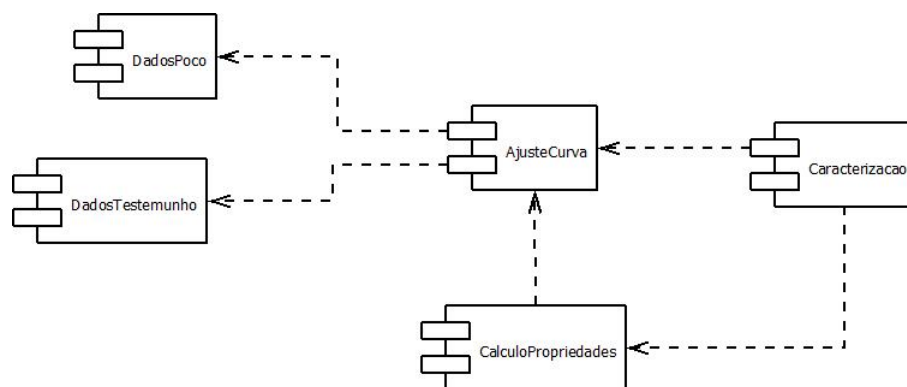


Figura 3.4: Diagrama de pacotes mostra as dependências entre os diversos pacotes do sistema

Capítulo 4

AOO – Análise Orientada a Objeto

A Análise Orientada a Objeto (AOO) é a etapa de desenvolvimento de um software que 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 e 4.2.

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 raioPoco [ft ou m] : referente ao raio do poço.
 - atributo vazao [$\frac{bbl}{d}$ ou $\frac{m^3}{d}$] : referente à vazão de produção.
 - atributo tempoProducao [$horas$]: referente ao tempo de produção.
 - atributo pressaoPoco [psi ou $\frac{kgf}{cm^2}$]: referente à pressão no poço.
 - 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.



Figura 4.1: Diagrama de Classes - Parte 1

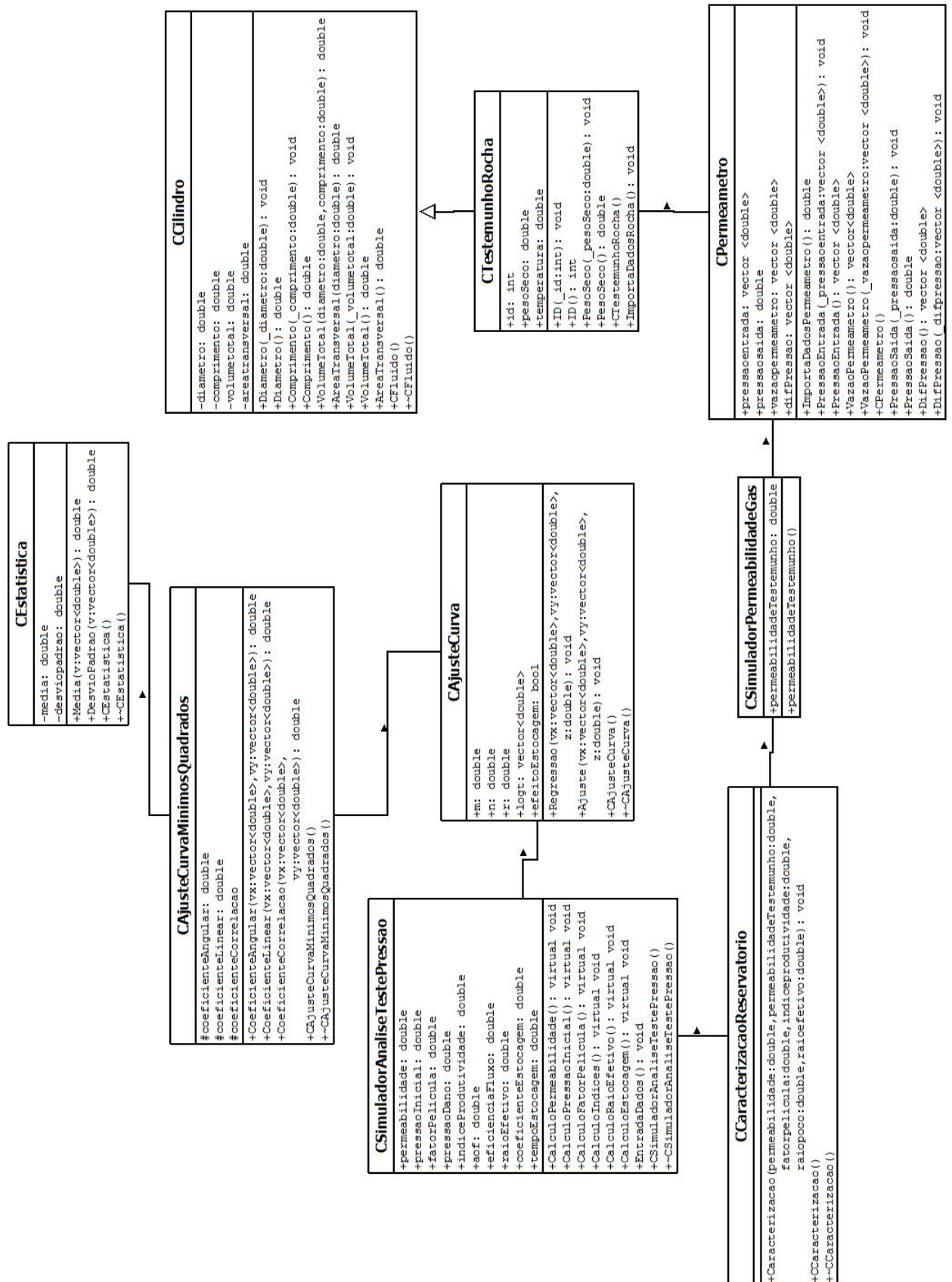


Figura 4.2: Diagrama de Classes - Parte 2

- 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 [*adimensional*] : referente à porosidade da rocha reservatório.
 - atributo altura [*ft ou m*] : referente ao intervalo do reservatório.
 - atributo sistemaUnidade : referente ao sistema de unidades escolhido.
 - 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 Altura (_altura): Método que seta o valor do atributo altura.
 - método Altura (): Método que retorna o valor do atributo altura.
 - 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 Porosidade (_porosidade): Método que seta o valor do atributo porosidade.
 - método Porosidade (): Método que retorna o valor do atributo porosidade.
 - método SistemaUnidade (_sistemaUnidade): Método que seta o valor do atributo sistemaUnidade.
 - método SistemaUnidade (): Método que retorna o valor do atributo sistemaUnidade.
- Classe CFluido: Classe que possui as características/atributos do fluido, e tem uma função de entrada de dados por parte do usuário.
 - atributo viscosidade [*cp*] : referente à viscosidade do fluido.

- atributo compressibilidade [psi^{-1} ou $\frac{kgf}{cm^2}^{-1}$] : referente à compressibilidade do fluido.
 - atributo fatorVolumeFormacao [$\frac{bbl}{STB}$ ou $\frac{m^3}{m^3std}$] : referente ao fator volume-formação do fluido.
 - atributo temperaturafluido [*Fahrenheit*]: referente à temperatura do fluido.
 - 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 Viscosidade (_viscosidade): Método que seta o valor do atributo viscosidade.
 - método Viscosidade (): Método que retorna o valor do atributo viscosidade.
 - método ViscosidadeCorrelacao () : Método que calcula a viscosidade de óleo morto através de correlação Petrosky & Farshad se a mesma não for informada.
 - método Compressibilidade (_compressibilidade): Método que seta o valor do atributo compressibilidade.
 - método Compressibilidade (): Método que retorna o valor do atributo compressibilidade.
 - método FatorVolumeFormacao (_fatorVolumeFormacao): Método que seta o valor do atributo fatorVolumeFormacao.
 - método FatorVolumeFormacao (): Método que retorna o valor do atributo fatorVolumeFormacao.
- 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 [psi ou $\frac{kgf}{cm^2}$].
 - atributo tempoSemProducao [*horas*].
 - método Importa (): Método que preenche os vetores com dados de pressão medida e tempo sem produção.
 - 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 `desviopadrao`.
 - método `Media (v)`: Retorna a média do vetor `v`.
 - método `DesvioPadrao (v)`: Retorna o desvio padrão do vetor `v`.
- Classe `CAjusteCurvaMinimosQuadrados`: Classe que faz a regressão linear através do método dos mínimos quadrados.
 - atributo `coeficienteAngular`.
 - atributo `coeficienteLinear`.
 - atributo `coeficienteCorrelacao`.
 - 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`.
- 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 CGnuplot: Classe que possibilita a geração de gráficos usando o programa externo Gnuplot.
- 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 [mD].
 - atributo pressaoInicial [psi ou $\frac{kgf}{cm^2}$].
 - atributo fatorPelicula [*adimensional*].
 - atributo pressaoDano [psi ou $\frac{kgf}{cm^2}$].
 - atributo indiceProdutividade [*adimensional*].
 - atributo eficienciaFluxo [*adimensional*].
 - atributo raioEfetivo [ft ou m].
 - atributo coeficienteEstocagem [*adimensional*].
 - atributo tempoEstocagem [*horas*].
 - método CalculoPermeabilidade (): Função que calcula e exibe a permeabilidade do reservatório.
 - método CalculoPressaoInicial (): Função que calcula e exibe a pressão inicial pela extrapolação da reta.
 - método CalculoFatorPelicula (): Função que calcula e exibe o fator de película do reservatório e a queda de pressão devido à esse fator.
 - método CalculoIndices (): Função que calcula e exibe o índice de produtividade do reservatório e a eficiência de fluxo.
 - método CalculoRaioEfetivo (): Função que calcula e exibe o raio efetivo.
 - método CalculoEstocagem (): Função que calcula e exibe a estocagem.
 - método EntradaDados (): Método que pede ao usuário os parâmetros necessários para o programa.

- Classe CCaracterizacaoReservatorio: Classe que caracteriza o reservatório, interpretando os resultados obtidos.
 - método Caracterizacao (permeabilidade, fatorPelicula, indiceProdutividade, raioPoco, raioEfetivo): Método que analisa os resultados e informa ao usuário a qualidade do reservatório submetido ao teste de pressão.
- Classe CSimuladorPermeabilidadeGas : Classe que calcula a permeabilidade através do permeâmetro à gás em um testemunho rochoso.
 - atributo permeabilidade [mD];
 - método permeabilidade () : Método para cálculo da permeabilidade através do permeâmetro em um testemunho;
 - método ComparaPermeabilidades (k_reservatorio) : Método para comparar permeabilidade obtida do teste de poço com amostra de rocha;
- Classe CGasPermeametro: classe utilizada para importar os dados de viscosidade do gás utilizado no permeâmetro.
 - atributo viscosidade [cp]: valor da viscosidade do fluido (gás);
 - método ImportaDadosFluido(): método que importa dados de um arquivo.dat;
 - método Viscosidade(_viscosidade): utilizado para setar o valor do atributo viscosidade;
 - método Viscosidade(): utilizado para retornar o valor do atributo viscosidade;
- Classe CTestemunhoRocha: representa os parâmetros referentes à amostra.
 - atributo id: nome da amostra.
 - atributo pesoseco [g]: valor do peso seco da amostra.
 - atributo temperatura [$^{\circ}C$]: representa o valor da temperatura durante a realização das medidas.
 - método ID (_id): utilizado para setar o atributo id.
 - método ID () : utilizado para retornar o valor do atributo id.
 - método PesoSeco (_peso_seco) : utilizado para setar o atributo pesoSeco.
 - método PesoSeco () : utilizado para retornar o valor do atributo pesoSeco.
 - método ImportaDadosRocha () : utilizado para importar dados da rocha de um arquivo .dat.
- Classe CCilindro: representa os atributos e métodos referentes a um cilindro.

- atributo diametro [m] : valor do diâmetro de um cilindro;
 - atributo comprimento [m]: valor do comprimento de um cilindro;
 - atributo volumeTotal [m^3]: valor referente ao volume de um cilindro;
 - atributo areaTransversal [m^2]: valor referente à área transversal de um cilindro;
 - método Diametro(_diametro): utilizado para setar o atributo diametro;
 - método Diametro(): utilizado para retornar o valor do atributo diametro;
 - método Comprimento(_comprimento): utilizado para setar o atributo comprimento;
 - método Comprimento(): utilizado para retornar o valor do atributo comprimento;
 - método VolumeTotal(_volumeTotal): utilizado para setar o atributo volumeTotal da amostra;
 - método VolumeTotal(): utilizado para retornar o valor do atributo volume;
 - método AreaTransversal(): utilizado para retornar o valor do atributo AreaTransversal;
 - método AreaTransversal (_diametro, _comprimento): método que calculará o valor do atributo areaTransversal;
 - método VolumeTotal (_diametro, _comprimento): método que calculará o valor do atributo volumeTotal;
- Classe CPermeametro: representa os atributos adquiridos através de medidas no permeâmetro à gás.
 - atributo pressaoEntrada: vetor que armazena os diversos valores da pressão de entrada setada no equipamento;
 - atributo pressaoSaida: representa a pressão de saída;
 - atributo vazaoPermeametro: vetor que armazena os diversos valores da vazão medida no equipamento;
 - atributo difPressao: vetor que armazena as diferenças de pressão registradas;
 - método ImportaDadosPermeametro (): método que irá importar os dados e calcular o valor das diferenças de pressões de entrada e saída.
 - método PressaoEntrada (_pressaoEntrada): utilizado para setar o atributo pressaoentrada.
 - método PressaoEntrada (): utilizado para retornar o valor do atributo pressaoentrada;

- método `PressaoSaida (_pressaoSaida)`: utilizado para setar o atributo `pressaoSaida`;
- método `PressaoSaida ()`: utilizado para retornar o valor do atributo `pressaoSaida`;
- método `VazaoPermeametro (_vazao)`: utilizado para setar o atributo `vazaopermeametro`;
- método `VazaoPermeametro ()`: utilizado para retornar o valor do atributo `vazaopermeametro`;
- método `DifPressao(_difPressao)`: utilizado para setar o atributo `difPressao`;
- método `DifPressao()`: utilizado para retornar o valor do atributo `difPressao`;

4.2 Diagrama de seqüência – eventos e mensagens

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

4.2.1 Diagrama de seqüência geral

Veja o diagrama de seqüência geral na Figura 4.3.

4.2.2 Diagrama de seqüência específico

Veja o diagrama de seqüência específico na Figura 4.4.

4.3 Diagrama de comunicação – colaboração

Veja na Figura 4.5 o diagrama de comunicação. A entrada de dados fornece os atributos para as classes/objetos de `CPoco`, `CReservatorio` e `CFluido`. Já a classe `CDadosRegistradorPressao` e `CPermeametro` importam seus dados de arquivos de texto. A classe `CDadosRegistradorPressao` passa os dados para a classe `CAjusteCurva`, que utiliza a classe `CAjusteCurvaMinimosQuadrados` para fazer a regressão de dois vetores usando a média e o desvio padrão obtidos por `Cestatistica`. A função de ajuste encontra coeficientes de estocagem, enquanto `CSimuladorAnaliseTestePressao` faz cálculos dos parâmetros do reservatório com os dados entrados pelo usuário e pela importação de arquivo `DadosRegistrador.dat`. A classe `CCaracterizacao` caracteriza o reservatorio com a função `Caracterizacao` que calcula os atributos permeabilidade, `fatorPelícula`, `indiceProdutividade`,

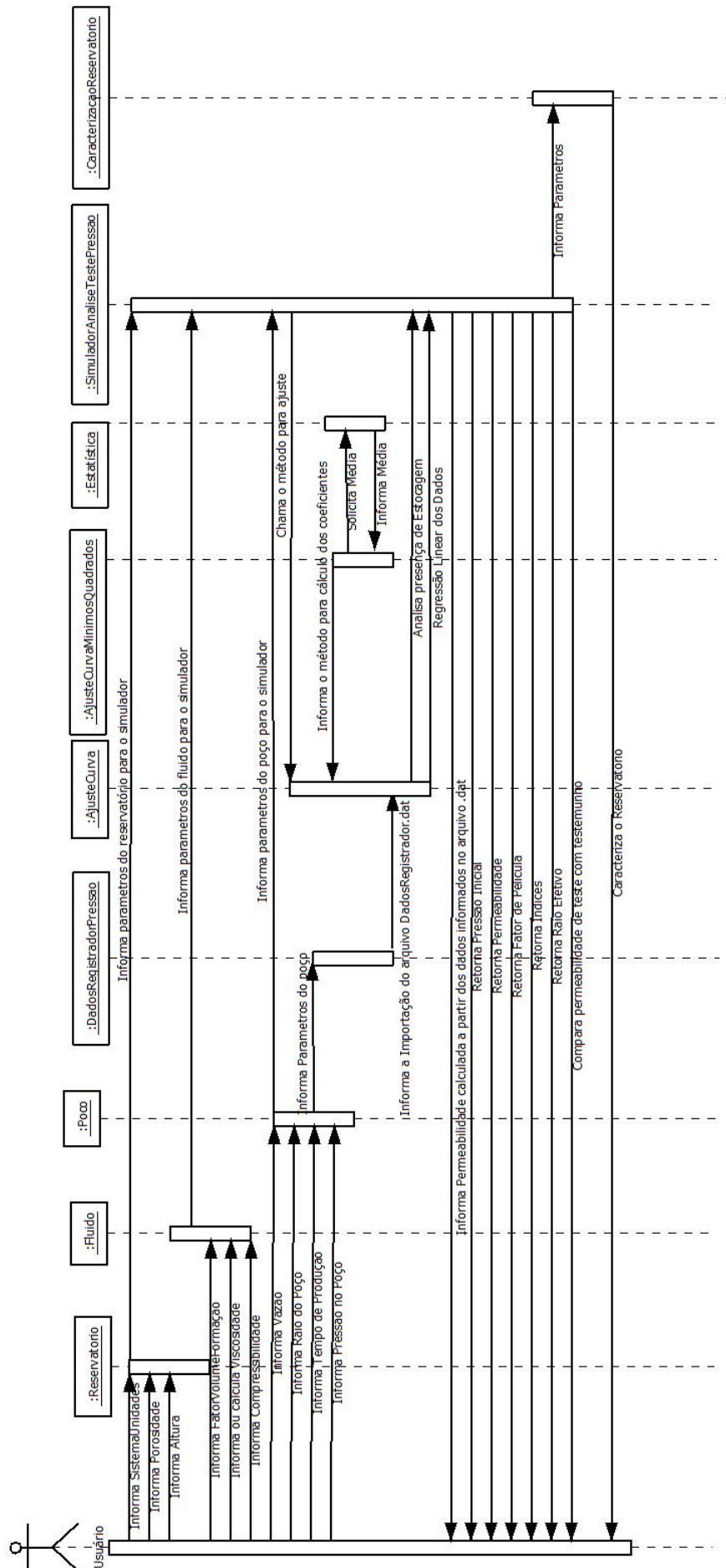


Figura 4.3: Diagrama de sequência geral, mostrando a ordem temporal e sequencial dos eventos

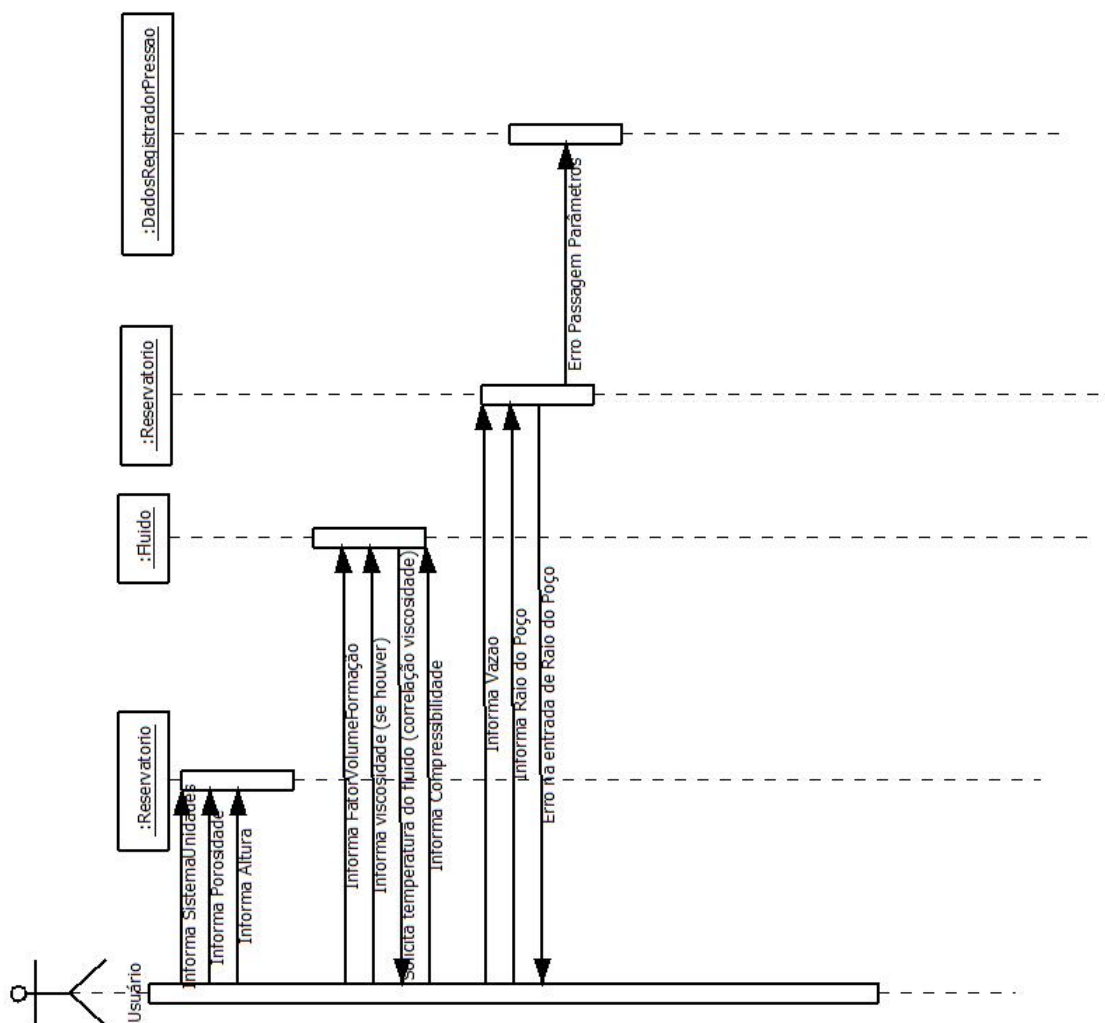


Figura 4.4: Diagrama de sequência específico mostra de forma mais detalhada uma parte do programa

raioPoco, raioEfetivo. A classe CSimuladorPermeabilidadeGas compara a permeabilidade obtida pelo testemunho e a permeabilidade obtida pelo teste de poço, explicando as diferenças ou semelhanças dos resultados obtidos.

4.4 Diagrama de máquina de estado

Veja na Figura 4.6 o diagrama de máquina de estado para o objeto da classe CSimuladorAnaliseTestePressao. Observe que o objeto possui atributos informados pelo usuário na seleção do parâmetro a ser variado, e como será feita tal variação.

4.5 Diagrama de atividades

Na Figura 4.7, o diagrama de atividades do programa mostra que, os atributos altura e porosidade do objeto da classe CReservatorio (informado pelo usuário) deve seguir requisito para que sua entrada seja correta. Caso haja erro na entrada (porosidade inferior à 0 ou superior a 1 e altura inferior a 0), o programa pede uma nova entrada de porosidade ou altura.

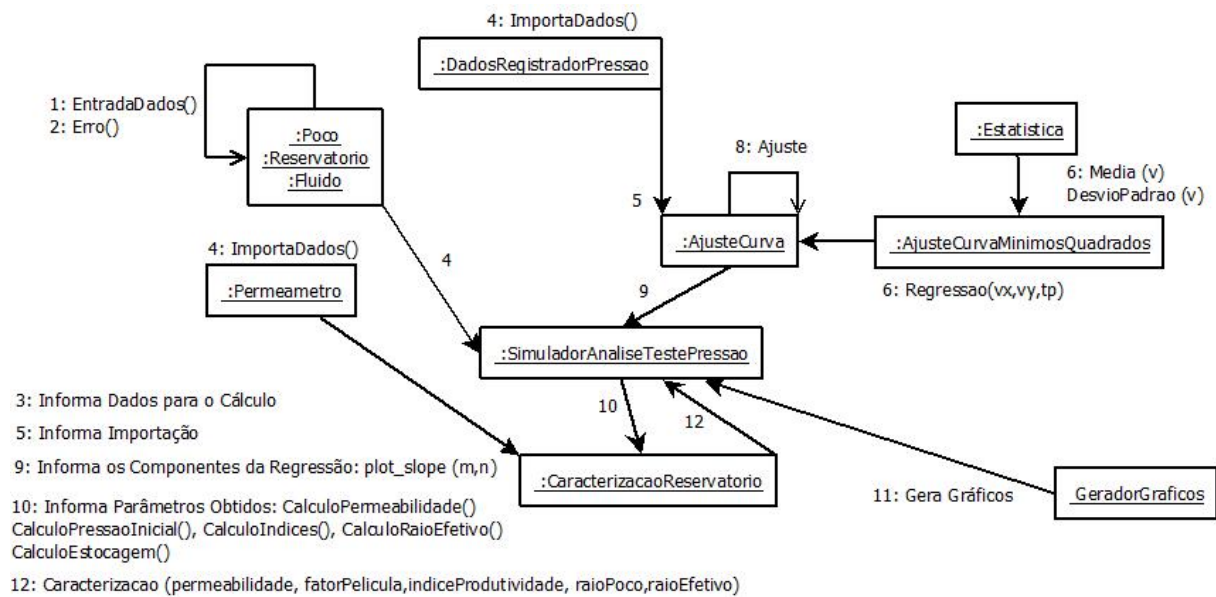


Figura 4.5: Diagrama de comunicação, que mostra o conjunto de objetos e seus relacionamentos, incluindo as mensagens trocadas entre eles

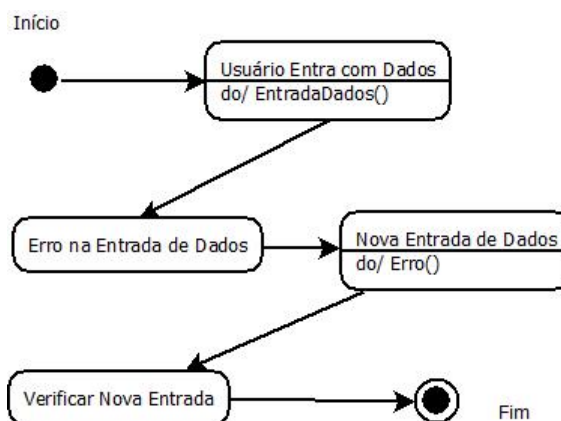


Figura 4.6: O diagrama de máquina de estado mostra os diversos estados que o objeto assume e os eventos que ocorrem ao longo do processo., modelando aspectos dinâmicos do objeto

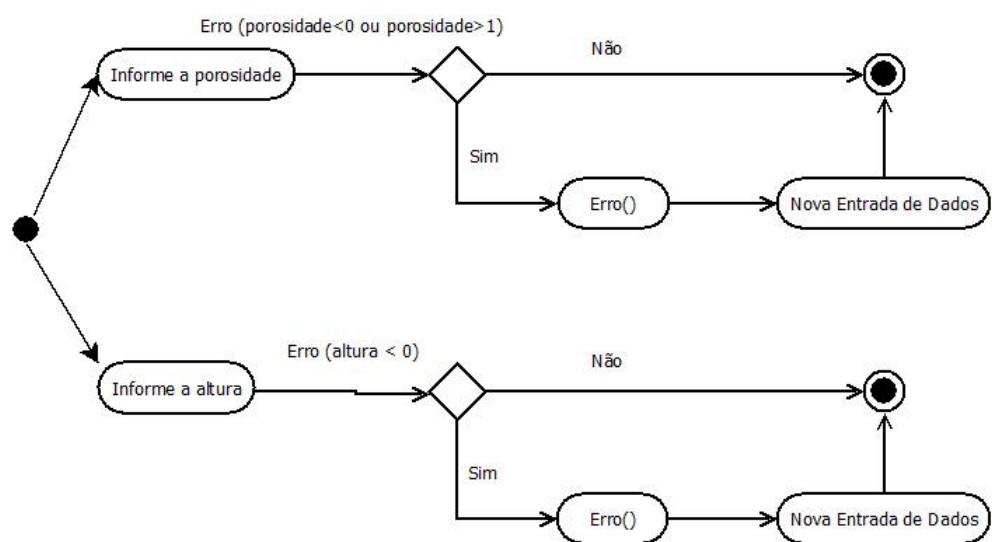


Figura 4.7: Diagrama de atividades, mostrando o fluxo de controle do método CReservatorio::Erro()

Capítulo 5

Projeto

Esse capítulo define o projeto do software em si, avaliando as plataformas à serem suportadas, os protocolos, recursos e interfaces utilizadas, associação à bibliotecas externas, subdivisão em hardwares, entre outros. A análise do projeto visa otimizar a estrutura do programa e otimizar os tempos de execução, memória, custos e desenvolver a estrutura dos dados.

5.1 Projeto do sistema

Após a Análise Orientada à Objeto:

1. Protocolos

- Definição da interface API de suas bibliotecas e sistemas
 - O programa utilizará bibliotecas C/C++
- Definição do formato dos arquivos de entrada pelo programa.
 - O programa terá como entrada arquivos de extensão .dat

2. Recursos

- Identificação e alocação dos recursos globais, como os recursos do sistema serão alocados, utilizados, compartilhados e liberados. Implicam modificações no diagrama de componentes
 - As diferentes funções acessarão os dados privados, e o usuário terá acesso aos dados calculados.

3. Controle

- Identificação da necessidade de otimização. Por exemplo: prefira sistemas com grande capacidade de memória; prefira vários hds pequenos a um grande.

- Os cálculos realizados requerem pouco espaço na memória, não havendo necessidade de otimização neste sentido.

4. Plataformas

- Identificação e definição das plataformas a serem suportadas: hardware, sistema operacional e linguagem de programação.
 - Software multiplataforma (Windows, Linux, iOS). Linguagem: C++.
- Seleção das bibliotecas externas a serem utilizadas.
 - Programa gerador de gráficos definido como o Gnuplot, por ser de código livre.
- Seleção do ambiente de desenvolvimento para montar a interface de desenvolvimento – IDE.
 - Programas de software livre *XCode Dev-C++*.

5.2 Projeto orientado a objeto – POO

Efeitos do projeto no modelo estrutural

- Estabelecer as dependências e restrições associadas à plataforma escolhida.
 - Caso o Gnuplot não esteja instalado no Windows, uma mensagem de erro será exibida.

Efeitos do projeto nas associações

- A classe Gnuplot foi associada à CSimuladorAnaliseTestePressao.

5.3 Diagrama de componentes

O diagrama de componentes mostra a forma como os componentes do programa se relacionam, suas dependências. Veja na Figura 5.1 o diagrama de componentes. A geração dos objetos depende dos arquivos de classe de extensão `.h` e `.cpp`. O subsistema banco de dados representa o arquivo que o programa importará os dados a serem manipulados. O programa executável a ser gerado depende das bibliotecas, dos arquivos desta e do banco de dados.

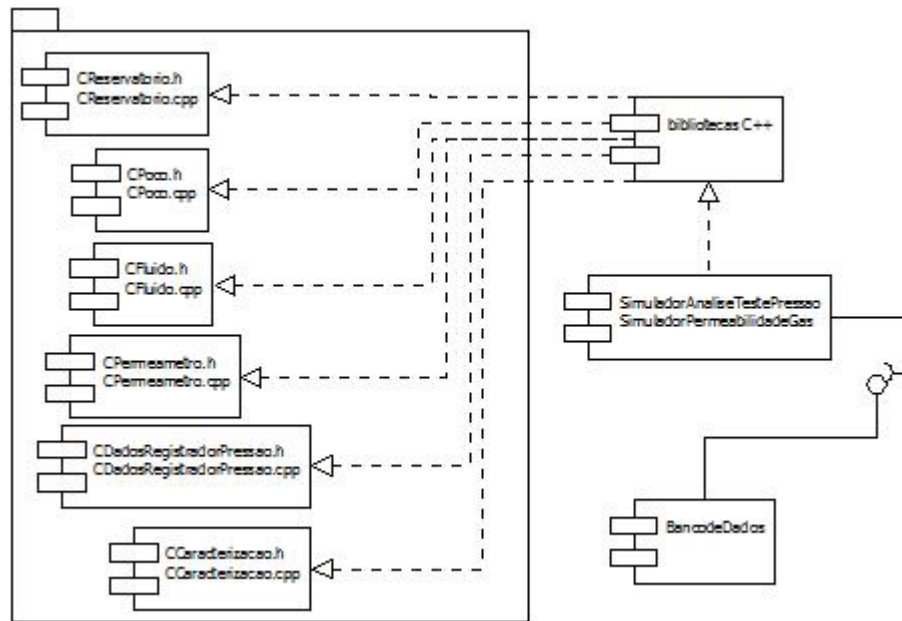


Figura 5.1: Diagrama de componentes

5.4 Diagrama de implantação

O diagrama de implantação é um diagrama que inclui relações entre o sistema e o hardware e deve incluir os elementos necessários para que o sistema seja colocado em funcionamento. Veja na Figura 5.2 diagrama de implantação do programa. Primeiramente, o registrador de pressão no poço registra a pressão e tempo de medição, enviando os dados para um computador na superfície. Os dados do testemunho obtidos a partir do permeametro à gás também são enviados. Esses arquivos são compilados em formato `.dat`. O programa importa os dados desse arquivo e na sua execução precisa de um monitor para mostrar os resultados e do teclado para receber parâmetros informados pelo usuário ou cliente.

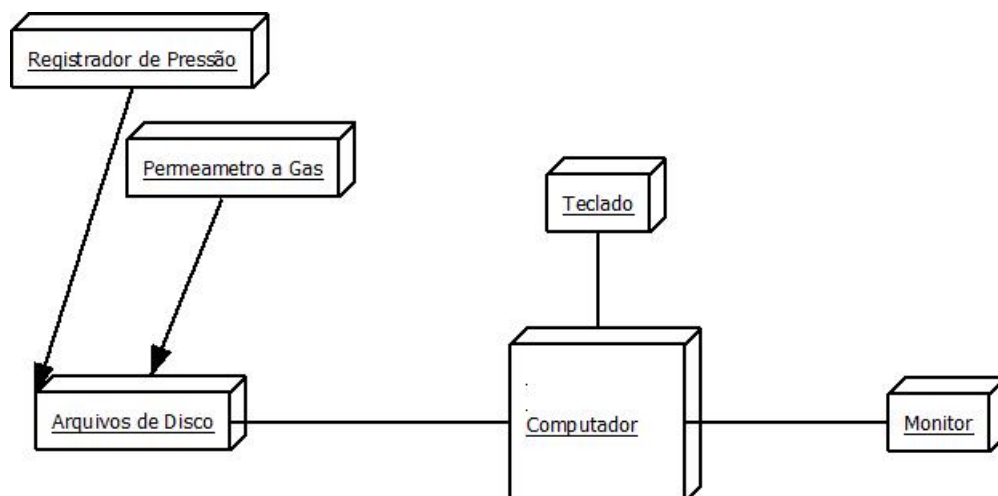


Figura 5.2: Diagrama de implantação

Capítulo 6

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 6.1 o arquivo com código da classe CPoco.

Listing 6.1: Arquivo de cabeçalho da classe CPoco.h.

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

class CPoco;

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

        ///raio do poco
        double raioPoco;
        ///vazao de producao
        double vazao;
        ///tempo de producao
        double tempoProducao;
        ///pressao no poco
        double pressaoPoco;

    public:
```

```

    ///Funcao que recebe dados do usuario, preenchendo atributos
    raiopoco, vazao, tempoproducao, pressaopoco
void EntradaDados();

    ///Funcao que verifica se houve erro na entrada de dados e pede
    nova entrada ate nao ocorrer erro
void Erro();

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

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

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

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

};
#endif

```

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

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

```

#include "CPoco.h"

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

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

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

```

```

    cout << "Informe o tempo de producao: " << endl;
    cin >> tempoProducao;
    cin.get();

    cout << "Informe a pressao no poco: " << endl;
    cin >> pressaoPoco;
    cin.get();

    cout << "Informe o raio do poco: " << endl;
    cin >> raioPoco;
    cin.get();
}

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

    while (tempoProducao<0.00)
    {
        cout << "Reinforme o tempo de producao: " << endl;
        cin >> tempoProducao;
        cin.get();
    }

    while (pressaoPoco<0.00)
    {
        cout << "Reinforme a pressão no poco: " << endl;
        cin >> pressaoPoco;
        cin.get();
    }

    while ((raioPoco<0.00) || (raioPoco>3.0))
    {
        cout << "Reinforme o raio do poco: " << endl;
        cin >> raioPoco;
        cin.get();
    }
}

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

```

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

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

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

//set
void CPoco::PressaoPoco(double _pressaoPoco)
{
    pressaoPoco = _pressaoPoco;
}

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

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

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

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

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

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



```

class CReservatorio;

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

        ///porosidade do reservatorio
        double porosidade;
        ///altura do reservatorio
        double altura;

    public:

        ///Funcao que recebe dados do usuario, preenchendo atributos
        porosidade e altura
        void EntradaDados();

        ///Funcao que verifica se houve erro na entrada de dados e pede
        nova entrada ate nao ocorrer erro
        void Erro();

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

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

};
#endif

```

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

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

```

#include "CReservatorio.h"

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

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

```

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

    cout << "Informe a porosidade da rocha reservatorio: " << endl;
    cin >> porosidade;
    cin.get();

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

}

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

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

}

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

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

//set
```

```

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

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

```

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

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

```

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

class CFluido;

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

    ///Fator Volume formacao do fluido
    double fatorVolumeFormacao;
    ///Viscosidade
    double viscosidade;
    ///Compressibilidade Total
    double compressibilidade;
    ///Temperatura do fluido em Fahrenheit
    double temperatura;
    ///Grau API do fluido
    double api;

public:

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

    ///Funcao que verifica se houve erro na entrada de dados e pede
    nova entrada ate nao ocorrer erro
    void Erro();

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

```

```

        ///Funcao get do fatorvolumeformacao

        double FatorVolumeFormacao() const;

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

        ///Funcao que seta a compressibilidade
        void Compressibilidade(double _compressibilidade);
        ///Funcao get da compressibilidade
        double Compressibilidade() const;
        ///Funcao que calcula e retorna o valor da viscosidade atraves
        da correlacao
        void Viscosidadecorrelacao();

};
#endif

```

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

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

```

#include "CFluido.h"
#include <cmath>

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

using namespace std;

//funcao de entrada de dados da classe
void CFluido::EntradaDados()
{
    cout << "Informe o fator volume-formacao do fluido: " << endl;
    cin >> fatorVolumeFormacao;
    cin.get();

    int resp;
    cout << "Usuario deseja entrar com o valor da viscosidade (1) ou
        calcular a partir de uma correlacao (2)? " << endl;
    cin >> resp;

    switch (resp)
    {
        case 1:

```

```

        cout<<"Entre com o valor da viscosidade: " <<endl;
        cin>> viscosidade;
        cin.get();

        break;

        case 2:

            cout << "Entre com a temperatura do fluido em Fahrenheit: " <<
                endl;
            cin >> temperatura;
            cin.get();

            cout<< "Entre com o grau API do fluido: " <<endl;
            cin>> api;
            cin.get();

            cout<< "A viscosidade sera calculada a partir da correlacao de
                Petrosky Farshad para Oleo Morto" << endl;
            Viscosidadecorrelacao();
            cout<< "A viscosidade calculada e: " << viscosidade << endl;
        }

        cout << "Informe a compressibilidade total (fluido+rocha): " << endl;
        cin >> compressibilidade;
        cin.get();

    }

    //funcao que acusa e conserta erro de entrada
    void CFluido::Erro()
    {

        //repete a entrada enquanto o valor for equivocado

        while (fatorVolumeFormacao<0.00)
        {
            cout << "Reinforme o Fator Volume-Formacao: " << endl;
            cin >> fatorVolumeFormacao;
            cin.get();
        }

        while (viscosidade<0.00)
        {
            cout << "Reinforme a viscosidade." << endl;
            cin >> viscosidade;

```

```

        cin.get();
    }

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

    while (api<0.00)
    {
        cout << "Reinforme o grau API" << endl;
        cin >> api;
        cin.get();
    }
}

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

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

//set
void CFluido::Viscosidadecorrelacao()
{
    double X;
    X = (4.59388*log10(temperatura))-22.82792;
    viscosidade = (2.3511 * pow(10,7)) * pow(temperatura, (-2.10255)) *
        pow(log10(api), X);
}

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

//get
double CFluido::Viscosidade() const
{

```

```

    return viscosidade;
}

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

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

```

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

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

```

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

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

class CDadosRegistradorPressao;

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

public:

    ///Funcao que importa os dados registrados do arquivo .dat,
    preenchendo os atributos da classe
    void Importa();

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

```

```

double PressaoMedida(int posicao) const;
///Funcao get da pressaomedida
std::vector<double> PressaoMedida() const;

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

};
#endif

```

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

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

```

#include "CDadosRegistradorPressao.h" //inclui o cabeçalho da classe

//inclui biblioteca para importacao de arquivos de disco
#include <fstream>

//inclui a biblioteca iostream pois usa funcoes cin, cout
#include <iostream>

//inclui a biblioteca vector pois usa vetores
#include <vector>

//inclui a biblioteca string pois usa variaveis string
#include <string>

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

void CDadosRegistradorPressao::Importa()
{
    //limpa os vetores de importacao para novo preenchimento
    tempoSemProducao.resize(0);
    pressaoMedida.resize(0);

    //indica o que o eixo x representa
    string eixoX;
    //indica o que o eixo y representa
    string eixoY;
    double x;
    double y;
    //nome do arquivo com os dados a serem importados
    string nomeArquivo;

```



```

cout << "Informe o nome do arquivo com os dados do registrador de
    pressao:" << endl;
//armazena a string digitada em nomearquivo
getline (cin,nomeArquivo);

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

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

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

}

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

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

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

//set

```

```

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

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

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

```

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

Listing 6.9: Arquivo de cabeçalho da classe CEstadistica.h.

```

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

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

class CEstadistica;

///Classe que calcula estatisticas do vetor, como media e desvio padrao,
    util para regressao
class CEstadistica
{
private:

    double media;
    double desvio;

public:

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

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

};

```

```
#endif
```

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

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

```
#include "CEstatistica.h"

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

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

using namespace std;

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

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

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

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

    return desvio = sqrt((vquadrado - ((1.0/v.size())*soma*soma))/(v.size()
        -1.0));
}
```

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

Listing 6.11: Arquivo de cabeçalho da classe `CAjusteCurvaMinimosQuadrados.h`.

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

```

#define CAjusteCurvaMinimosQuadrados_h

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

class CAjusteCurvaMinimosQuadrados;

//Classe que obtem os coeficiente da regressao linear por meio do
    metodo dos minimos quadrados
class CAjusteCurvaMinimosQuadrados
{

    //encapsulamento que permite o acesso para a classe e para a classe
        herdeira
protected:
    //coeficiente angular da reta do tipo  $y=ax+b$ 
    double coeficienteAngular;
    //coeficiente linear da reta do tipo  $y=ax+b$ 
    double coeficienteLinear;
    //coeficiente de correlacao da reta do tipo  $y=ax+b$ 
    double coeficienteCorrelacao;
    //cria um objeto da classe CEstatistica para ser utilizado
        funcoes de calculo
    CEstatistica estatistica;

public:

    //Funcao que retorna o valor do coeficiente de correlacao
    double CoeficienteCorrelacao () const;
    //Funcao que retorna o valor do coeficiente angular
    double CoeficienteAngular () const ;
    //Funcao que retorna o valor do coeficiente linear
    double CoeficienteLinear () const;
    //Funcao que faz os calculos do coefiente angular, linear e de
        correlacao
    void CalcularAjusteMinimosQuadrados (std::vector<double> vx, std
        ::vector<double> vy);

};

#endif

```

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

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

```
#include "CAjusteCurvaMinimosQuadrados.h"

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

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

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

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

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

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

    coeficienteAngular = mnum/mden;

    coeficienteLinear = mediay - (coeficienteAngular * mediax);

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

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

    coeficienteCorrelacao = -somar / sqrt(variay * variax);
}
```

```
double CAjusteCurvaMinimosQuadrados::CoeficienteLinear () const
{
    return coeficienteLinear;
}

double CAjusteCurvaMinimosQuadrados::CoeficienteCorrelacao() const
{
    return coeficienteCorrelacao;
}

double CAjusteCurvaMinimosQuadrados::CoeficienteAngular() const
{
    return coeficienteAngular;
}
```

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

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

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

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

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

///Declaracao da Classe filha de CAjusteCurvaMinimosQuadrados
class CAjusteCurva;

///Classe que executa a regressao linear e ajusta ate a correlacao
satisfatoria
class CAjusteCurva: public CAjusteCurvaMinimosQuadrados
{
public:
    ///coef. angular
    double m;
    ///coef. linear
    double n;
    ///coef. de correlacao
    double r;
    ///indica que o coef. de correlacao nao foi satisfatorio (ha
    estocagem)
    bool efeitoEstocagem;
```

```

        ///ajuste de variavel para logaritmica
        std::vector<double> logt;

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

};

//Fim da condicao de definicao da classe
#endif

```

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

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

```

#include "CAjusteCurva.h"

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

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

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

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

/// Funcao que cria a variavel logaritmica a partir dos parametros 1 e 3
da funcao, executa a regressao linear atraves do metodo dos minimos
quadrados.
void CAjusteCurva::Regressao(vector<double> vx, vector<double> vy,
    double z)
{
    ///limpa o vetor do eixo x para novo preenchimento
    logt.resize(0);

    ///loop que percorre todo o vetor vx e preenche logt
    for(int i=0 ; i<vx.size() ; i++)

```

```

        //transformacao da variavel em logaritmica
        logt.push_back (log10((z+vx[i]) / vx[i]));

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

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

/// Funcao que ajusta a regressao para o periodo correto, removendo os
    pontos referentes a estocagem
void CAjusteCurva::Ajuste(vector<double> vx, vector<double> vy, double z
    )
{
    /// variavel que contem os coef. de correlacao
    vector<double> coef(vx.size()/2,r);

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

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

    // loop principal que vai aumentando o valor de k e retirando as
        primeiras posicoes dos vetores (estocagem)
    for (int k=1 ; k<(vx.size()/2) ; k++)
    {
        // repete o loop ate achar o coef. de correlacao aceitavel
        if(coef[k-1]<0.9900)
        {
            // ocorre estocagem se cair na condicao
            efeitoEstocagem = true;
            cout << "Necessario novo Ajuste." << endl;

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

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

```



```

    }

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

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

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

    cout << "EQUACAO: y= " << m << " * x + " << n << endl <<
        "r= " << coef[k] << endl;

    if (1.2*coef[k]<coef[0])
        cout << "Regressao Linear nao tao perfeita,
                Indicativo de Reservatorio Heterogeneo" <<
                endl;

    //apos a segunda regressao
    if (k>1)
    {
        if ((1.2*coef[k])<coef[k-1])
        {
            k = vx.size()/2; //para terminar o loop
            cout << "Maximo Coeficiente de Correlacao
                    alcançado." << endl;
        }
    }

    }

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

```

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

Listing 6.15: Arquivo de cabeçalho da classe CCilindro.h.

```

#ifndef CCilindro_h
#define CCilindro_h

```

```
/// Declaracao da classe CCilindro
class CCilindro
{
private:
    /// declaracao do atributo diametro
    double diametro;
    /// declaracao do atributo comprimento
    double comprimento;
    /// declaracao do atributo volumeTotal
    double volumeTotal;
    /// declaracao do atributo areaTransversal
    double areaTransversal;

public:
    /// metodo set para o atributo diametro
    void Diametro ( double _diametro)
    { diametro = _diametro; };

    /// metodo get para o atributo diametro
    double Diametro()
    { return diametro; };

    /// metodo set para o atributo comprimento
    void Comprimento ( double _comprimento)
    { comprimento = _comprimento; };

    /// metodo get para o atributo comprimento
    double Comprimento()
    { return comprimento; };

    /// metodo set para o atributo volumeTotal
    void VolumeTotal ( double _volumeTotal)
    { volumeTotal = _volumeTotal; };

    /// metodo get para o atributo volumeTotal
    double VolumeTotal()
    { return volumeTotal; };

    /// metodo get para o atributo areaTransversal
    double AreaTransversal()
    { return areaTransversal; };

    /// metodo para calcular o volume total de um cilindro
    double VolumeTotal ( double diametro , double comprimento );

    /// metodo para calcular a área transversal de um cilindro
```

```

        double AreaTransversal ( double diametro );

};
#endif

```

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

Listing 6.16: Arquivo de implementação da classe CCilindro.cpp.

```

#include "CCilindro.h"
#include <cmath>

using namespace std;

///Calculando o volume total de um cilindro
double CCilindro :: VolumeTotal ( double diametro, double comprimento )
{
    volumeTotal = 3.14 * pow (( diametro / 2.0 ), 2.0) * comprimento;
    return volumeTotal;
}

/// Calculando a area transversal de um cilindro.
double CCilindro :: AreaTransversal ( double diametro )
{
    areaTransversal = 3.14 * pow (( diametro / 2.0 ), 2.0);
    return areaTransversal;
}

```

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

Listing 6.17: Arquivo de cabeçalho da classe CTestemunhoRocha.h.

```

#ifndef CTestemunhoRocha_h
#define CTestemunhoRocha_h
#include "CCilindro.h"

using namespace std;

/// declaracao da classe CTestemunhoRocha
class CTestemunhoRocha: public CCilindro
{
private:
    /// declaracao do atributo id
    int id;
    /// declaracao do atributo pesoSeco
    double pesoSeco;
    /// declaracao do atributo temperatura
    double temperatura;

public:

```

```

    /// metodo set para o atributo id
    void ID ( int _id )
    { id = _id; };

    /// metodo get para o atributo id
    int ID()
    { return id; };

    /// metodo set para o atributo pesoSeco
    void PesoSeco ( double _pesoSeco )
    { pesoSeco = _pesoSeco; };

    /// metodo get para o atributo pesoSeco
    double PesoSeco ()
    { return pesoSeco; };

    /// Construtor no qual ao ser acessada a classe gera um valor para
        temperatura igual a 25 graus Celsius
    CTestemunhoRocha() : temperatura (25) {};

    ///metodo para importar os dados referentes aos atributo da rocha e do
        cilindro
    void ImportaDadosRocha();

};
#endif

```

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

Listing 6.18: Arquivo de implementação da classe CTestemunhoRocha.cpp.

```

#include "CTestemunhoRocha.h"
#include "CCilindro.h"
#include <string>
#include <fstream>

using namespace std;

/// importando os dados do testemunho
void CTestemunhoRocha:: ImportaDadosRocha ()
{
    int x;
    double y;
    double z;
    double w;
    double k;
    string m;

```

```

ifstream fin;
fin.open ( "DadosFluidoRochaCilindro.dat" );

for (int i=0; i<5; i++) // Desconsidera o cabeçalho do arquivo
    DadosFluidoRochaCilindro.dat
    fin >> m;
// Neste ponto ira importar os dados na seguinte ordem id, pesoSeco,
// Comprimento, Diametro e para o id não pegar o valor da viscosidade
// é pedido para ler mais um valor.
while ( !fin.eof() )

{
    fin >> x;
    ID(x);
    fin >> y;
    PesoSeco(y);
    fin >> z;
    Comprimento(z);
    fin >> w;
    Diametro(w);
    fin >> k;
}
}

```

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

Listing 6.19: Arquivo de cabeçalho da classe CPermeametro.h.

```

#ifndef CPermeametro_h
#define CPermeametro_h
#include <vector>

///Declaracao da classe CPermeametro
class CPermeametro
{
private:
    /// declaracao do atributo pressaoSaida
    double pressaoSaida;

    /// declaracao do vetor da diferenca Pentrada^2 - Psaida^2
    std::vector <double> difPressao;

    /// declaracao do vetor pressaoEntrada
    std::vector <double> pressaoEntrada;

    /// declaracao do vetor vazao
    std::vector <double> vazao;

public:

```

```

    /// construtor da classe no qual e definido o valor 1 para a pressao
    saida
    CPermeametro () : pressaoSaida (1) {};

    /// método para importar os atributos para cálculo da permeabilidade
    void ImportaCalculaDadosPermeametro ();

    /// metodo set para o atributo pressaoSaida
    void PressaoSaida ( double _pressaoSaida)
    { pressaoSaida = _pressaoSaida; };

    /// metodo get para o atributo pressaoSaida
    double PressaoSaida()
    { return pressaoSaida; };

    /// metodo set para o atributo do valor da diferença dos quadrados das
    pressões
    void DifPressao ( std::vector <double> _difPressao)
    { difPressao = _difPressao; };

    /// metodo get para o atributo da diferença dos quadrados das pressões
    std::vector<double> DifPressao()
    { return difPressao; };

    /// metodo set para o atributo pressaoEntrada
    void PressaoEntrada ( std::vector <double> _pressaoEntrada)
    { pressaoEntrada = _pressaoEntrada; };

    /// metodo get para o atributo da pressaoEntrada
    std::vector<double> PressaoEntrada()
    { return pressaoEntrada; };

    /// metodo set para o atributo vazao
    void Vazao ( std::vector <double> _vazao)
    { vazao = _vazao; };

    /// metodo get para o atributo vazao
    std::vector<double> Vazao ()
    { return vazao; };

};
#endif

```

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

Listing 6.20: Arquivo de implementação da classe CPermeametro.cpp.

```

#include "CPermeametro.h"
#include <fstream>
#include <vector>

```

```

#include <cmath>
#include <string>

using namespace std;

/// importando os dados para cálculo da permeabilidade
void CPermeametro :: ImportaCalculaDadosPermeametro ()
{
    double x;
    double y;
    string m;
    ifstream fin;
    fin.open("DadosPermeametroGas.dat");

    //desconsidera o cabeçalho do arquivo DadosPermeametroGas.dat
    for (int i=0; i<2; i++)
        fin >> m;

    // faz com que a primeira coluna do arquivo seja os valores da vazao e
    a segunda da pressao de entrada
    while (!fin.eof())
    {
        fin>>x;
        vazao.push_back(x);
        fin>>y;
        pressaoEntrada.push_back(y);
    }

    /// gerando o vetor da diferenca dos quadrados das pressões
    for (int i=0 ; i<pressaoEntrada.size() ; i++)
        difPressao.push_back(pow(pressaoEntrada[i],2)-pow(pressaoSaida,2));
}

```

Apresenta-se na listagem 6.21 o arquivo com código da classe CGasPermeametro.

Listing 6.21: Arquivo de cabeçalho da classe CGasPermeametro.h.

```

#ifndef CGasPermeametro_h
#define CGasPermeametro_h

/// Declaracao da classe CGasPermeametro
class CGasPermeametro
{
private:
    /// declaracao do atributo viscosidade
    double viscosidade;

public:
    /// metodo para importar os dados referendo ao fluido
    void ImportaDadosFluido ();

```

```

    /// metodo set para o atributo viscosidade
    void Viscosidade ( double _viscosidade )
    { viscosidade = _viscosidade; };

    /// metodo get para o atributo diametro viscosidade
    double Viscosidade ()
    { return viscosidade; };

};
#endif

```

Apresenta-se na listagem 6.22 o arquivo de implementação da classe CGasPermeametro.

Listing 6.22: Arquivo de implementação da classe CGasPermeametro.cpp.

```

# include "CGasPermeametro.h"
# include <fstream>
# include <string>

using namespace std;

/// importando os dados do fluido
void CGasPermeametro :: ImportaDadosFluido ()
{
    double x;
    string m;

    ifstream fin;
    fin.open ( "DadosFluidoRochaCilindro.dat" );

    for (int i=0; i<5; i++)
        fin >> m;

    // Faz com que o valor referente a viscosidade seja sempre o ultimo do
    // arquivo de onde esta sendo extraido os dados.
    while ( !fin.eof() )
    {
        fin >> x;
        viscosidade=x;
    }
}

```

Apresenta-se na listagem 6.23 o arquivo com código da classe CCaracterizacaoReservatorio.

Listing 6.23: Arquivo de cabeçalho da classe CCaracterizacaoReservatorio.h.

```

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

```



```
#define CCaracterizacaoReservatorio_h

class CCaracterizacaoReservatorio;

///Classe que caracteriza o reservatorio.
class CCaracterizacaoReservatorio
{

public:
    ///Funcao que analisa os resultados e caracteriza o reservatorio.
    void Caracterizacao(double permeabilidade, double fatorpelicula,
        double indiceprodutividade, double raio pouco, double
        raioefetivo);

};
#endif
```

Apresenta-se na listagem 6.24 o arquivo de implementação da classe CCaracterizacaoReservatorio.

Listing 6.24: Arquivo de implementação da classe CCaracterizacaoReservatorio.cpp.

```
#include "CCaracterizacaoReservatorio.h"

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

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

///Funcao que caracteriza o reservatorio, dado os parametros calculados
void CCaracterizacaoReservatorio::Caracterizacao (double permeabilidade,
    double fatorpelicula, double indiceprodutividade, double raio pouco,
    double raioefetivo)
{
    ///condicoes que se satisfeitas, escrevem na tela caracteristicas do
reservatorio.

    if (permeabilidade<=10)
        cout << "1-Reservatorio com permeabilidade ruim." << endl;

    if ((permeabilidade<100)&&(permeabilidade>10))
        cout << "1-Reservatorio com permeabilidade boa." << endl;

    if (permeabilidade>=100)
        cout << "1-Reservatorio com permeabilidade excelente." << endl;
```

```

if (fatorpelicula==0)
    cout << "2-Reservatorio sem dano e sem estimulo." << endl;

if (fatorpelicula<0)
    cout << "2-Reservatorio estimulado" << endl;

if ((fatorpelicula>0)&&(fatorpelicula<5))
    cout << "2-Reservatorio com dano baixo, nao necessita de
        processos de acidificacao e/ou fraturamento hidraulico." << endl
        ;

if ((fatorpelicula>5)&&(fatorpelicula<10))
    cout << "2-Reservatorio com dano intermediario, pode ser usado
        processos de acidificacao e/ou fraturamento hidraulico." << endl
        ;

if (fatorpelicula>10)
    cout << "2-Reservatorio com dano alto, necessita de processos de
        acidificacao e/ou fraturamento hidraulico." << endl;

if (indiceprodutividade<=0.01)
    cout << "3-Reservatorio com produtividade baixo, considerar uso
        de tecnicas de recuperacao secundaria." << endl;

if ((indiceprodutividade>0.01)&&(indiceprodutividade<0.1))
    cout << "3-Reservatorio com produtividade regular, considerar uso
        de tecnicas de recuperacao secundaria." << endl;

if (indiceprodutividade>0.1)
    cout << "3-Reservatorio com produtividade boa." << endl;

if ((raioefetivo/raiopoco)<=0.0001)
    cout << "4-Razao de dano alto, pois o raio efetivo e muito menor
        que o raio do poco real, afetando a produtividade." << endl;
}

```

Apresenta-se na listagem 6.25 o arquivo com código da classe CSimuladorAnaliseTestePressao.

Listing 6.25: Arquivo de cabeçalho da classe CSimuladorAnaliseTestePressao.h.

```

#ifndef CSimuladorAnaliseTestePressao_h
#define CSimuladorAnaliseTestePressao_h

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

#include "cgplot.h"
#include "CReservatorio.h"

```

```

#include "CFluido.h"
#include "CPoco.h"
#include "CDadosRegistradorPressao.h"
#include "CEstatistica.h"
#include "CAjusteCurva.h"
#include "CCaracterizacaoReservatorio.h" ///criacao do objeto
    caracterizar da classe CCaracterizacao.h
#include "CSimuladorPermeabilidadeGas.h"

class CSimuladorAnaliseTestePressao;

///Classe que faz a analise do teste de pressao realizado no campo e
infere as propriedades do reservatorio
class CSimuladorAnaliseTestePressao
{

public:
    ///permeabilidade do reservatorio
    double permeabilidade;
    ///pressao inicial que se encontrava o reservatorio
    double pressaoInicial;
    ///skin factor do poco
    double fatorPelicula;
    ///queda de pressao referente ao fator de pelicula
    double pressaoDano;
    ///indice de produtividade do reservatorio
    double indiceProdutividade;
    ///eficiencia de fluxo do reservatorio
    double eficienciaFluxo;
    ///raio efetivo do poco
    double raioEfetivo;
    ///coeficiente de estocagem do poco
    double coeficienteEstocagem;
    ///tempo de duracao do efeito de estocagem
    double tempoEstocagem;

    ///criacao do objeto poco da classe CPoco
    CPoco poco;
    ///criacao do objeto fluido da classe CFluido
    CFluido fluido;
    ///criacao do objeto reservatorio da classe CReservatorio
    CReservatorio reservatorio;
    ///criacao do objeto registrador da classe CRegistrador
    CDadosRegistradorPressao registrador;
    ///criacao do objeto ajuste da classe CAjuste
    CAjusteCurva ajuste;

```

```

        ///criacao do objeto caracterizar da classe CCaracterizacao
        CCaracterizacaoReservatorio caracterizar;
        ///cria objeto de armazenamento de dados
        std::ofstream fout;
        ///sistema de unidades utilizado do reservatorio
        double sistemaUnidade;

public:

        ///Funcao que calcula e preenche o atributo permeabilidade
        void CalculoPermeabilidade ();
        ///Funcao que calcula e preenche o atributo pressao inicial
        void CalculoPressaoInicial ();
        ///Funcao que calcula e preenche o atributo fatorpelicula
        void CalculoFatorPelicula ();
        ///Funcao que calcula e preenche o atributo raioefetivo
        void CalculoRaioEfetivo ();
        ///Funcao que calcula e preenche os atributos
            indiceprodutividade, eficienciafluxo, pressaodano
        void CalculoIndices ();
        ///Funcao que calcula e preenche os atributos
            coeficienteestocagem e tempoestocagem
        void CalculoEstocagem ();
        ///Funcao que chama as entradas de dados necessarias
        void EntradaDados();
        ///Funcao principal que executa a simulacao do teste
        void ImportaDados();
        ///Funcao que executa o simulador
        void Executar();
        ///Funcao que exporta os dados para um arquivo.dat
        void Exporta ();
        ///Funcao que exibe um menu e recebe dado do usuario, esse dado
            preenche o atributo sistemaunidade
        void EntradaSistemaUnidades();
        ///Funcao que seta o sistemaunidade
        void SistemaUnidade(double _sistemaUnidade);
        ///Funcao get do sistemaunidade
        double SistemaUnidade() const;

};

#endif

```

Apresenta-se na listagem 6.26 o arquivo de implementação da classe CSimuladorAnaliseTestePressao.

Listing 6.26: Arquivo de implementação da classe CSimuladorAnaliseTestePressao.cpp.

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

```

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

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

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

#include <fstream>

#include <string>

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

///Funcao principal que executa a simulacao do teste
void CSimuladorAnaliseTestePressao::Executar()
{
    cout << endl << "
    =====
    " << endl
        <<      endl << "SIMULADOR PARA CARACTERIZACAO DE
        RESERVATORIOS INTEGRADO A ANALISE PETROFISICA" <<
        endl
        << endl << "
        =====
        " << endl
        << endl << "1-Rodar o Programa" << endl << "2-Sair" << endl <<
        endl;
    int i = 0;
    cin >> i;

    while (i==1) //quando terminar a execucao do programa, se o usuario
        quiser, o programa roda novamente
    {
        cout << "Entrada de Dados do teste de pressao realizado" <<
            endl
            << "-----"
            << endl;

        int n;
        EntradaSistemaUnidades();

        cout << "Digite (1) Para entrar com os parametros do teste
            manualmente ou (2) para puxar os dados a partir de um

```

```

        arquivo_de_disco: << endl;
cin>>n; cin.get();

switch(n)
{
    case 1:

        EntradaDados();

        break;

    case 2:

        ImportaDados();

        break;
}

cout << "Entrada_de_Dados_Finalizada" << endl
    << "-----"
    << endl << endl;
cout << "Importacao_dos_dados_do_registrador_de_Pressao" <<
endl
    << "-----"
    << endl << endl;

registrador.Importa();
cout << "Dados_do_registrador_importados_com_sucesso" <<
endl << endl;

cout << "Regressao_Linear_dos_Dados" << endl
    << "-----"
    << endl << endl;

ajuste.Regressao (registrador.TempoSemProducao(),
    registrador.PressaoMedida(), poco.TempoProducao());

cin.get ();
cout << "Localizando_o_Periodo_Transiente" << endl
    << "-----"
    << endl << endl;

ajuste.Ajuste(registrador.TempoSemProducao(), registrador.
    PressaoMedida(), poco.TempoProducao());

cout << "Regressao_linear_feita_com_sucesso" << endl
    << "-----"
    << endl;

```

```

//GERA O GRAFICO CASO USUARIO QUEIRA

cout << "Deseja gerar o grafico: " << endl << "1-Sim" <<
    endl << "2-Nao" << endl << endl << endl;
int j;
cin >> j;

if (j==1)
{
    CGnuplot plot;
    //gera o grafico com a reta perfeita obtida
    plot.set_ylabel ("Pressao");
    plot.set_xlabel ("Log T");

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

cout << "Parametros do Reservatorio" << endl
    << "-----"
    << endl << endl;
//CALCULOS
CalculoPermeabilidade();
CalculoPressaoInicial();
CalculoFatorPelicula();
CalculoRaioEfetivo();
CalculoIndices();

cin.get();

char resp;
cout<<"\nVoce deseja comparar a permeabilidade a partir dos
    dados de pouco com a permeabilidade a partir de dados de
    laboratorio?(s/n)" <<endl;
cin>> resp;

if( resp== 's')
{
    CSimuladorPermeabilidadeGas simulador2;

    //executa a funcao Permeabilidade para calculo

```

```

        da mesma a partir de dados de laboratorio

        simulador2.Permeabilidade();
        simulador2.ComparaPermeabilidades(permeabilidade
        );
    }
    //se nao ocorreu estocagem
    if (ajuste.efeitoEstocagem==false)
        cout << "Reservatorio sem o periodo de estocagem" <<
            endl;
    else
    {
        CalculoEstocagem ();
        //zera o valor em caso de novo calculo
        ajuste.efeitoEstocagem = false;
    }

    cout << "Caracterizacao do Reservatorio." << endl
        << "-----"
        << endl;

    caracterizar.Caracterizacao(permeabilidade, fatorPelícula,
        indiceProdutividade, poco.RaioPoco(), raioEfetivo);
    cin.get();

    char z;

    cout<<"Deseja salvar os resultados do teste num arquivo de
        disco(s/n):"<< endl;
    cin>>z; cin.get();

    if(z=='s')
        Exporta();

    //Nova Escolha

        cout << "\n1-Rodar o Programa Novamente" << endl
            << "2-Sair" << endl << endl;

    cin >> i;
    }
}

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

```



```

    fluido.Erro();
    poco.EntradaDados();
    poco.Erro();
}

//Calcula a permeabilidade
void CSimuladorAnaliseTestePressao::CalculoPermeabilidade ()
{
    permeabilidade = (1.151 * sistemaUnidade * poco.Vazao() * fluido.
        FatorVolumeFormacao() *
        fluido.Viscosidade()) / (-ajuste.m * reservatorio.
        Altura());

    cout << "Permeabilidade:_" << permeabilidade;

    cout << "_milidarcy" << endl;
}

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

    if (sistemaUnidade==141.2)
        cout << "_psi" << endl;

    if (sistemaUnidade==19.03)
        cout << "_kgf/cm2" << endl;
}

//Calcula fator pelicula
void CSimuladorAnaliseTestePressao::CalculoFatorPelicula ()
{
    fatorPelicula = 1.151 * (((ajuste.m * log10(poco.TempoProducao()
        )) + ajuste.n - poco.PressaoPoco())/
        -ajuste.m) - log10((sistemaUnidade *
        permeabilidade) /
        (reservatorio.Porosidade() * fluido.Viscosidade()
        * fluido.Compressibilidade()
        * poco.RaioPoco () * poco.RaioPoco())) - 0.3514 +
        log10 (poco.TempoProducao()+1));
}

```

```

        cout << "Fator de Pelicula: " << fatorPelicula << endl;
    }

    //Calcula raio efetivo
    void CSimuladorAnaliseTestePressao::CalculoRaioEfetivo()
    {
        raioEfetivo = poco.RaioPoco() * exp(-fatorPelicula);
        cout << "Raio Efetivo: " << raioEfetivo;

        if (sistemaUnidade==19.03)
            cout << " metros" << endl;

        if (sistemaUnidade==141.2)
            cout << " ft" << endl;
    }

    //Calcula do indice de produtividade, eficiencia de fluxo e queda de
    //pressao referente ao dano
    void CSimuladorAnaliseTestePressao::CalculoIndices ()
    {
        pressaoDano = 0.869 * (-ajuste.m) * fatorPelicula;

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

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

        cout << "Queda de Pressao devido ao dano: " << pressaoDano <<
            endl;

        //Exibir em porcentagens
        cout << "Indice de Produtividade: " << indiceProdutividade*100.0
            << " %" << endl <<
            "Eficiencia de Fluxo: " << eficienciaFluxo*100.0 << " %" <<
            endl;
    }

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

        tempoEstocagem = ((60.0 + 3.5 * fatorPelicula)/(permeabilidade *
            reservatorio.Altura()))

```

```

        * sistemaUnidade * 24.0 * coeficienteEstocagem *
        fluido.Viscosidade();

    cout << "Coeficiente de Estocagem: " << coeficienteEstocagem <<
        endl
        << "Tempo de Estocagem: " << tempoEstocagem;

    cout << " horas" << endl;

}

void CSimuladorAnaliseTestePressao::Exporta ()
{
    //armazena a string digitada em nomeSaida
    string nome;
    cout << "\nInforme o nome do arquivo de saida com os
        parametros calculados pelo simulador: " << endl;
    cin>>nome;
    cin.get();

    //getline (cin,nome);
    string formato = ".dat";
    string Saida = nome+formato;

    ///abre arquivo
    fout.open (Saida.c_str());

    fout<< "Permeabilidade: " << permeabilidade;
    fout<< " milidarcy" << endl;
    fout<< "Pressao Inicial: " << pressaoInicial;

    if (sistemaUnidade==141.2)
        fout << " psi" << endl;

    if (sistemaUnidade==19.03)
        fout << " kgf/cm2" << endl;

    fout << "Fator de Pelicula: " << fatorPelicula << endl;
    fout << "Raio Efetivo: " << raioEfetivo;

    if (sistemaUnidade==19.03)
        fout << " metros" << endl;

    if (sistemaUnidade==141.2)
        fout << " ft" << endl;
    fout << "Queda de Pressao devido ao dano: " << pressaoDano <<
        endl;
}

```

```

        //Exibir em porcentagens
        fout << "Indice_de_Produtividade:_" << indiceProdutividade*100.0
            << " _%" << endl
            << "Eficiencia_de_Fluxo:_" << eficienciaFluxo*100.0 << "
                _%" << endl;

        fout.close ();
    }

    void CSimuladorAnaliseTestePressao:: ImportaDados () ///importando os
        dados do testemunho
    {
        double a;
        double b;
        double c;
        double d;
        double e;
        double f;
        double g;
        double h;
        double j;

        string m;
        string nomeArquivo;

        cout << "Informe_o_nome_do_arquivo_com_os_dados:_" << endl;
        //armazena a string digitada em nomearquivo
        getline (cin,nomeArquivo);

        ifstream fin;
        fin.open ( nomeArquivo.c_str() );

        for (int i=0; i<9; i++)
            fin >> m;

        while ( !fin.eof() ) ///neste ponto ira importar os dados na
            seguinte ordem id, pesoSeco, Comprimento, Diametro e para o id não
            pegar o valor da
            ///viscosidade é pedido para ler mais um valor.
        {
            fin >> a;
            reservatorio.porosidade=a;
            fin>>b;
            reservatorio.altura=b;
            fin>>c;
            fluido.fatorVolumeFormacao=c;
            fin>>d;
            fluido.viscosidade=d;
            fin>>e;
            fluido.compressibilidade=e;
        }
    }

```

```

        fin>>f;
        poco.vazao=f;
        fin>>g;
        poco.tempoProducao=g;
    fin >> h;
    poco.pressaoPoco=h;
    fin >> j;
    poco.raioPoco=j;
}
}

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

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

//Funcao que preenche o sistemaunidade por um pequeno menu
void CSimuladorAnaliseTestePressao::EntradaSistemaUnidades()
{
    cout << "Qual o sistema de unidades utilizado para informar os
        parametros:" << endl <<
        "1- Americano (Oilfield)" << endl << "2- Brasileiro (Petrobras)
        " << endl;

    int i;
    cin >> i;
    cin.get();

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

    if(i==1)
        sistemaUnidade = 141.2;

    if(i==2)

```

```

        sistemaUnidade = 19.03;
    }

```

Apresenta-se na listagem 6.27 o arquivo com código da classe

CSimuladorPermeabilidadeGas.

Listing 6.27: Arquivo de cabeçalho da classe CSimuladorPermeabilidadeGas.h.

```

#ifndef CSimuladorPermeabilidadeGas_h
#define CSimuladorPermeabilidadeGas_h
#include "CGasPermeametro.h"
#include "CTestemunhoRocha.h"
#include "CAjusteCurvaMinimosQuadrados.h"
#include "CPermeametro.h"
#include "CReservatorio.h"

class CSimuladorPermeabilidadeGas /// declara a classe
    CSimuladorPermeabilidadeGas
{
private:
    /// declaracao do objeto testemunho referente a classe
    CTestemunhoRochaCilindrica
    CTestemunhoRocha dados;
    /// declaracao do objeto fluido referente a classe CGasPermeametro
    CGasPermeametro fluido;
    /// declaracao do objeto permeametroGas referente a classe
    CPermeametroGas
    CPermeametro permeametro;
    /// declaracao do objeto ajuste referente a classe CAjusteCurva
    CAjusteCurvaMinimosQuadrados ajuste;
    /// declaracao do atributo permeabilidade do testemunho
    double permeabilidade;

public:
    /// metodo para calculo da permeabilidade
    double Permeabilidade ();
    /// metodo para comparacao das permeabilidades
    void ComparaPermeabilidades(double k_reservatorio);
};

#endif

```

Apresenta-se na listagem 6.28 o arquivo de implementação da classe CSimuladorPermeabilidadeGas.

Listing 6.28: Arquivo de implementação da classe CSimuladorPermeabilidadeGas.cpp.

```

#include "CSimuladorPermeabilidadeGas.h"
#include <iostream>
#include <cmath>

using namespace std;

// metodo para calcular a permeabilidade
double CSimuladorPermeabilidadeGas :: Permeabilidade ()
{
    // importa os dados do fluido
    fluido.ImportaDadosFluido();

    // importa os dados referentes as caracteristicas da rocha e do
    cilindro
    dados.ImportaDadosRocha();

    // importa os dados para calcular a permeabilidade
    permeametro.ImportaCalculaDadosPermeametro();

    // calculo da permeabilidade utilizando o coeficiente angular do
    ajuste
    ajuste.CalcularAjusteMinimosQuadrados(permeametro.DifPressao(),
        permeametro.Vazao());
        permeabilidade = ( 2000 * dados.Comprimento() * fluido.
            Viscosidade() * permeametro.PressaoSaida() * ajuste.
            CoeficienteAngular() )/ dados.AreaTransversal(dados.Diametro
            ());
            cout << "Permeabilidade_a_partir_de_teste_de_laboratorio
                _=_ " << permeabilidade;
            cout << "_milidarcy" << endl;

    cout << "Viscosidade_do_fluido=_ " << fluido.Viscosidade() << "_cp" <<
        endl << endl;

    // retornar a permeabilidade
    return permeabilidade;
}

void CSimuladorPermeabilidadeGas::ComparaPermeabilidades(double
    k_reservatorio)
{
    double erro= (k_reservatorio-permeabilidade)*(100/k_reservatorio
        );

    if(erro<0)
    {

```

```

double erro= (permeabilidade-k_reservatorio)*(100/
    permeabilidade);
    if (erro>50)
    {
        cout<< "O erro da permeabilidade foi: " << erro
            << " porcentos." << endl;
        cout<< "\nEsse erro pode ser explicado devido a
            heterogeneidades no reservatorio visto que
            num teste de laboratorio avalia-se um pequeno
            intervalo da formacao.\n" << endl;
    }
    else
    {
        cout<< "O erro da permeabilidade foi: " << erro
            << " porcentos." << endl;
        cout<< "\nA analise da amostra de rocha foi
            condizente com o encontrado no teste de poco
            .\n" <<endl;
    }
}
else if (erro>50)
{
    cout<< "O erro da permeabilidade foi: " << erro << "
        porcentos." << endl;
    cout<< "\nEsse erro pode ser explicado devido a
        heterogeneidades no reservatorio visto que num teste
        de laboratorio avalia-se um pequeno intervalo da
        formacao.\n" << endl;
}
else
{
    cout<< "O erro da permeabilidade foi: " << erro << " porcentos
        .\n" << endl;
    cout<< "\nA analise da amostra de rocha foi condizente com o
        encontrado no teste de poco.\n" <<endl;
}
}
}

```

Apresenta-se na listagem 6.29 o programa que usa as classes listadas acima.

Listing 6.29: Arquivo de implementação da função main().

```

#include "CSimuladorAnaliseTestePressao.h"

using namespace std;

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

```



```
        CSimuladorAnaliseTestePressao simulador;  
  
        //executa a funcao de analise do teste de pressao  
        simulador.Executar();  
  
        //retorna 0 se o programa rodou normalmente  
  
        cin.get();  
  
        return 0;  
    }
```

Capítulo 7

Teste

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

7.1 Dados entrada Teste na plataforma GNU/Linux

O teste realizado no sistema operacional GNU/Linux, onde o código do software foi desenvolvido utilizando o compilador 'g++'. Verificou-se se o programa retornava valores corretos, se os erros de entradas de dados eram observadas, se a comparação entre as permeabilidades obtidas pelo teste de poço e amostragem retornava corretamente.

A Figura 7.1 mostra a inicialização do programa, onde é solicitada a seleção do sistema de unidades, o tipo de entrada dos parâmetros, dando ao usuário a opção de importar os dados de um arquivo de disco ou inserir manualmente e a importação dos dados do registrador de pressão para execução do teste.

```
=====
SIMULADOR PARA CARACTERIZACAO DE RESERVATORIOS INTEGRADO A ANALISE PETROFISICA
=====

1-Rodar o Programa
2-Sair

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)
2
Digite (1) Para entrar com os parametros do teste manualmente ou (2) para puxar os dados a partir de um arquivo de disco:
2
Informe o nome do arquivo com os dados :
ExemploEntradaDados-Caso2.dat
Entrada de Dados Finalizada
-----

Importacao dos dados do registrador de Pressao
-----

Informe o nome do arquivo com os dados do registrador de pressao:
DadosRegistrador-Caso2.dat
Dados do registrador importados com sucesso

Regressao Linear dos Dados
-----

EQUACAO: y = -738.931 * x + 5032.77
r = 0.966934
■
```

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

Em seguida, como mostra a Figura 7.2 realiza a regressão linear, fazendo ajustes necessários devido ao período de estocagem observado e exibindo o resultado na tela.

```
Informe o nome do arquivo com os dados do registrador de pressao:
DadosRegistrador-Caso2.dat
Dados do registrador importados com sucesso

Regressao Linear dos Dados
-----
EQUACAO: y = -738.931 * x + 5032.77
r = 0.966934

Localizando o Período Transiente
-----
Necessario novo Ajuste.
EQUACAO: y = -650.956 * x + 4936.79
r = 0.982274
Necessario novo Ajuste.
EQUACAO: y = -603.603 * x + 4887.15
r = 0.987105
Necessario novo Ajuste.
EQUACAO: y = -567.885 * x + 4850.88
r = 0.989599
Necessario novo Ajuste.
EQUACAO: y = -531.661 * x + 4814.99
r = 0.992985
Necessario novo Ajuste.
EQUACAO: y = -479.137 * x + 4764.6
r = 0.992523
Regressao linear feita com sucesso
-----
Deseja gerar o grafico:
1-Sim
2-Nao
```

Figura 7.2: Tela do programa mostrando a regressão linear realizada com ajustes devido ao período de estocagem

Após a realização da regressão linear e os ajustes devidos, o usuário pode solicitar a geração do gráfico, como mostrado na Figura 7.2. Caso o usuário opte por gerar o gráfico, o resultado será exibido como mostrado na Figura 7.3. Após a geração do gráfico, os parâmetros calculados à partir do teste de formação serão exibidos na tela, como mostrado na Figura 7.9. O usuário, então, poderá escolher se deseja comparar os dados do teste de poço com os dados obtidos em amostragem de testemunho. Caso opte por comparar, o usuário obterá a comparação e em seguida a caracterização do reservatório, conforme mostrado na Figura 7.4. Além disso, possibilita a exportação dos resultados para um arquivo de disco, além de opção por rodar novamente o programa.

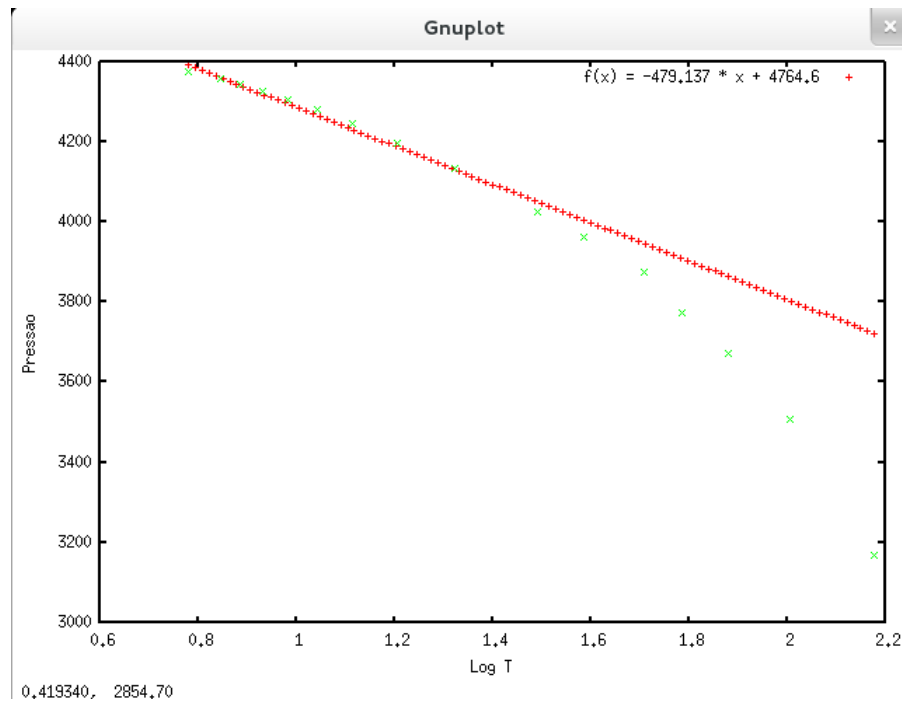


Figura 7.3: Gráfico de Pws vs. $\log[(Tp + \Delta t)/\Delta t]$ gerado

```

Permeabilidade: 3.9959 mDarcy
Pressao Inicial: 4764.6 kgf/cm2
Fator de Pelicula: 0.178299
Ralo Efetivo: 0.597592 metros
Queda de Pressao devido ao dano: -74.2385
Indice de Produtividade: 24.012 %
Eficiencia de Fluxo: 101.804 %

Voce deseja comparar a permeabilidade a partir dos dados de poço com a permeabilidade a partir de dados de laboratorio? (s/n)
s
Permeabilidade a partir de teste de laboratorio = 13.2469 mDarcy
Viscosidade do fluido = 0.0174 cp

O erro da permeabilidade foi: 69.8351 porcentos.

Esse erro pode ser explicado devido a heterogeneidades no reservatorio visto que num teste de laboratorio avalia-se um pequeno intervalo da formacao.

Coeficiente de Estocagem: 0.0362512
Tempo de Estocagem: 19.3301 horas
Caracterizacao do Reservatorio.
-----
1 - Reservatorio com permeabilidade ruim.
2 - Reservatorio estimulado
3 - Reservatorio com produtividade boa.
Deseja salvar os resultados do teste num arquivo de disco (s/n) :
s
Informe o nome do arquivo de saida com os parametros calculados pelo simulador:
Saida
1-Rodar o Programa Novamente
2-Sair

```

Figura 7.4: Tela exibindo cálculo dos parâmetros, comparação entre permeabilidades, caracterização do reservatório, opção por saída de dados e opção por rodar novamente o programa

7.2 Dados entrada Teste na plataforma Windows

O teste realizado no sistema operacional Windows 7, onde o código do software foi desenvolvido utilizando o compilador 'Dev C++'. Verificou-se se o programa retornava valores corretos, se os erros de entradas de dados eram observadas, se a comparação entre as permeabilidades obtidas pelo teste de poço e amostragem retornava corretamente.

A Figura 7.5 mostra a inicialização do programa, onde é solicitada a seleção do sistema de unidades e a entrada dos parâmetros.

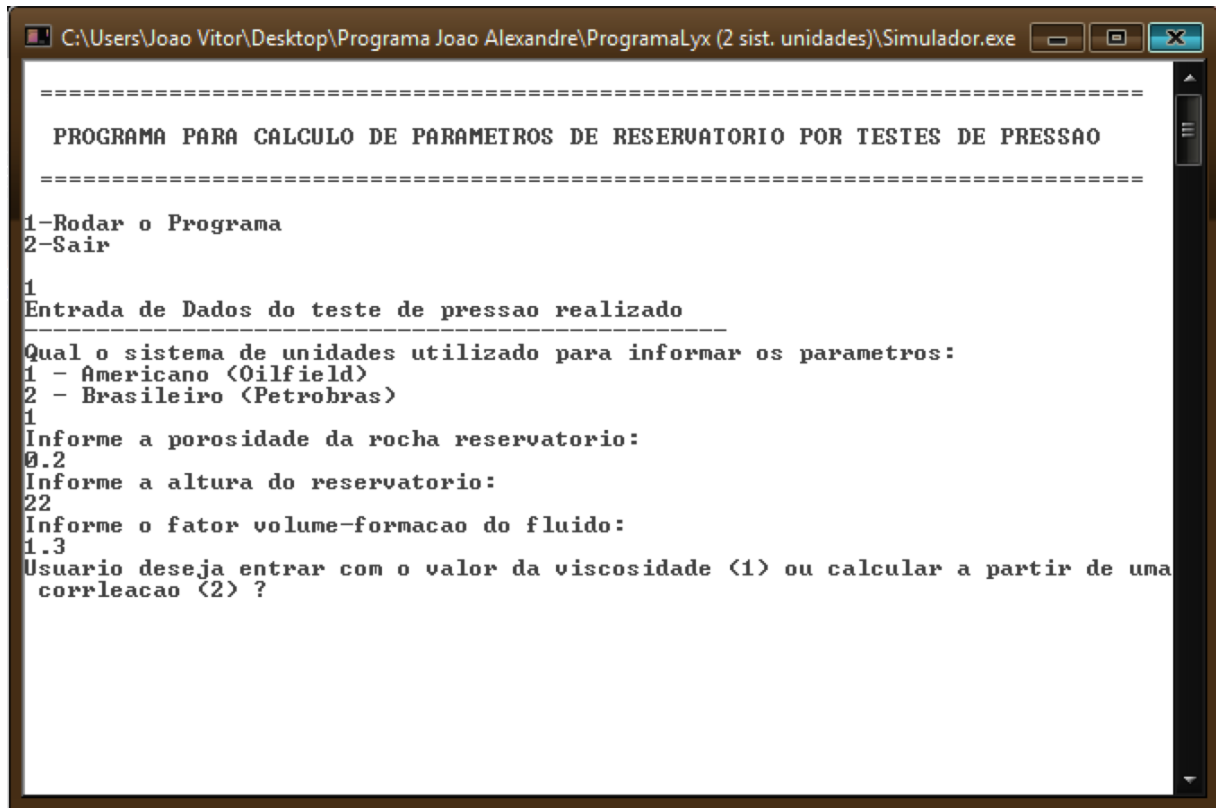
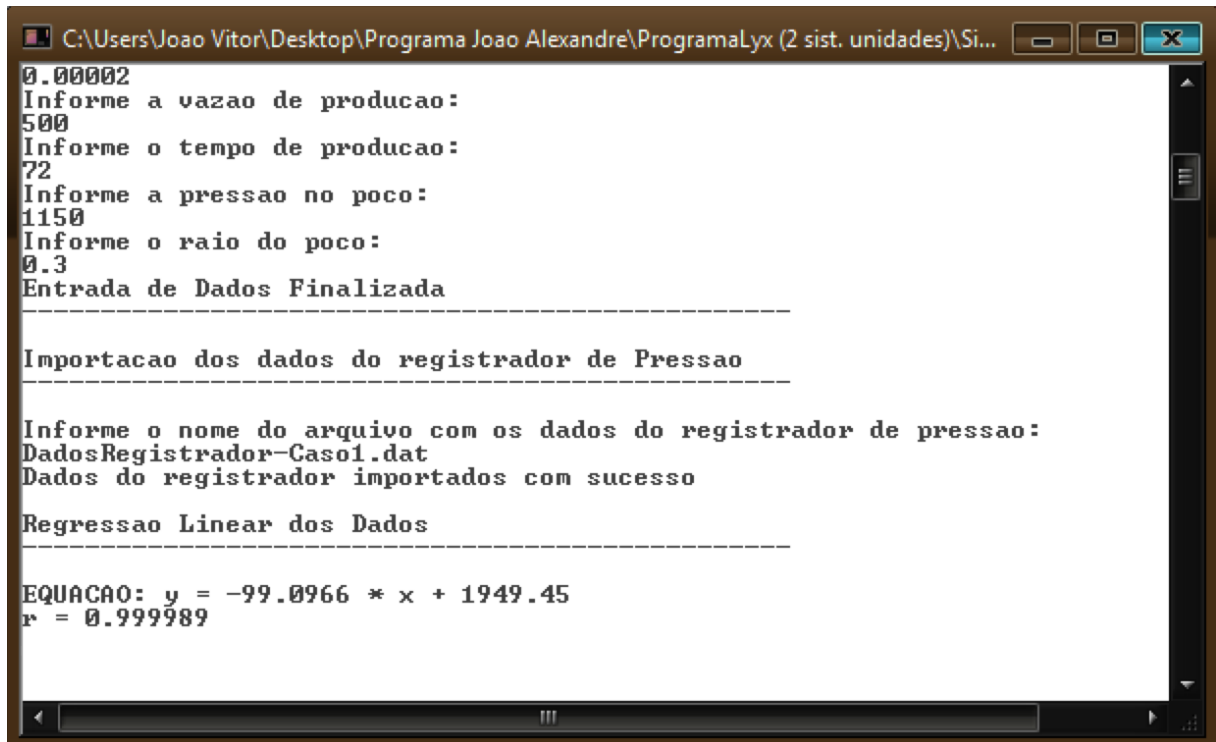


Figura 7.5: Tela do programa mostrando a entrada de dados

Em seguida, como mostra a Figura 7.6, o programa solicita os dados do registrador de pressão, e realiza a regressão linear, exibindo o resultado na tela.



```
C:\Users\Joao Vitor\Desktop\Programa Joao Alexandre\ProgramaLyx (2 sist. unidades)\Si...
0.00002
Informe a vazao de producao:
500
Informe o tempo de producao:
72
Informe a pressao no poço:
1150
Informe o raio do poço:
0.3
Entrada de Dados Finalizada
-----
Importacao dos dados do registrador de Pressao
-----
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
```

Figura 7.6: 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 na Figura 7.7. Caso o usuário opte por gerar o gráfico, o resultado será exibido como mostrado na Figura 7.8. Após a geração do gráfico, os parâmetros calculados à partir do teste de formação serão exibidos na tela, como mostrado na Figura 7.9. O usuário, então, poderá escolher se deseja comparar os dados do teste de poço com os dados obtidos em amostragem de testemunho. Caso opte por comparar, o usuário obterá a comparação e em seguida a caracterização do reservatório, conforme mostrado na Figura 7.10. Caso não opte por comparar, o usuário obterá a caracterização do reservatório, como mostrado na Figura 7.11. Em ambos os casos, o usuário pode optar entre executar novamente o programa ou encerrá-lo.

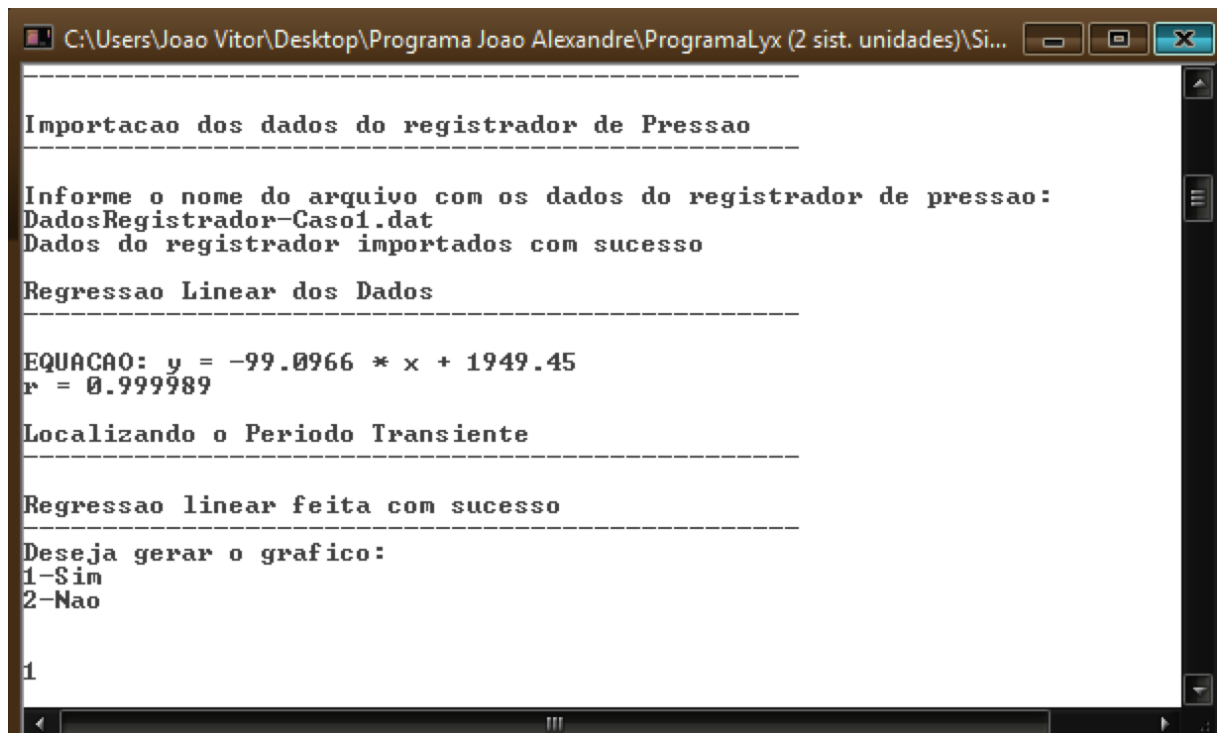


Figura 7.7: Tela do programa mostrando a possibilidade de geração do gráfico.

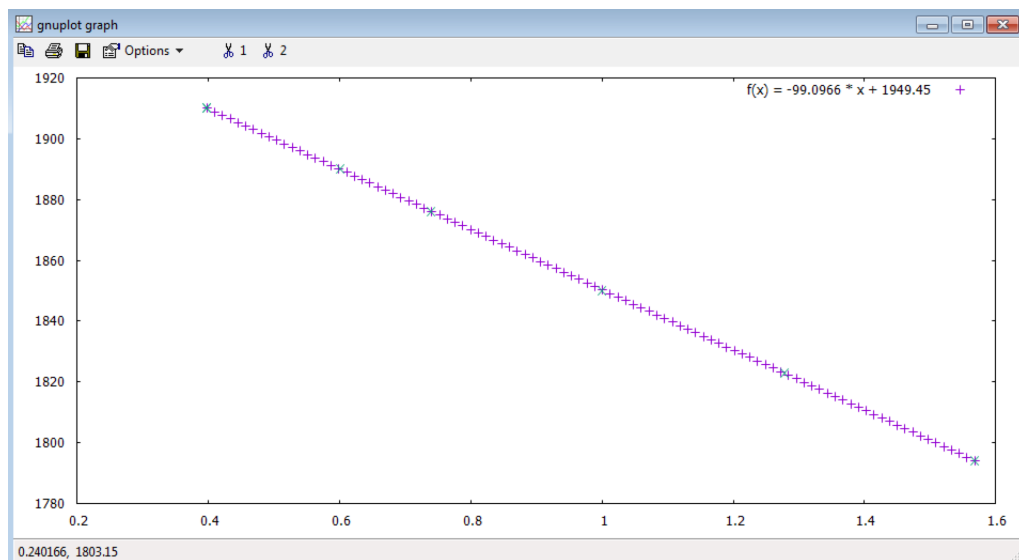


Figura 7.8: Gráfico de Pws vs. $\log[(Tp + \Delta t)/\Delta t]$ gerado pelo Gnuplot.

```

C:\Users\Joao Vitor\Desktop\Programa Joao Alexandre\ProgramaLyx (2 sist. unidades)\Si...
EQUACAO: y = -99.0966 * x + 1949.45
r = 0.999989

Localizando o Período Transiente

-----

Regressão linear feita com sucesso

-----

Deseja gerar o gráfico:
1-Sim
2-Não

1

Parâmetros do Reservatório

-----

Permeabilidade: 48.4554 mDarcy
Pressão Inicial: 1949.45 psi
Fator de Película: -2.94305
Raio Efetivo: 5.69207 ft
Queda de Pressão devido ao dano: -253.44
Índice de Produtividade: 62.5433 %
Eficiência de Fluxo: 131.702 %

```

Figura 7.9: Parâmetros do reservatório calculado pelo teste de formação realizado

```

C:\Users\Joao Vitor\Desktop\Programa Joao Alexandre\ProgramaLyx (2 sist. unidades)\Simulador.exe
Pressão Inicial: 1949.45 psi
Fator de Película: -2.94305
Raio Efetivo: 5.69207 ft
Queda de Pressão devido ao dano: -253.44
Índice de Produtividade: 62.5433 %
Eficiência de Fluxo: 131.702 %

Você deseja comparar a permeabilidade a partir dos dados de poço com a permeabi
lidade a partir de dados de laboratório? (s/n)
s
Permeabilidade a partir de teste de laboratório = 13.2469 mDarcy
Viscosidade do fluido = 0.0174 cp

O erro da permeabilidade foi: 72.6617 porcentos.

A análise da amostra de rocha foi condizente com o encontrado no teste de poço.

Reservatório sem o período de estocagem
Caracterização do Reservatório.

-----

1 - Reservatório com permeabilidade boa.
2 - Reservatório estimulado
3 - Reservatório com produtividade boa.

1-Rodar o Programa Novamente
2-Sair

```

Figura 7.10: Comparação de permeabilidades seguida da caracterização do reservatório e possibilidade de nova execução ou encerramento do programa.

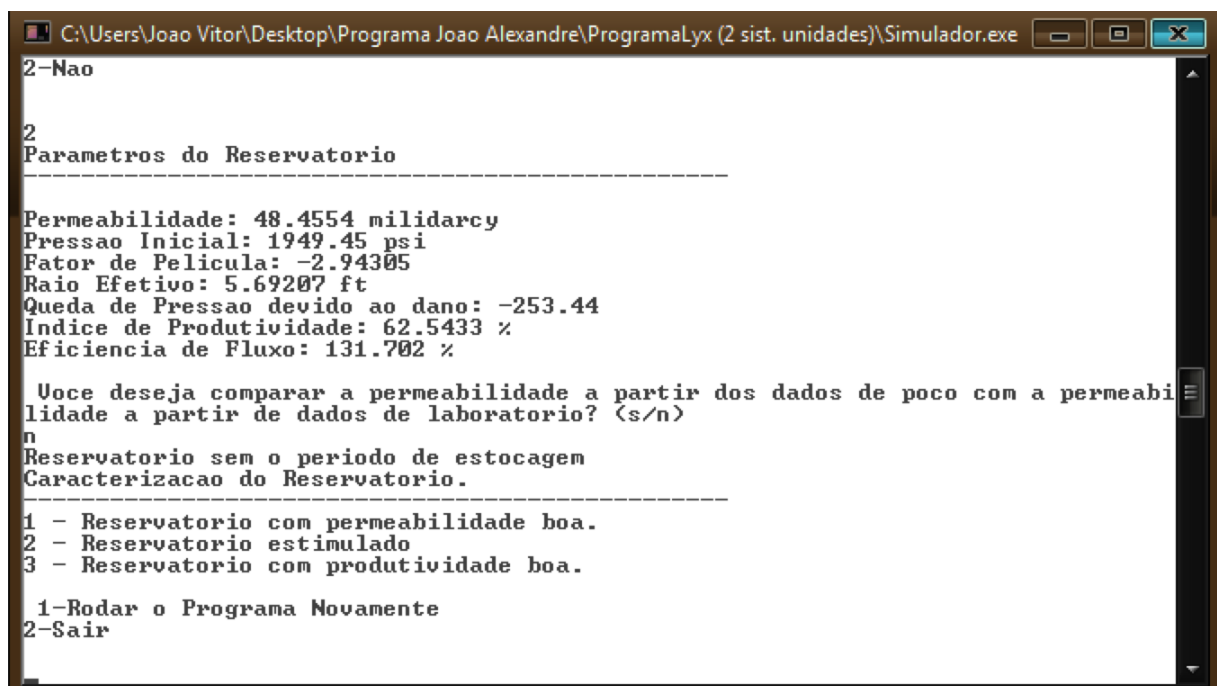


Figura 7.11: O usuário optou por não comparar as permeabilidades e assim obter diretamente a caracterização do reservatório e a opção de nova execução do programa

Capítulo 8

Documentação

A documentação apresentada é referente à execução do software “PROGRAMA EM C++ PARA CARACTERIZAÇÃO DE RESERVATÓRIOS INTEGRADO À ANÁLISE PETROFÍSICA”. Será explicado o passo a passo da utilização do programa.

8.1 Documentação para Usuário

O programa calcula a permeabilidade, o fator de película, a pressão inicial, a eficiência de fluxo, o índice de produtividade, o coeficiente e o tempo de estocagem do reservatório que foi submetido ao teste de pressão (de curta ou longa duração), assim como correlaciona à viscosidade do fluido com temperatura e densidade, além de comparar a permeabilidade obtida à partir da testemunhagem do reservatório. Deve-se assegurar o seguinte para o funcionamento do software:

1. Os dados registrados no registrador de pressão do poço devem ser colocados na pasta do programa, sob qualquer nome, em formato de texto.
2. Item opcional: Para gerar o gráfico característico do reservatório: Ter instalado o software livre “Gnuplot” no computador.
3. Para abrir o programa no sistema operacional “Windows”, simplesmente clique duas vezes em “Simulador.exe”. Caso o sistema operacional utilizado seja o “Linux”, abra o terminal, selecione o caminho da pasta do programa. Digite então: “g++ *.cpp” para compilar os arquivos do programa e em seguida “./a.out” para executar o programa.
4. Ao abrir o arquivo executável, com interação própria do software com o usuário, digite 1 para dar início ao software. Para sair, digite 2 e em seguida ‘enter’.
5. Escolha o sistema de unidades que o registrador de pressão trabalha: Digite 1 para o sistema americano (oilfield) ou 2 para o brasileiro (Petrobras).

6. Escolha se deseja importar a partir de um arquivo .dat os dados do reservatório, poço e fluido ou se deseja informar cada um dos parâmetros.
7. Os passos 8 à 16 serão seguidos somente caso o usuário opte por informar os parâmetros manualmente. Caso os dados sejam importados de um arquivo .dat, prosseguir para passo 18.
8. Informe a porosidade do reservatório e tecle enter.
9. Informe a altura do reservatório e tecle enter.
10. Informe o fator volume formação do fluido e tecle enter.
11. Informe se deseja inserir a viscosidade do fluido (opção 1) ou se deseja calcular à partir de correlações (opção 2).
12. Informe a compressibilidade total (rocha+fluido) e tecle enter.
13. Informe a vazão de produção e tecle enter.
14. Informe o tempo de produção e tecle enter.
15. Informe a pressão no poço e tecle enter.
16. Informe o raio do poço e tecle enter.
17. O programa pedirá para que o usuário entre com o nome do arquivo contendo os dados do registrador de pressão para execução do teste de poço.
18. Nesse momento, o programa fará a regressão linear dos dados importados e mostrará na tela a equação da reta no formato " $y = a * x + b$ ". Ele identificará o período de estocagem do reservatório para não haver erro de cálculo, pois a análise do teste de pressão deve ser feita do período transiente.
19. Escolha se quer que o programa gere o gráfico com o Gnuplot: Digite 1 para sim ou 2 para não e tecle enter.
20. Nesse momento o programa:
 - Mostrará na tela os parâmetros do reservatório calculados, com as devidas unidades.
 - Mostrará na tela os detalhes do período de estocagem do poço, se houver.
21. Em seguida, o usuário terá a opção de comparar os dados de permeabilidade obtidos pelo software com dados reais do testemunho. Digite 's' se deseja comparar, digite 'n' para dar continuidade à caracterização do reservatório

22. Item opcional (caso o usuário decida comparar a permeabilidade): Nos arquivos DadosFluidoRochaCilindro.dat e DadosPermeametroGas.dat, contidos na pasta do software, o usuário deverá inserir os dados obtidos a partir do permeâmetro, para que o programa possa fazer o ajuste linear necessário e encontrar a permeabilidade da amostra.
23. O programa exibirá em tela a interpretação dos resultados do teste no âmbito da caracterização do reservatório. Caso faça a opção por fazer a comparação com a amostra, o programa também calculará o erro resultante dessa comparação e fará uma interpretação à partir deste erro.
24. Digite 1 para rodar o programa novamente ou 2 para sair, tecle enter.

Observação: Se houver uma entrada negativa de valor (equivocada), ou entradas absurdas (acima dos limites superiores utilizados na indústria), o programa pedirá para o usuário fazer nova entrada dos dados.

8.2 Documentação para desenvolvedor

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

8.2.1 Dependências

Para compilar o software é necessário atender as seguintes dependências:

- No sistema operacional GNU/Linux: Instalar o compilador g++ da GNU disponível em <http://gcc.gnu.org>. Para instalar no GNU/Linux use o comando `yum install gcc`. No sistema operacional Windows é necessário instalar um compilador apropriado. Recomenda-se o Dev C++ disponível em <http://dev-c.softonic.com.br/>.
- Biblioteca CGnuplot; os arquivos para acesso a biblioteca CGnuplot devem estar no diretório com os códigos do software;
- O software `gnuplot`, disponível no endereço <http://www.gnuplot.info/>, deve estar instalado. É possível que haja necessidade de setar o caminho para execução do `gnuplot`.
- O programa depende da existência de um arquivo de dados (formato .dat) para preencher os vetores relacionados aos dados do registrador de pressão (Tempo sem produção e pressão medida).
- O programa depende da existência dos arquivos DadosFluidoRochaCilindro.dat e DadosPermeametroGas.dat caso o usuário deseje comparar os dados obtidos do teste de poço com a permeabilidade de um testemunho do reservatório.

8.2.2 Documentação usando doxygen

A documentação do código do software foi feita usando o padrão JAVADOC. Depois de documentar o código, o software *doxygen* foi usado 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.

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

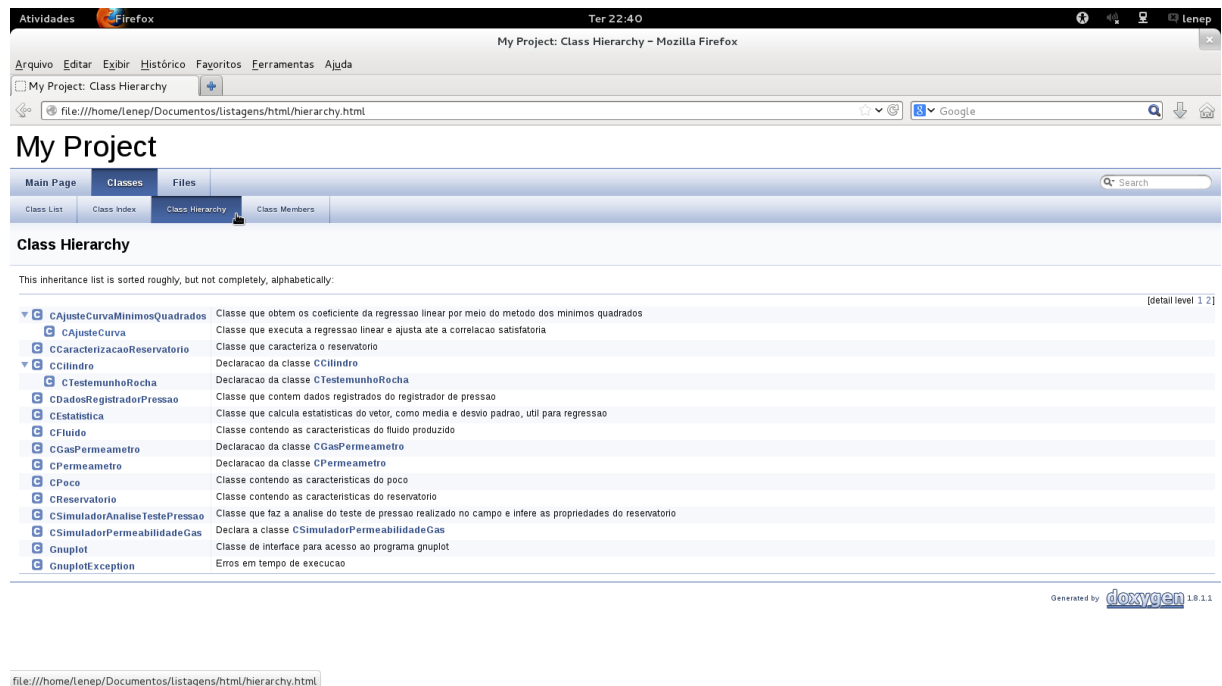


Figura 8.1: Hierarquia de classes

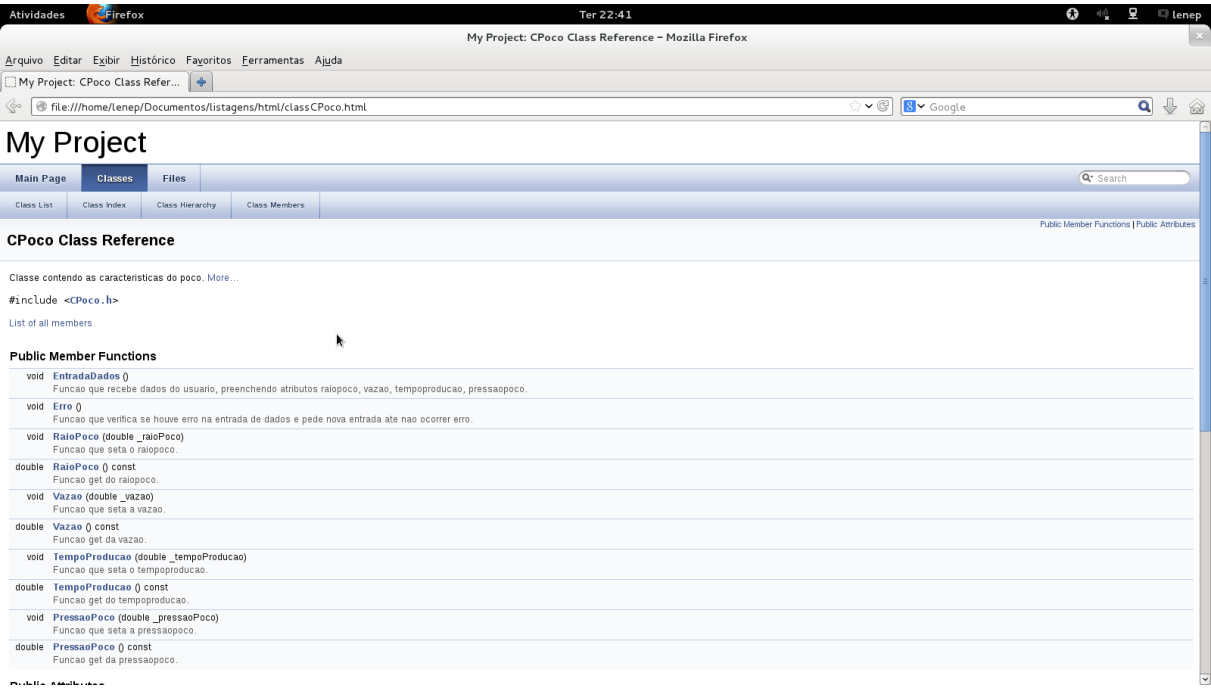


Figura 8.2: Exemplo de características de uma classe CPoco)

Referências Bibliográficas

- [Adalberto José Rosa, 1987] Adalberto José Rosa, A. C. d. F. C. (1987). *Análise de testes de pressão em poços*. Petrobras. 8
- [Adalberto Rosa, 2006] Adalberto Rosa, Renato Carvalho, D. X. (2006). *Engenharia de Reservatórios de petróleo*. Interciência, Rio de Janeiro. 10
- [Bueno, 2003] Bueno, A. D. (2003). *Programação Orientada a Objeto com C++ - Aprenda a Programar em Ambiente Multiplataforma com Software Livre*. Novatec, São Paulo.
- [de Souza, 2012] de Souza, F. R. (2012). Programa para cálculo da permeabilidade e porosidade de rochas a partir de dados experimentais.
- [e Farshad, 1993] e Farshad, P. (1993). Viscosity correlations for gulf of mexico crude oils. 12
- [e Schellardt, 1936] e Schellardt, R. (1936).
- [Ferraz, 2012] Ferraz, M. K. (2012). Programa para calculo de parametros de reservatorios por dados de teste de pressao. 1, 9
- [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.
- [LyX-Team, 2004] LyX-Team, editor (2004). *The LyX User's Guide*. LyX Team - <http://www.lyx.org>.
- [Steding-Jessen, 2000] Steding-Jessen, K. (2000). *Latex demo: Exemplo com Latex 2e*.
- [Álvaro M. M. Peres, 2008] Álvaro M. M. Peres (2008). Teoria dos testes de pressão em poços. Rio de Janeiro. Anais do VI Encontro ENGEF. 10