

o co
UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE DARCY
RIBEIRO
LABORATÓRIO DE ENGENHARIA E EXPLORAÇÃO DE PETRÓLEO
CENTRO DE CIÊNCIA E TECNOLOGIA

SOFTWARE EDUCACIONAL PARA ANÁLISE
E SOLUÇÃO DE PROBLEMAS EM ENGENHARIA DE POÇO

TRABALHO DE CONCLUSÃO DE CURSO

Versão 1:

NATHAN RANGEL MAGALHÃES
THAUAN FERREIRA BARBOSA;

Versão 2:

NATHAN RANGEL MAGALHÃES

Orientador: André Duarte Bueno

MACAÉ - RJ

Maio - 2025

UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE DARCY
RIBEIRO
LABORATÓRIO DE ENGENHARIA E EXPLORAÇÃO DE PETRÓLEO
CENTRO DE CIÊNCIA E TECNOLOGIA

NATHAN RANGEL MAGALHÃES

Trabalho de Conclusão de Curso apresentado ao Centro de Ciências e Tecnologia da Universidade Estadual do Norte Fluminense Darcy Ribeiro, como parte das exigências para obtenção do Título de Engenheiro de Exploração e Produção de Petróleo.

Orientador: André Duarte Bueno, D.Sc.

Macaé - RJ
Maio - 2025

Dedico este trabalho aos meus pais, que, sob o sol, lutaram para que eu pudesse caminhar na sombra; aos meus avós, que sempre me apoiaram e foram luz no meu caminho; e à minha esposa, que foi meu alicerce nos momentos em que achei que não conseguiria.

Resumo

O presente trabalho descreve o desenvolvimento de um software educacional voltado para alunos de Engenharia de Petróleo e áreas afins, com ênfase no estudo e na aplicação dos principais conceitos da disciplina de Engenharia de Poços. O software reúne ferramentas de visualização, análise, simulação e cálculo, abordando os principais tópicos da disciplina LEP01353, ministrada no Laboratório de Engenharia e Exploração de Petróleo (LENEP/UENF) desde o período letivo 2024/01.

Com uma interface intuitiva, o sistema visa facilitar o aprendizado e o aprofundamento dos usuários, podendo ser utilizado tanto por estudantes da disciplina quanto por outros interessados, independentemente de sua vinculação institucional. Sua proposta é contribuir com o ensino de Engenharia de Poços em contextos didáticos diversos, reforçando o caráter educacional do projeto.

Este documento apresenta a estruturação e o desenvolvimento do projeto, destacando as metodologias utilizadas para garantir sua funcionalidade e relevância no processo de ensino-aprendizagem. O desenvolvimento foi conduzido com base em metodologias ágeis e no uso de controle de versões via Git/GitHub, práticas amplamente adotadas no mercado de trabalho. O software estará disponível publicamente por meio de repositório específico, conforme o link abaixo:

<https://github.com/lpsc/SoftwareEducacionalEngenhariaDePoco>.

Palavras-chave: Simulador educacional. Engenharia de Poços. Visualização e análise. Simulação. Ensino de Engenharia de Petróleo.

Listas de Figuras

1.1	Etapas para o desenvolvimento do software - <i>projeto de engenharia</i>	15
2.1	Diagrama de caso de uso – caso de uso geral	20
2.2	Diagrama de caso de uso específico: cálculo de pressão hidrostática e densidade	21
2.3	Diagrama de caso de uso específico - cálculo da perda de pressão devido a perda por fricção no poço e no anular	22
3.1	Diagrama de corpo livre, atuação de forças em um elemento de fluido	29
3.2	Coluna composta por fluidos com diferentes características	31
3.3	Fluxo laminar de fluido Newtoniano	33
3.4	Diagrama de pacotes	41
4.1	Diagrama de classes	44
4.2	Diagrama de sequência	47
4.3	Diagrama de comunicação	48
4.4	Diagrama de máquina de estado	49
4.5	Diagrama de atividades	49
5.1	Diagrama de componentes	51
6.1	Versão 0.1, interface do software	53
6.2	Versão 0.2, interface do software	54
6.3	Versão 1.0, interface do software	55
6.4	Versão 1.1, interface do software	57
6.5	Versão 1.1, Menu de interface do software	59
6.6	Versão 1.1, interface do modulo 01 do software	60
6.7	Versão 1.1, interface do modulo 02 software	61
8.1	Soluções geradas pelo código para modelo Newtoniano no poço considerando regime laminar	150
8.2	Soluções geradas pelo código para modelo Newtoniano no anular considerando regime laminar	152

8.3 Soluções geradas pelo código para modelo Newtoniano no anular considerando regime turbulento	154
9.1 Logo e documentação do <i>software</i>	157
9.2 Código fonte da classe CSimuladorPoco, no <i>Doxxygen</i>	157
10.1 Versão 1.1, Menu de interface do software	159
10.2 Versão 1.1, interface do modulo 01 do software	160
10.3 Versão 1.1, interface do modulo 02 software	161
10.4 Acesso às funcionalidades de configuração do poço e dos fluidos	163
10.5 Exemplo de estrutura de arquivo .dat para importação de dados	164
10.6 Exemplo de estrutura de arquivo .dat para importação de dados	165

Lista de Tabelas

2.1	Características básicas do Software Educacional Para Análise e Solução de Problemas em Engenharia de Poço. Fonte: Elaborado pelo autor (2025).	17
2.2	Caso de uso 1	19

Sumário

1	Introdução	13
1.1	Escopo do problema	13
1.2	Objetivos	13
1.3	Metodologia utilizada	14
2	Concepção	16
2.1	Nome do sistema/produto	16
2.2	Especificação	17
2.3	Requisitos	17
2.3.1	Requisitos Funcionais	17
2.3.2	Requisitos Não Funcionais	18
2.4	Casos de Uso do Software	18
2.4.1	Diagrama de caso de uso geral	19
2.4.2	Diagrama de caso de uso específico	19
3	Elaboração	23
3.1	Análise de Domínio	23
3.2	Formulação Teórica	24
3.2.1	Ementa da disciplina	24
3.2.2	Termos e Unidades	26
3.2.3	Hidráulica de Perfuração	28
3.2.4	Pressão Hidrostática	28
3.2.5	Pressão Hidrostática em Colunas Com Mais de Um Tipo de Fluido	30
3.2.6	Densidade Equivalente	32
3.2.7	Modelos Reológicos de Fluidos de Perfuração	32
3.2.8	Perda de Pressão Friccional em um Tubo de Perfuração	35
3.2.9	Perda de Pressão Friccional em um Anular	36
3.2.10	Variações de Carga Axial e Deslocamento em Colunas de Poço . . .	38
3.3	Identificação de Pacotes – Assuntos	40
3.4	Diagrama de Pacotes – Assuntos	41

4 AOO – Análise Orientada a Objeto	43
4.1 Diagramas de classes	43
4.1.1 Dicionário de Classes	45
4.2 Diagrama de Sequência – Eventos e Mensagens	46
4.2.1 Diagrama de Sequência Geral	46
4.3 Diagrama de Comunicação – Colaboração	48
4.4 Diagrama de Máquina de Estado	48
4.5 Diagrama de Atividades	49
5 Projeto	50
5.1 Projeto do sistema	50
5.2 Diagrama de componentes	51
6 Ciclos de planejamento/detalhamento	52
6.1 Versão Inicial – Interação via Terminal e Geração de Gráficos com Gnuplot	52
6.2 Versão 0.2 – Otimização de Entrada, Validação e Armazenamento Automático de Dados	53
6.3 Versão 1.0 – Interface Gráfica, Amigável e Intuitiva Utilizando o Framework Qt Creator	54
6.4 Versão 1.1 – Consolidação Visual, Barra de Tarefas e Automação de Processos	55
6.5 Versão 2.0 – Navegação por Módulos e Simulação Mecânica de Variação de Comprimento (ΔL)	57
7 Ciclos Construção - Implementação	62
7.1 Código-Fonte (modelo)	62
7.1.1 Código Qt (interface)	132
8 Resultados	148
8.1 Metodologia de Validação	148
8.2 Casos de Teste	149
8.2.1 Validação da pressão Hidroestática	149
8.2.2 Validação da perda de fricção pelo modelo Newtoniano no Poço e Anular	150
8.2.3 Validação da perda de fricção pelo modelo Bingham no Poço e Anular	152
8.2.4 Validação da perda de fricção pelo modelo Potência no Poço e Anular	154
8.2.5 Validação MOD2 delta temperatura, efeito balão...	154
8.2.6 Validação importação e exportação do documento (salvar como...) .	154
8.3 Conclusão	154

9 Documentação	156
9.1 Documentação do usuário	156
9.2 Documentação do desenvolvedor	156
10 Manual do usuário	158
10.1 Instalação	158
10.1.1 Dependências	158
10.2 Interface gráfica	158
10.3 Funcionalidades	162
10.3.1 Módulo 1 – Hidráulica de Perfuração	162
Referências Bibliográficas	166

Capítulo 1

Introdução

O presente projeto de engenharia propõe o desenvolvimento de um software educacional voltado ao apoio didático na disciplina de Engenharia de Poço, no contexto da Engenharia de Petróleo. Utilizando a linguagem C++ e o paradigma da programação orientada a objetos, a aplicação tem como objetivo facilitar a assimilação de conceitos complexos por meio de simulações computacionais e recursos interativos.

A ferramenta busca aproximar a teoria da prática, simulando condições operacionais típicas da área, como o comportamento de fluidos em diferentes regimes de escoamento e os efeitos de variações térmicas e mecânicas sobre a coluna de completação. Espera-se, com isso, ampliar a compreensão dos alunos e tornar o aprendizado mais dinâmico e intuitivo.

1.1 Escopo do problema

Tradicionalmente, o ensino da disciplina de Engenharia de Poço baseia-se em exercícios teóricos e análises manuais. Entretanto, essa abordagem apresenta limitações, especialmente na visualização de fenômenos complexos e na aplicação dos conceitos em contextos reais. A ausência de ferramentas computacionais que permitam simulações e manipulação de parâmetros dificulta a assimilação por parte dos estudantes.

Nesse contexto, o software proposto surge como uma resposta a essas dificuldades, oferecendo uma plataforma de apoio educacional que possibilita a realização de simulações interativas. A ferramenta promove um aprendizado mais dinâmico e eficaz, contribuindo significativamente para a formação acadêmica ao apresentar de forma prática os efeitos e as interações dos principais parâmetros operacionais de um poço.

1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:

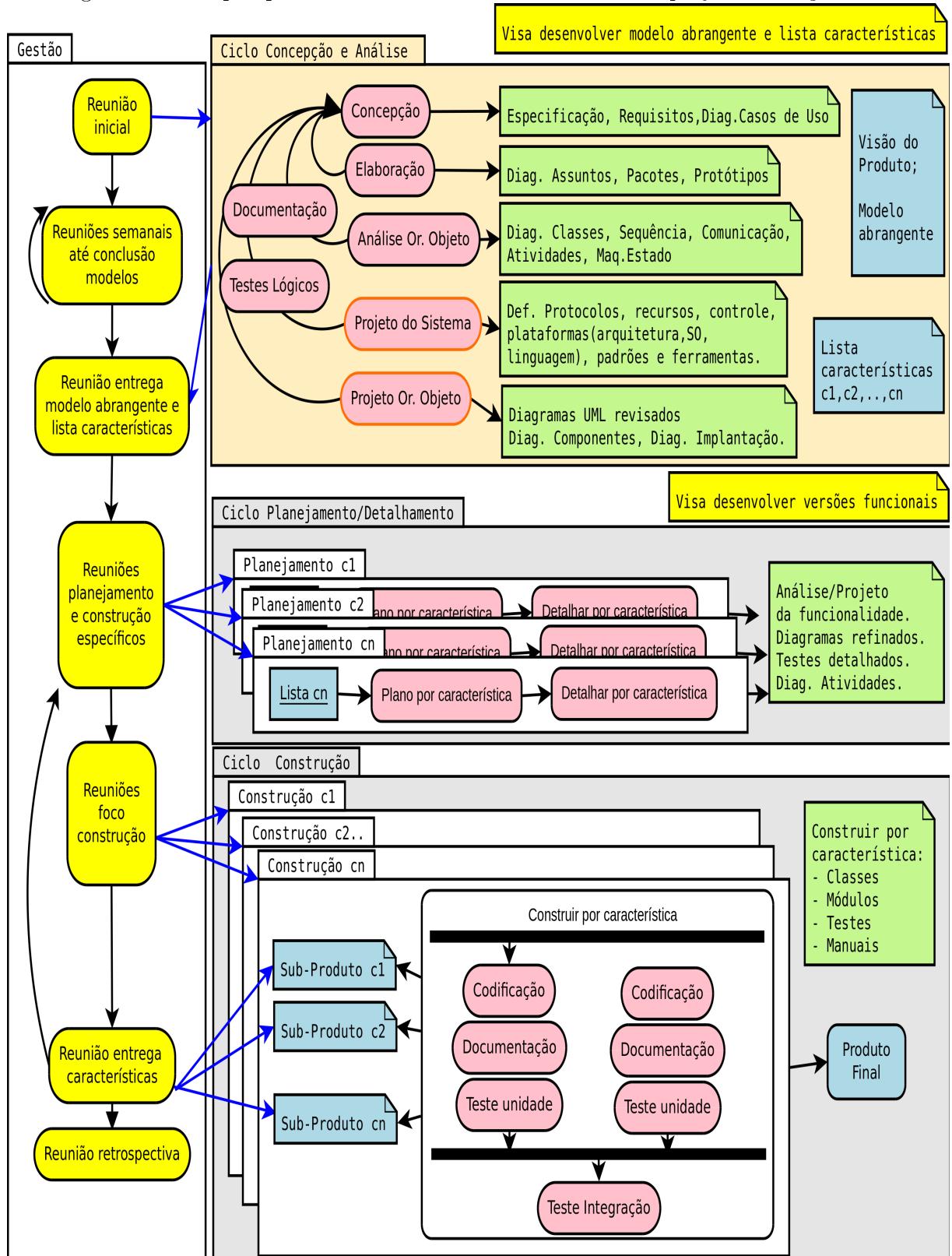
- Desenvolver um software capaz de analisar e calcular as principais equações que sustentam os fundamentos da disciplina de Engenharia de Poço, favorecendo a consolidação do conhecimento teórico adquirido em sala de aula.
- Objetivos específicos:
 - Modelar física e matematicamente os problemas abordados, incluindo a definição das propriedades relevantes a serem avaliadas..
 - Realizar a modelagem estática e dinâmica da estrutura do software.
 - Calcular propriedades hidrodinâmicas e reológicas associadas ao poço.
 - Realizar simulações computacionais para teste e validação dos algoritmos desenvolvidos.
 - Desenvolver um manual simplificado para orientar o uso do software.

1.3 Metodologia utilizada

A metodologia empregada para o desenvolvimento do software fundamenta-se nos ciclos propostos pelo Prof. André Duarte Bueno, cujos materiais foram disponibilizados via GitHub e complementados por conteúdos ministrados nas disciplinas Introdução a Projetos: Metodologia Científica e Projeto de Softwares (LEP01582) e Programação Orientada a Objetos com C++ (LEP01447).

O processo foi estruturado em três fases: concepção e análise, planejamento detalhado e implementação. O uso da linguagem C++ com o framework Qt viabilizou a construção de interfaces gráficas intuitivas, enquanto práticas modernas de controle de versão, com Git e GitHub, garantiram rastreabilidade e organização modular do código.

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



Fonte: Apostila da disciplina "Programação Orientada a Objeto em C++" do professor André Duarte Bueno

Capítulo 2

Concepção

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

2.1 Nome do sistema/produto

O sistema desenvolvido, denominado Software Educacional para Análise e Solução de Problemas em Engenharia de Poço, foi construído em linguagem C++ com uso do framework Qt. Essa base tecnológica permitiu a criação de uma interface gráfica robusta, que facilita a navegação e oferece recursos visuais para apoio ao ensino.

Entre suas funcionalidades, o programa realiza o cálculo de propriedades como pressão hidrostática, densidade e viscosidade média dos fluidos, além de permitir a seleção entre diferentes modelos reológicos: newtoniano, plástico de Bingham e lei das potências. Também simula cenários operacionais com variações de temperatura e pressão, incluindo efeitos como deslocamentos axiais (ΔL) da coluna de completação, efeito balão e atuação de packers e pistões.

O sistema aceita entrada de dados por meio de arquivos .dat, oferece visualização gráfica dos resultados e permite a exportação dos dados, apoiando tanto o estudo individual quanto a elaboração de relatórios acadêmicos.

A Tabela 2.1 apresenta as características do software desenvolvido.

Tabela 2.1: Características básicas do Software Educacional Para Análise e Solução de Problemas em Engenharia de Poço. Fonte: Elaborado pelo autor (2025).

Nome	Software Educacional Para Análise e Solução de Problemas em Engenharia de Poço
Componentes principais	Banco de dados com métodos e propriedades da Engenharia de Poço. Algoritmo de aproximação de resultados. Interface gráfica para o plotar resultados. Saída gráfica e em arquivo .dat.
Missão	A missão do software é fornecer uma ferramenta eficiente para potencializar o aprendizado de alunos que buscam se aprofundar nos conceitos de engenharia de poço. O software oferece uma ferramenta didática para a engenharia de petróleo.

2.2 Especificação

O software educacional desenvolvido apresenta uma interface gráfica intuitiva que permite ao usuário selecionar, a qualquer momento, as funções desejadas. As operações implementadas baseiam-se em modelos e equações consolidados na área de Engenharia de Poço, conforme a ementa da disciplina (LEP01353/2024-01).

2.3 Requisitos

Apresenta-se a seguir os requisitos funcionais e não funcionais.

2.3.1 Requisitos Funcionais

Apresenta-se a seguir os requisitos funcionais

RF-01	O sistema deve conter uma base de dados confiáveis retiradas de referências bibliográficas como Mitchell & Miska (2011) e Jr. <i>et al.</i> (1991).
RF-02	O usuário poderá carregar dados da propriedade para a simulação.
RF-03	O usuário deverá ter liberdade para alterar as propriedades reológicas do poço/fluido.
RF-04	Deve permitir a exportação de simulações.

RF-05	Deve permitir cenários de simulação baseado em diferentes modelos teóricos.
RF-06	O usuário poderá comparar os resultados da simulação em diferentes modelos reológicos.
RF-07	O usuário deve ter liberdade para adicionar ou retirar simplificações das premissas do modelo.
RF-08	O usuário poderá visualizar seus resultados em um gráfico. O gráfico poderá ser salvo como imagem.
RF-09	O sistema deve permitir a análise dos deslocamentos axiais (ΔL) da coluna de completação em diferentes condições operacionais.
RF-10	O sistema deve contemplar variações térmicas, efeito balão, atuação de pistão, packer e crossover nas análises de carga.

2.3.2 Requisitos Não Funcionais

RNF-01	Suas primeiras versões devem suportar os sistemas operacionais Linux e <i>Windows</i> .
RNF-02	A linguagem predominante a ser utilizada no desenvolvimento é C++.
RNF-03	A geração dos gráficos deve ser realizada por meio da biblioteca QCustomPlot.
RNF-04	O sistema deve permitir a exportação dos resultados em arquivos de texto e no formato .dat.
RNF-05	A interface gráfica deve ser desenvolvida com o Qt Framework, oferecendo usabilidade intuitiva.
RNF-06	O sistema deve manter organização modular do código, facilitando manutenção e futuras expansões.
RNF-07	Os arquivos de entrada e saída devem seguir padrões consistentes e documentados para garantir interoperabilidade.

2.4 Casos de Uso do Software

Nesta seção iremos mostrar o caso de uso do software a ser desenvolvido.

2.4.1 Diagrama de caso de uso geral

O diagrama de caso de uso geral da Figura 2.1 mostra o usuário de frente a interface com as opções permitidas do simulador. Com essas opções ele poderá executar, analisar os resultados obtidos e salvar as imagens ou os dados em um arquivo PDF. As condições do caso de uso são apresentadas na Tabela 2.2.

Tabela 2.2: Caso de uso 1

Nome do caso de uso:	Simulação das propriedades de fluido e poço
Resumo/descrição:	Calcular as propriedades de fluido e poço para diferentes condições
Etapas:	<ol style="list-style-type: none">1. Adicionar propriedades do fluido2. Adicionar propriedades do poço3. Incluir diferentes tipos de fluidos no poço4. Calcular pressão hidrostática do poço5. Calcular densidade do fluido6. Calcular a queda de pressão devido a perdas por fricção6. Plotar perfis de poço7. Salvar dados em saída .dat

2.4.2 Diagrama de caso de uso específico

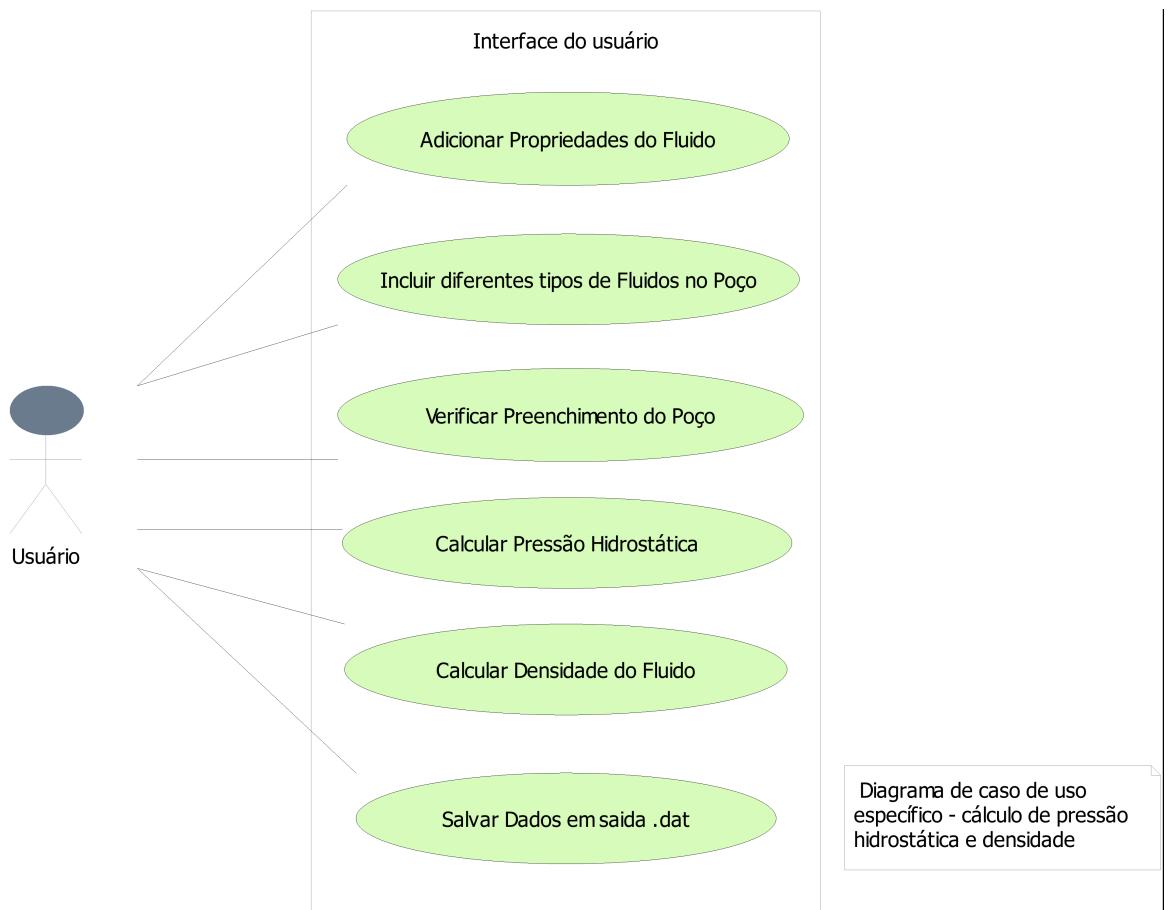
O caso de uso específico da Figura 2.2 mostra o cenário onde o usurário deseja calcular a pressão hidrostática e a densidade dos fluidos configurados no poço.

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



Fonte: Produzido pelo autor.

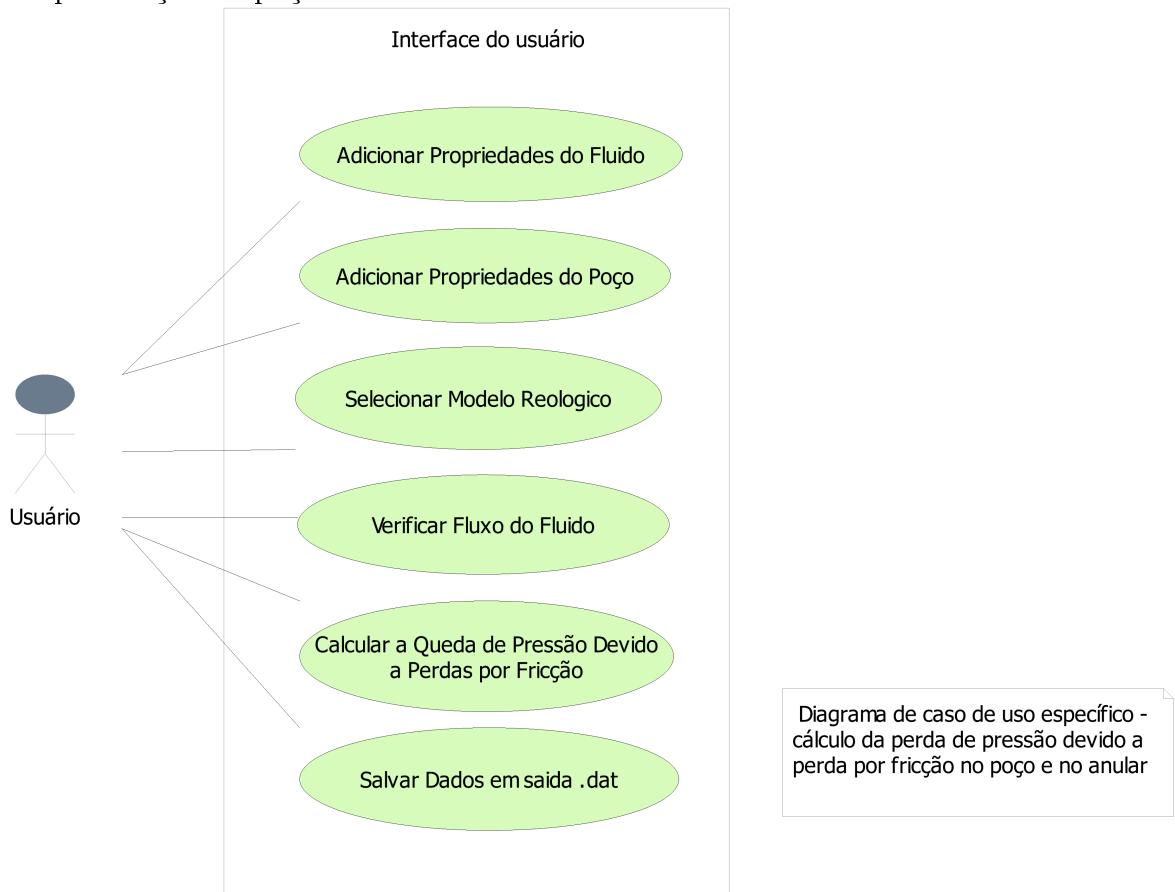
Figura 2.2: Diagrama de caso de uso específico: cálculo de pressão hidrostática e densidade



Fonte: Produzido pelo autor.

O caso de uso específico da Figura 2.3 mostra o cenário onde o usuário deseja calcular a perda de pressão devido a perda por fricção no poço e no anular.

Figura 2.3: Diagrama de caso de uso específico - cálculo da perda de pressão devido a perda por fricção no poço e no anular



Fonte: Produzido pelo autor.

Capítulo 3

Elaboração

Neste capítulo, apresenta-se a elaboração do simulador, abrangendo o desenvolvimento teórico, a formulação das equações analíticas, a definição dos pacotes computacionais utilizados e os algoritmos adicionais integrados ao software.

3.1 Análise de Domínio

A análise de domínio é uma etapa essencial no desenvolvimento de um projeto, pois envolve a identificação e compreensão dos conceitos fundamentais que orientarão a construção do simulador. Essa fase permite mapear os elementos-chave do problema, definindo as entidades, relações e comportamentos que deverão ser representados no sistema.

Este projeto está vinculado a cinco conceitos essenciais:

1. Mecânica dos fluidos:

No contexto da engenharia de perfuração, a mecânica dos fluidos trata do comportamento dos líquidos sob diferentes condições de temperatura, pressão e velocidade, considerando também a presença de sólidos como cascalho e cimento. Os fluidos utilizados nas operações têm papel fundamental na estabilização da perfuração, controle de pressões, remoção de cascalho e aplicação de cimento. Neste projeto, calcula-se a pressão dos fluidos considerando uma condição padrão: o estado de repouso da coluna e do fluido.

2. Mecânica das rochas:

A mecânica das rochas é crucial para entender o comportamento das formações geológicas durante a perfuração. Essa análise permite avaliar a estabilidade do poço, prevenir colapsos e falhas no revestimento, além de estimar tensões e fraturas que possam comprometer a integridade da operação. Conhecimentos sobre compressão, cisalhamento e tensões atuantes são fundamentais para garantir a segurança da estrutura do poço.

3. Equações analíticas:

As equações analíticas oferecem modelos matemáticos que permitem prever e controlar aspectos operacionais, como o escoamento de fluidos e o comportamento das rochas. Equações como a Lei de Darcy e a equação de Bernoulli são aplicadas para calcular perdas de carga, pressões hidrostáticas e tensões nas paredes do poço. Esses cálculos são indispensáveis para prever fraturas, definir pressões de poro e garantir a estabilidade do sistema.

4. Programação:

A programação orientada a objetos (POO) é a base do desenvolvimento do simulador, promovendo modularidade, reutilização de código e manutenção facilitada. A linguagem C++ foi adotada por sua robustez, alto desempenho e compatibilidade com bibliotecas como Qt (para interfaces gráficas) e Gnuplot (para gráficos científicos). Com conceitos como herança, polimorfismo e encapsulamento, a POO permite a estruturação eficiente dos componentes do simulador.

5. Modelagem Gráfica:

A modelagem gráfica é fundamental para representar visualmente fenômenos complexos, facilitando a análise e a tomada de decisões. Através de gráficos 2D e 3D, é possível visualizar perfis de pressão, porosidade, velocidade e ondas sísmicas em formações geológicas. A integração com a API Qt permite criar uma interface intuitiva e interativa, essencial para o uso didático e técnico do simulador.

3.2 Formulação Teórica

3.2.1 Ementa da disciplina

A seguir dados da ementa da disciplina cujo software atende.

- Dados básicos:

- Sigla: LEP01353
- Nome: Engenharia de Poço
- Centro: CCT - Centro de Ciência e Tecnologia
- Laboratório: CCT/LENEP - Laboratório de Engenharia e Exploração de Petróleo
- Criação: 2024/1, 01/01/2024
- Horas teórica: 68
- Horas prática: 0

- Horas extra classe: 0
- Horas extensão: 0
- Carga horária total: 68
- Créditos: 4
- Tipo de aprovação: Média/Frequência

- Objetivos:

- Conhecer os tipos de sonda de perfuração de poços de petróleo; conhecer as funções dos componentes da coluna de perfuração e de completação; conhecer o processo de cimentação de poços de poços; conhecer as propriedades, funções e características dos fluidos de perfuração; modelar o escoamento do fluido de perfuração no espaço anular e dentro da coluna de perfuração; analisar a estabilidade de poços de petróleo. calcular as tensões na coluna de produção; selecionar a metalurgia ideal para a completação de poços.

- Ementa resumida:

- Introdução à perfuração e completação de poços de petróleo; Fluidos de perfuração e completação de poços de petróleo; Hidráulica de perfuração;
- Estabilidade mecânica durante a perfuração de poços de petróleo;
- Seleção de materiais para completação de poços de petróleo; Análise de tensões na coluna de produção;

- Conteúdo programático:

- Introdução à perfuração de poços de petróleo:
 - * Tipos de sonda de perfuração;
 - * Elementos da coluna de perfuração e produção;
 - * Cimentação;
- Fluidos de perfuração e completação de poços de petróleo:
 - * Composição dos fluidos de perfuração;
 - * Função e características dos fluidos;
 - * Dano de formação;
 - * Reologia;
 - * Filtração estática e dinâmica;
- Hidráulica de perfuração:
 - * Transporte de cascalho;
 - * Fluxo não-newtoniano dentro da coluna de perfuração e no espaço anular;

- Estabilidade mecânica de poços de petróleo:
 - * Introdução à mecânica das rochas;
 - * Gradiente de sobrecarga e de pressão de poros;
 - * Tensões ao redor de um poço;
 - Introdução à completação de poços de petróleo:
 - * Elementos da coluna de produção;
 - * Operações de completação de poços;
 - Seleção de materiais para completação de poços de petróleo:
 - * Tipo de metais;
 - * Corrosão;
 - * Seleção de metalurgia.
 - Análise de tensões na coluna de produção:
 - * Carga axial;
 - * Colapso e explosão da coluna de produção;
 - * Fatores de segurança;
 - * Obturadores;
 - Tópicos especiais
 - Avaliação do curso
 - Provas escritas
- Bibliografia
- BELLARBY, J.: Well completion design. Amsterdam: Elsevier, 2009.
 - BOURGOYNE, A.; MILLHEIM, K.K.; CHENEVERT, M.E. YOUNG JR., F.D. Applied drilling engineering. Richardson, TX: Society of Petroleum Engineers, 1986.
 - GRAY, G.R.; DARLEY, H.; CAENN, R. Fluidos de perfuração e completação. Rio de Janeiro: LTC, 2014.
 - RENPU, W. Engenharia de completação de poços. Amsterdam: Elsevier, 2017.
 - ROCHA, L.A.S.; AZEVEDO, C.T.: Projetos de poços de petróleo. Geopressões e assentamento de colunas de revestimento. Rio de Janeiro: Interciência, 2019.

3.2.2 Termos e Unidades

Os principais termos e suas unidades utilizadas neste projeto estão listadas abaixo:

- dp é a variação de pressão [psi];
- dZ é a variação de profundidade [ft];
- ρ é a densidade do fluido [lbm/gal];
- p_0 é a constante de integração igual a pressão na superfície [psi];
- p é a pressão [psi];
- Z é a profundidade [ft];
- z é o fator de desvio de gás;
- R é constante universal dos gases [$\text{psi} \cdot \text{ft}^3/\text{lb} - \text{mol} \cdot ^\circ R$];
- T é a temperatura absoluta [${}^\circ R$];
- M é o peso molecular do gás [$\text{lb/lb} - \text{mol}$];
- ΔZ é a variação de profundidade [ft];
- g é a gravidade [ft/s^2];
- v velocidade [ft/s];
- τ é a tensão de cisalhamento exercida sobre o fluido [psi];
- μ é a viscosidade aparente [cP];
- $\dot{\gamma}$ é a taxa de cisalhamento [$1/s$];
- τ_y é a tensão de escoamento ou o ponto de escoamento [$\text{lbf}/100.\text{sq.ft}$];
- μ_p é a viscosidade plástica [cP];
- K é o índice de consistência do fluido [cP];
- n é o expoente da lei de potência ou o índice de comportamento do fluxo;
- N_{re} é o número de Reynolds;
- d é o diâmetro interno do revestimento ID [in];
- \bar{v} é a velocidade média [ft/s];
- q é a vazão do poço [gal/min];
- $\frac{dp_f}{dL}$ é a perda de pressão por fricção [psi/ft];
- f é o fator de fricção;

- τ_w é a tensão de cisalhamento na parede [lb/ft^2];
- N_{rec} é o número de Reynolds crítico;
- N_{He} é o número de Hedstrom;
- d_1 é o diâmetro externo do revestimento OD [in];
- d_2 é o diâmetro do poço [in];
- d_2 é o diâmetro do poço [in];

3.2.3 Hidráulica de Perfuração

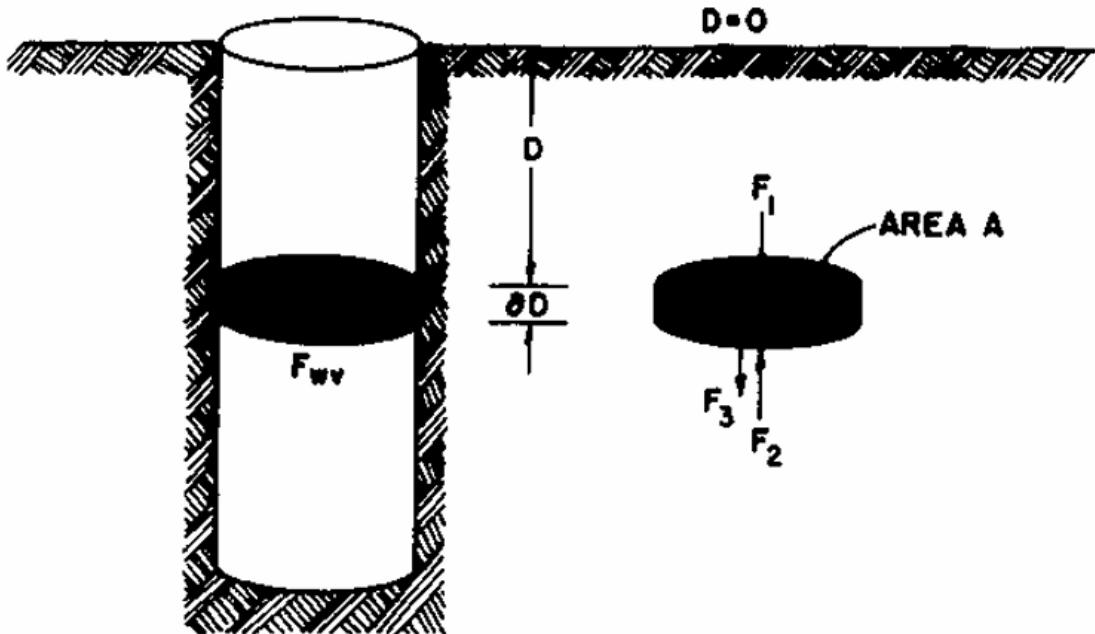
Na engenharia de perfuração, um fluido de perfuração possui três funções principais: transportar cascalho, prevenir o influxo de fluidos e manter a estabilidade do poço. Para cumprir essas funções, o fluido depende do seu escoamento na tubulação e das pressões associadas. Para que o engenheiro possa formular o fluido mais adequado a cada situação específica, é essencial que ele seja capaz de prever as pressões e os escoamentos ao longo do poço.

Os fluidos de perfuração podem variar amplamente em termos de composição e propriedades, indo desde fluidos incompressíveis, como a água, até fluidos altamente compressíveis, como a espuma. O simulador desenvolvido neste trabalho se propõe a resolver dois tipos de problemas: os estáticos, que envolvem o cálculo da pressão hidrostática, e os dinâmicos, relacionados ao escoamento dos fluidos no interior da tubulação.

3.2.4 Pressão Hidrostática

A pressão hidrostática corresponde à variação da pressão em função da profundidade ao longo de uma coluna de fluido, sendo comumente mais facilmente calculada em condições de poço estático. Sua dedução pode ser realizada a partir da análise do diagrama de corpo livre apresentado na Figura 3.1.

Figura 3.1: Diagrama de corpo livre, atuação de forças em um elemento de fluido



Fonte Jr. et al. (1991)

A partir dessa dedução chegamos à Equação (3.1) a seguir em unidades *oil field*, onde dp é a variação de pressão [psi], dZ é a variação de profundidade[ft] e ρ é a densidade do fluido [lb/gal].

$$\frac{dp}{dZ} = 0.05195\rho \quad (3.1)$$

Podemos calcular a pressão hidrostática para dois tipos de fluidos, os incompressíveis e os fluidos compressíveis.

Fluidos incompressíveis

Sabemos que alguns fluidos utilizados como lama de perfuração apresentam um comportamento aproximadamente incompressível, como é o caso da água salgada. Nesses casos, a compressibilidade do fluido em baixas temperaturas pode ser desprezada, permitindo considerar o peso específico constante ao longo da profundidade. Assim, a partir da integração da Equação (3.1), obtém-se a equação hidrostática para fluidos incompressíveis:

$$p = 0.05195\rho Z + p_0 \quad (3.2)$$

Onde p_0 é a constante de integração, igual a pressão na superfície [psi], p é a pressão [psi] e Z é a profundidade [ft]. Uma importante aplicação dessa equação é determinar a densidade correta de um fluido de perfuração, de modo que ele seja capaz de evitar o influxo de fluidos da formação para o poço, prevenindo, assim, situações de *kik* ou

blowout, além de não causar fraturas na formação as quais poderiam provocar uma perda de circulação de fluido que também é indesejada Jr. *et al.* (1991).

Fluidos compressíveis

Em diversas operações de perfuração ou completação, a presença de gás é comum, seja por injeção planejada ou por fluxo proveniente de formações geológicas. O cálculo da pressão hidrostática em uma coluna de gás estática é mais complexo, pois a compressibilidade do gás faz com que sua densidade varie com a pressão. Para representar esse comportamento, utiliza-se a equação do gás real:

$$p = \rho z \frac{RT}{M} \quad (3.3)$$

Onde z é o fator de desvio de gás, R é constante universal dos gases [$\text{psi} \cdot \text{ft}^3/\text{lb} - \text{mol} \cdot ^\circ\text{R}$], T é a temperatura absoluta [$^\circ\text{R}$] e M é o peso molecular do gás [$\text{lb}/\text{lb} - \text{mol}$] Jr. *et al.* (1991).

Realizando a combinação da equação da pressão hidrostática para fluidos incompressíveis e da equação do gás real chegamos a seguinte equação da pressão hidrostática para fluidos compressíveis:

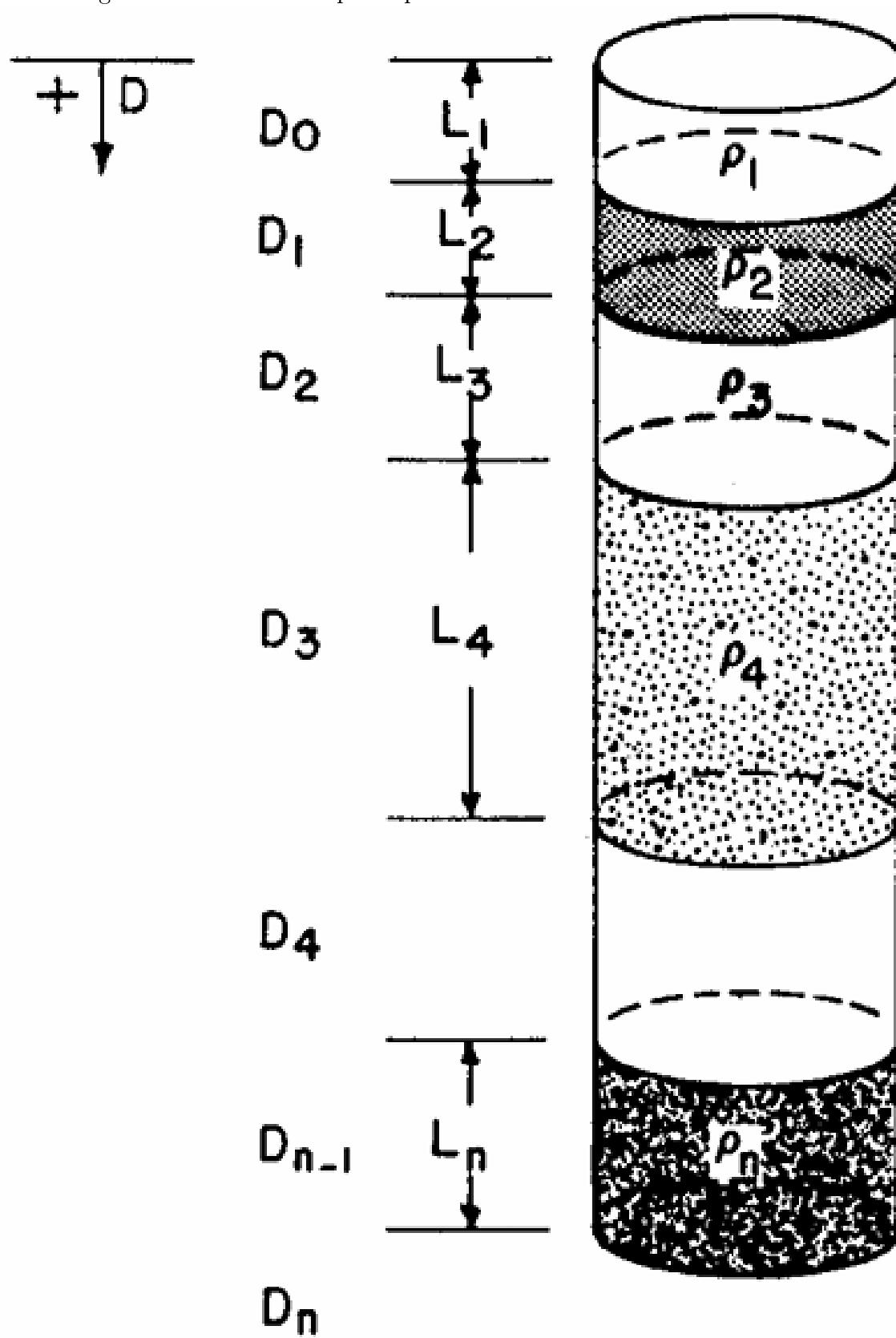
$$p = p_0 \exp \left(\frac{M \Delta Z}{1544 z T} \right) \quad (3.4)$$

Onde ΔZ é a variação de profundidade [ft].

3.2.5 Pressão Hidrostática em Colunas Com Mais de Um Tipo de Fluido

Outra situação bastante comum durante a perfuração é a presença de seções contendo fluidos com diferentes densidades ao longo da coluna. Para calcular a pressão hidrostática nesse tipo de condição, é necessário determinar a variação de pressão separadamente para cada seção, conforme ilustrado na Figura 3.2.

Figura 3.2: Coluna composta por fluidos com diferentes características



Fonte Jr. et al. (1991)

Em geral a pressão em qualquer profundidade Z pode ser calculada por meio da equa-

ção:

$$p = p_0 + g \sum p_i (Z_i - Z_{i-1}) + g \rho_n (Z_i - Z_{i-1}) \quad (3.5)$$

Onde g é a gravidade [ft/s^2].

3.2.6 Densidade Equivalente

Em muitas situações de campo é útil comparar uma coluna com vários fluidos com uma coluna com um único fluido equivalente que esteja aberta para a atmosfera. Isso só é possível calculando a densidade da lama equivalente, definida por:

$$\rho_e = \frac{p}{0.05195Z} \quad (3.6)$$

A densidade da lama equivalente sempre deve ser calculada utilizando uma profundidade de referência específica Jr. *et al.* (1991).

3.2.7 Modelos Reológicos de Fluidos de Perfuração

Durante o processo de perfuração de um poço, é frequentemente necessário vencer forças viscoelásticas consideráveis para que o fluido de perfuração possa escoar através dos conduítes longos e estreitos utilizados nessa operação. Por isso, torna-se essencial a análise da perda de pressão por atrito.

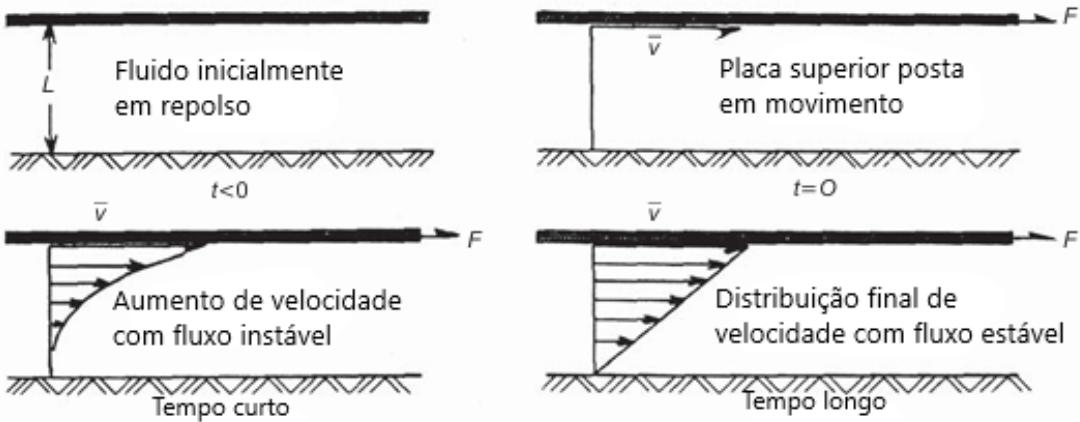
Na maioria dos casos, as propriedades elásticas dos fluidos de perfuração e seus efeitos durante o escoamento no poço são desprezíveis, sendo consideradas apenas as forças viscosas nos cálculos. Entretanto, com o avanço tecnológico, lamas cada vez mais complexas vêm sendo desenvolvidas, e, portanto, os testes devem considerar também as propriedades elásticas associadas à deformação do fluido durante o escoamento.

Para isso, é necessário descrever matematicamente e desenvolver equações que representem as perdas por atrito. Nesse contexto, engenheiros de perfuração costumam utilizar modelos reológicos para representar o comportamento dos fluidos. Neste trabalho, são abordados três modelos principais: o modelo Newtoniano, o modelo plástico de Bingham e o modelo da lei das potências Mitchell & Miska (2011). Ressalta-se ainda que outros modelos poderão ser incorporados em versões futuras do simulador, visando seu aprimoramento contínuo.

Visão geral dos modelos reológicos

As forças viscosas de um fluido são governadas pela viscosidade do mesmo, para entender o que é a viscosidade podemos analisar um simples experimento em que um fluido é colocado entre duas placas paralelas de área A separadas por uma distância L como mostra a Figura 3.3.

Figura 3.3: Fluxo laminar de fluido Newtoniano



Adaptado de Mitchell & Miska (2011)

Ao colocar a placa superior inicialmente em repouso em um movimento na direção x [ft] com uma velocidade constante v [ft/s] por um tempo suficiente, percebemos que uma força F [lbf] constante é necessária para manter a placa superior em movimento HALLIDAY & RESNICK (2009), a magnitude dessa força pode ser determinada por:

$$\frac{F}{A} = \mu \frac{\nu}{L} \quad (3.7)$$

A razão $\frac{F}{A}$ é conhecida como tensão de cisalhamento exercida sobre o fluido τ [psi]. A constante de proporcionalidade μ é chamada de viscosidade aparente [cP]. Dessa forma podemos definir a tensão de cisalhamento como:

$$\tau = \frac{F}{A} \quad (3.8)$$

A taxa de cisalhamento $\dot{\gamma}$ [$1/s$] é expressa como o gradiente da velocidade $\frac{v}{L}$:

$$\dot{\gamma} = \frac{d\nu}{dL} \approx \frac{\nu}{L} \quad (3.9)$$

A viscosidade aparente pode ser definida como a razão entre a tensão de cisalhamento e a taxa de cisalhamento. A principal característica de um fluido Newtoniano é a viscosidade constante do fluido. Como sabemos os fluidos de perfuração são misturas complexas que não podem ser caracterizadas por um único valor de viscosidade, quando um fluido não apresenta uma proporcionalidade entre tensão de cisalhamento e taxa de cisalhamento ele passa a ser conhecido como um fluido não Newtoniano, podendo ser pseudoplásticos se a viscosidade diminui com o aumento da taxa de cisalhamento e dilatantes se a viscosidade aumenta com o aumento da taxa de cisalhamento Mitchell & Miska (2011).

Modelo de fluido Newtoniano

Como já afirmamos um fluido Newtoniano tem a taxa de cisalhamento proporcional a tensão de cisalhamento:

$$\tau = \mu \dot{\gamma} \quad (3.10)$$

Onde a constante de proporcionalidade μ é o que chamamos de viscosidade. Para o caso de um fluido Newtoniano é retomando nosso experimento das placas, isso significa que se a força F for dobrada a velocidade da placa também será dobrada. Os principais fluidos Newtonianos são água, gás e salmouras, fluidos muito comuns na engenharia de poço.

A relação linear descrita pela Equação (3.10) só é válida para o fluxo laminar, quando o fluido se move em camadas, que ocorre apenas em taxas de cisalhamento baixas. Em altas taxas de cisalhamento o fluxo deixa de ser laminar e se torna turbulento, no qual as partículas se movem de forma caótica em relação ao sentido do fluxo criando vórtices e redemoinhos.

Modelo de fluidos plásticos de Bingham

O modelo plástico de Bingham Mitchell & Miska (2011) pode ser definido como:

$$\tau = \tau_y + \mu_p \dot{\gamma} \quad (3.11)$$

A principal característica de um plástico Bingham é a necessidade de um valor mínimo de tensão de cisalhamento para que o fluido comece a fluir, essa tensão mínima τ_y é chamada de tensão de escoamento [$lbf/100.sq.ft$]. Após a tensão de escoamento o fluido de Bingham se comporta como um fluido Newtoniano onde a mudança na tensão de cisalhamento é proporcional a mudança na taxa de cisalhamento. A constante de proporcionalidade μ_p é chamada de viscosidade plástica [cP].

Modelo fluidos de lei de potência

O modelo de lei de potência Mitchell & Miska (2011) pode ser definido como:

$$\tau = K \dot{\gamma}^n \quad (3.12)$$

O modelo de lei de potências requer também dois parâmetros para caracterização de fluidos, porém, esse modelo pode ser utilizado para representar um fluido pseudoplástico ($n < 1$), um fluido Newtoniano ($n = 1$) ou um fluido dilatante ($n > 1$).

O parâmetro K é chamado de índice de consistência do fluido [cP], e o parâmetro n é chamado de expoente da lei de potência ou índice de comportamento do fluxo.

3.2.8 Perda de Pressão Friccional em um Tubo de Perfuração

A perda de pressão friccional durante a circulação de fluidos em operações de perfuração pode ser calculada através de diferentes modelos de fluido. O primeiro passo é determinar o tipo de escoamento, para isso utilizamos o número de Reynolds N_{re} , porém, para cada modelo existe uma equação para a obtenção do número de Reynolds Jr. *et al.* (1991).

Modelo de fluido Newtoniano

Para um fluido Newtoniano o número de Reynolds pode ser obtido a partir da seguinte equação:

$$N_{re} = \frac{928\rho\bar{v}d}{\mu} \quad (3.13)$$

Onde d é o diâmetro interno do revestimento ID [in] e \bar{v} é a velocidade média [ft/s] que pode ser obtida pela seguinte equação:

$$\bar{v} = \frac{q}{2.448d^2} \quad (3.14)$$

Onde q é a vazão do poço [gal/min].

O fluxo é considerado turbulento quando o número de Reynolds é maior ou igual a 2100. Após determinar o regime de fluxo podemos utilizar uma das duas equações abaixo para calcular a perda de pressão por fricção em um poço $\frac{dp_f}{dL}$ [psi/ft].

Para fluxo laminar:

$$\frac{dp_f}{dL} = \frac{\mu\bar{v}}{1500d} \quad (3.15)$$

Para o fluxo turbulento:

$$\frac{dp_f}{dL} = \frac{f\rho\bar{v}^2}{25.8d} \quad (3.16)$$

Onde f é chamado de fator de fricção e pode ser calculado utilizando o método numérico de Newton-Raphson.

Fluidos plásticos de Bingham

Para um fluido que se comporta como plástico de Bingham a velocidade média podem ser obtida pela Equação (3.14). O número de Reynolds pode ser obtido a partir da seguinte equação:

$$N_{re} = \frac{928\rho\bar{v}d}{\mu_p} \quad (3.17)$$

O fluxo é considerado turbulento quando o número de Reynolds é maior ou igual ao número de Reynolds crítico N_{rec} . O número de Reynolds crítico pode ser calculado pela seguinte fórmula:

$$N_{rec} = \frac{1 - \frac{4}{3} \left(\frac{\tau_y}{\tau_w} \right) + \frac{1}{3} \left(\frac{\tau_y}{\tau_w} \right)^4}{8 \left(\frac{\tau_y}{\tau_w} \right)} N_{He} \quad (3.18)$$

Onde τ_w é a tensão de cisalhamento na parede [$lb f/ft^2$], N_{He} é chamado de número de Hedstrom e pode ser calculado pela seguinte fórmula:

$$N_{He} = \frac{37100 \rho \tau_y d^2}{\mu_p^2} \quad (3.19)$$

Para calcular a perda de pressão por fricção utilizamos as seguintes equações:

Para fluxo laminar:

$$\frac{dp_f}{dL} = \frac{\mu_p \bar{v}}{1500 d^2} + \frac{\tau_y}{225 d} \quad (3.20)$$

Para o fluxo turbulento podemos usar a Equação (3.16).

Fluidos de lei de potência

Para um fluido que atende ao modelo de lei de potência o número de Reynolds pode ser obtido a partir da seguinte equação:

$$N_{re} = \frac{89100 \rho \bar{v}^{2-n}}{K} \left(\frac{0.0416 d}{3 + \frac{1}{n}} \right)^n \quad (3.21)$$

A velocidade média pode ser obtida pela Equação (3.14). O fluxo é considerado turbulento quando o número de Reynolds é maior ou igual a 2100. Para calcular a perda de pressão por fricção utilizamos as seguintes equações:

Para fluxo laminar:

$$\frac{dp_f}{dL} = \frac{K \bar{v}^n \left(\frac{3 + \frac{1}{n}}{0.0416} \right)^n}{144000 d^{1+n}} \quad (3.22)$$

Para o fluxo turbulento podemos usar a Equação (3.16).

3.2.9 Perda de Pressão Friccional em um Anular

A perda de pressão friccional durante a circulação de fluidos em operações de perfuração também pode ocorrer no anular, e assim como a perda na tubulação, pode ser calculada através de diferentes modelos de fluido. Assim como vimos anteriormente o primeiro passo é determinar o tipo de escoamento, para isso utilizamos o número de Reynolds, porém para cada modelo existe uma equação para a obtenção do número de Reynolds.

Modelo de fluido Newtoniano

Para um fluido Newtoniano o número de Reynolds pode ser obtido a partir da seguinte equação:

$$N_{re} = \frac{757\rho\bar{v}(d_2 - d_1)}{\mu} \quad (3.23)$$

Onde d_1 é o diâmetro externo do revestimento OD [in] e d_2 é o diâmetro do poço [in].

A velocidade média pode ser obtida pela equação:

$$\bar{v} = \frac{q}{2.448(d_2^2 - d_1^2)} \quad (3.24)$$

O fluxo é considerado turbulento quando o número de Reynolds é maior ou igual a 2100. Após determinar o regime de fluxo podemos utilizar uma das duas equações abaixo para calcular a perda de pressão por fricção em um anular.

Para fluxo laminar:

$$\frac{dp_f}{dL} = \frac{\mu\bar{v}}{1000(d_2 - d_1)^2} \quad (3.25)$$

Para o fluxo turbulento:

$$\frac{dp_f}{dL} = \frac{f\rho\bar{v}^2}{21.1(d_2 - d_1)} \quad (3.26)$$

Fluidos plásticos de Bingham

Para um fluido que se comporta como plástico de Bingham a velocidade média pode ser obtida pela Equações (3.24). O número de Reynolds pode ser obtido a partir da seguinte equação:

$$N_{re} = \frac{757\rho\bar{v}(d_2 - d_1)}{\mu_p} \quad (3.27)$$

O fluxo é considerado turbulento quando o número de Reynolds é maior ou igual ao número de Reynolds crítico. O número de Reynolds crítico pode ser calculado usando a Equação (3.18), mas o número de Hedstrom deve ser calculado pela seguinte equação:

$$N_{He} = \frac{24700\rho\tau_y(d_2 - d_1)^2}{\mu_p^2} \quad (3.28)$$

Para calcular a perda de pressão por fricção utilizamos as seguintes equações:

Para fluxo laminar:

$$\frac{dp_f}{dL} = \frac{\mu\bar{v}}{1000(d_2 - d_1)^2} + \frac{\tau_y}{200(d_2 - d_1)} \quad (3.29)$$

Para o fluxo turbulento podemos usar a Equação (3.26).

Fluidos de lei de potência

Para um fluido que atende ao modelo de lei de potência o número de Reynolds pode ser obtido a partir da seguinte equação:

$$N_{re} = \frac{109000\rho\bar{v}^{2-n}}{K} \left(\frac{0.0208(d_2 - d_1)}{2 + \frac{1}{n}} \right)^n \quad (3.30)$$

A velocidade média pode ser obtida pela Equação (3.24). O fluxo é considerado turbulento quando o número de Reynolds é maior ou igual a 2100. Para calcular a perda de pressão por fricção utilizamos as seguintes equações:

Para fluxo laminar:

$$\frac{dp_f}{dL} = \frac{K\bar{v}^n \left(\frac{2+\frac{1}{n}}{0.0208} \right)^n}{144000(d_2 - d_1)^{1+n}} \quad (3.31)$$

Para o fluxo turbulento podemos usar a Equação (3.26).

Grande parte das informações apresentadas neste capítulo foram extraídas de Mitchell & Miska (2011) e Jr. *et al.* (1991).

3.2.10 Variações de Carga Axial e Deslocamento em Colunas de Poço

Durante a operação de perfuração ou completação, as colunas de revestimento e produção estão sujeitas a variações de pressão e temperatura que afetam diretamente sua integridade estrutural. Essas variações resultam em alterações na carga axial e no deslocamento da coluna, o que pode comprometer sua função caso não sejam corretamente previstas e monitoradas.

Com base em Bourgoyne et al. (2011), apresenta-se nesta seção a descrição dos principais efeitos envolvidos na variação da carga e do deslocamento axial de colunas tubulares em poços de petróleo. Mitchell & Miska (2011)

Efeito da Variação de Temperatura

A variação de temperatura ao longo da coluna provoca expansão ou contração térmica. Essa deformação axial é diretamente proporcional ao comprimento da coluna, ao coeficiente de dilatação térmica do material e à variação de temperatura:

$$\Delta L_t = \alpha_T L \Delta T \quad (3.32)$$

Esse efeito é abordado por Bourgoyne et al. (2011), que destacam sua relevância especialmente em colunas com extremidades restritas, como no caso de instalação de packers, onde a expansão térmica gera tensões axiais significativas. Mitchell & Miska (2011)

Efeito Balão (Ballooning Effect)

Quando ocorre variação na pressão interna e externa da coluna, há expansão ou contração radial da parede tubular. Essa deformação radial acarreta uma reação axial, denominada efeito balão, especialmente significativa em colunas de paredes finas.

$$\Delta L_b = \frac{2v[\Delta P_{in}A_{in} - \Delta P_{out}A_{out}]}{E(A_{out} - A_{in})} \quad (3.33)$$

Segundo Bourgoyne et al. (2011), esse efeito é crucial em operações com injeção de fluido ou sob variações bruscas de pressão. Mitchell & Miska (2011)

Variação Axial Devido à Força de Pistão (ΔF)

Além de alterar diretamente a carga axial, a força de pistão pode provocar variação de comprimento na coluna de forma semelhante ao efeito balão, especialmente quando o fluido pressiona diferencialmente regiões com diferentes seções transversais.

A equação que representa esse deslocamento axial é:

$$\Delta F = \Delta P_i A_i + \Delta P_{out} A_{out} \quad (3.34)$$

Esse termo representa o efeito pistão global sobre a coluna, sendo fundamental para simular condições com packer ativado, crossover, ou colunas parcialmente cimentadas.

Em aplicações práticas, a força de pistão pode causar deslocamentos significativos, inclusive contribuindo para falhas por flambagem, se não houver alívio de carga ou previsão de expansão térmica.

Força de Pistão Gerada por Packer

A presença de um packer impede o deslocamento axial livre da coluna. Assim, quando há diferença de pressão entre o interior e o exterior da coluna, gera-se uma força de pistão sobre a seção da coluna confinada, dada por:

$$\Delta L_{packer} = \frac{(\Delta F)L}{EA_s} \quad (3.35)$$

Bourgoyne et al. (2011) ressaltam que esse efeito pode causar variações expressivas no carregamento axial, afetando diretamente o desempenho da completação. Mitchell & Miska (2011)

Força de Pistão em Crossovers

Em transições entre tubulações com diferentes geometrias (conhecidas como crossovers), o diferencial de área provoca uma força de pistão adicional, resultando em alongamento ou encurtamento da coluna:

$$\Delta L_{crossover} = \frac{(\Delta F)L}{EA_s} \quad (3.36)$$

Esse comportamento deve ser contemplado em modelos estruturais de colunas com acessórios ou elementos de transição, conforme apontado por Bourgoyn et al. (2011). Mitchell & Miska (2011)

Efeitos de Injeção: Coluna Livre vs. Coluna Fixa

A maneira como a coluna está confinada impacta significativamente a reação à injeção de fluidos:

- **Coluna livre:** permite deslocamento axial, dissipando parte da carga;
- **Coluna fixa:** restringe deslocamento e transforma toda a variação em aumento de carga axial.

Segundo Bourgoyn et al. (2011, p. 407), as restrições impostas nas extremidades da coluna modificam significativamente a resposta estrutural da tubulação, influenciando tanto a distribuição das cargas quanto os deslocamentos axiais. Mitchell & Miska (2011)

3.3 Identificação de Pacotes – Assuntos

- Pacote engenharia de poço:
 - O pacote engenharia de poço é responsável por relacionar os pacotes mecânicas dos fluidos, mecânica das rochas e equações analíticas de forma a tornar possível e coerente os resultados obtidos pela simulação.
- Pacote mecânica dos fluidos:
 - É o pacote que relaciona todas as propriedades dos fluidos e como esses fluidos se correlacionam com o poço e com outros fluidos.
- Pacote mecânica das rochas:
 - É o pacote que relaciona todas as propriedades das rochas presentes no sistema.
- Pacote janela principal:
 - É o pacote que comprehende a interface amigável que o usuário terá contato, é o ambiente onde o usuário poderá enviar comandos para o simulador e é a partir daqui que poderá visualizar os resultados.
- Pacote equações analíticas:

- Neste pacote estão agrupadas todas as equações analíticas que são aplicadas durante a simulação
- Pacote modelagem gráfica:
 - Esse é o pacote responsável por montar os gráficos que são obtidos a partir dos resultados da simulação.

3.4 Diagrama de Pacotes – Assuntos

O diagrama de pacotes é apresentado na Figura 3.4.

Figura 3.4: Diagrama de pacotes

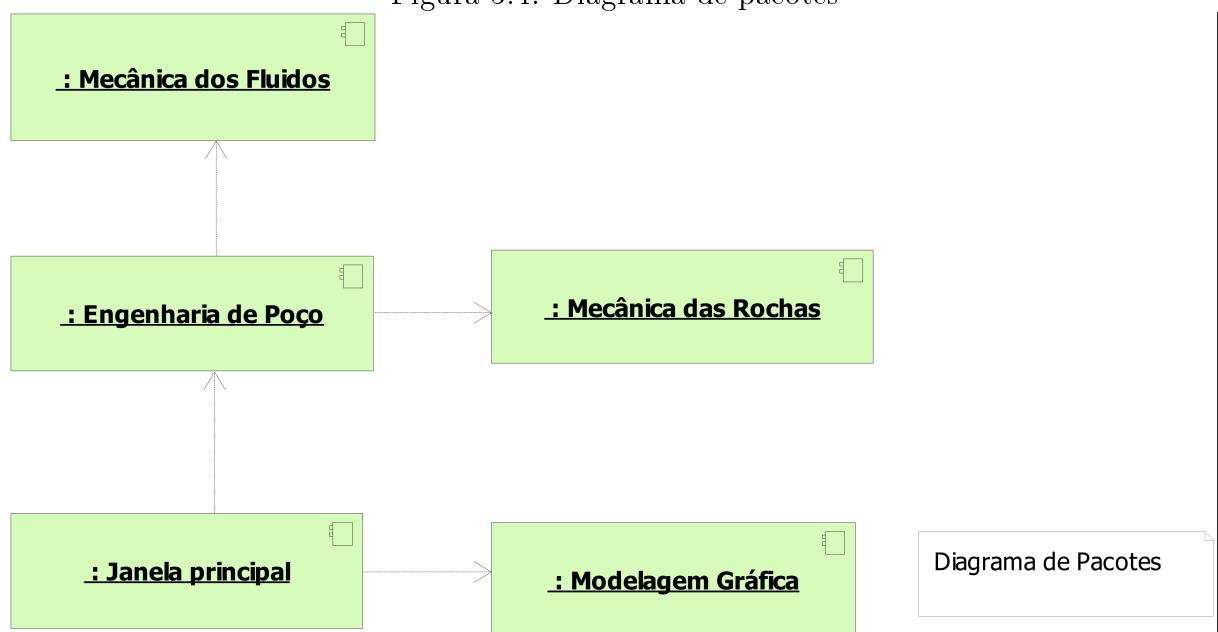


Diagrama de Pacotes

3 - Análise Orientada a Objeto

Capítulo 4

AOO – Análise Orientada a Objeto

Neste capítulo apresentam-se as classes desenvolvidas no projeto, suas respectivas relações, atributos e métodos. Apresenta-se também um breve conceito de cada classe. Todos os diagramas foram elaborados seguindo a estrutura da UML (Linguagem de Modelagem Unificada), com o objetivo de padronizar e facilitar a compreensão do sistema. Além do diagrama de classes, são incluídos os diagramas de sequência, de comunicação, de máquina de estado e de atividades BUENO (2003).

4.1 Diagramas de classes

O diagrama de classes é apresentado na Figura 4.1. Seu objetivo é representar graficamente a estrutura estática do sistema, evidenciando todas as classes envolvidas, seus respectivos atributos, métodos, relações de herança e associações entre as classes.

Essa representação segue a notação da UML (Linguagem de Modelagem Unificada) e serve como base para o entendimento da arquitetura do software, permitindo uma visão clara da organização interna dos componentes e de suas interdependências.

O diagrama de classes é apresentado na Figura . Ele tem como objetivo apresentar todas as classes, seus atributos, métodos, heranças e relações entre as classes.

Figura 4.1: Diagrama de classes



Fonte: Produzido pelo autor

4.1.1 Dicionário de Classes

O software foi desenvolvido com base em uma arquitetura modular orientada a objetos, utilizando linguagem C++ e as bibliotecas Qt e QCustomPlot. A estrutura é composta por múltiplas classes organizadas conforme suas responsabilidades funcionais e hierarquia de dependência.

A seguir, apresentam-se as principais classes e suas funcionalidades:

- **CFluido:** armazena os atributos físicos do fluido e realiza todos os cálculos relacionados às suas propriedades.
- **CObjetoPoco:** responsável por armazenar as propriedades gerais do poço e integrar os diferentes trechos que o compõem.
- **CTrechoTubulacao:** representa cada seção tubular do poço, permitindo uma modelagem segmentada. Os objetos dessa classe contêm fluido e estão organizados dentro da estrutura de CObjetoPoco.
- **CModeloReologico** (classe base) e suas derivadas:
 - **CModeloNewtoniano**
 - **CModeloBingham**
 - **CModeloPotencia**
 - * Essas classes implementam os cálculos de perda de pressão friccional com base nos respectivos modelos reológicos, sendo aplicadas conforme o comportamento do fluido analisado.
- **CSimuladorReologico:** classe principal do simulador. Esta janela integra os modelos reológicos e executa os cálculos associados às propriedades dos fluidos e trechos do poço.
- **CSimuladorPerdaTubulacao:** segunda janela do sistema, voltada para a análise de perda de pressão por fricção ao longo dos trechos, incluindo variações de comprimento (ΔL) e outros fatores associados ao escoamento.
- **CJanelaAdicionarFluido, CJanelaAdicionarTrechoTubulacao, CJanelaGráficoPressaoHidrostatica, CJanelaMenu:** classes auxiliares criadas com o Qt Creator, responsáveis por fornecer interfaces gráficas para entrada e visualização de dados. Cada janela é especializada em uma função específica, como adição de trechos, inserção de fluido ou exibição de gráficos.
- **QCustomPlot:** biblioteca externa utilizada para renderização de gráficos científicos, como perfis de pressão e densidade.

O diagrama de classes, apresentado na Figura 4.1, resume as relações entre as principais entidades do sistema, seus atributos, métodos e heranças, seguindo a notação da Linguagem de Modelagem Unificada (UML)

4.2 Diagrama de Sequência – Eventos e Mensagens

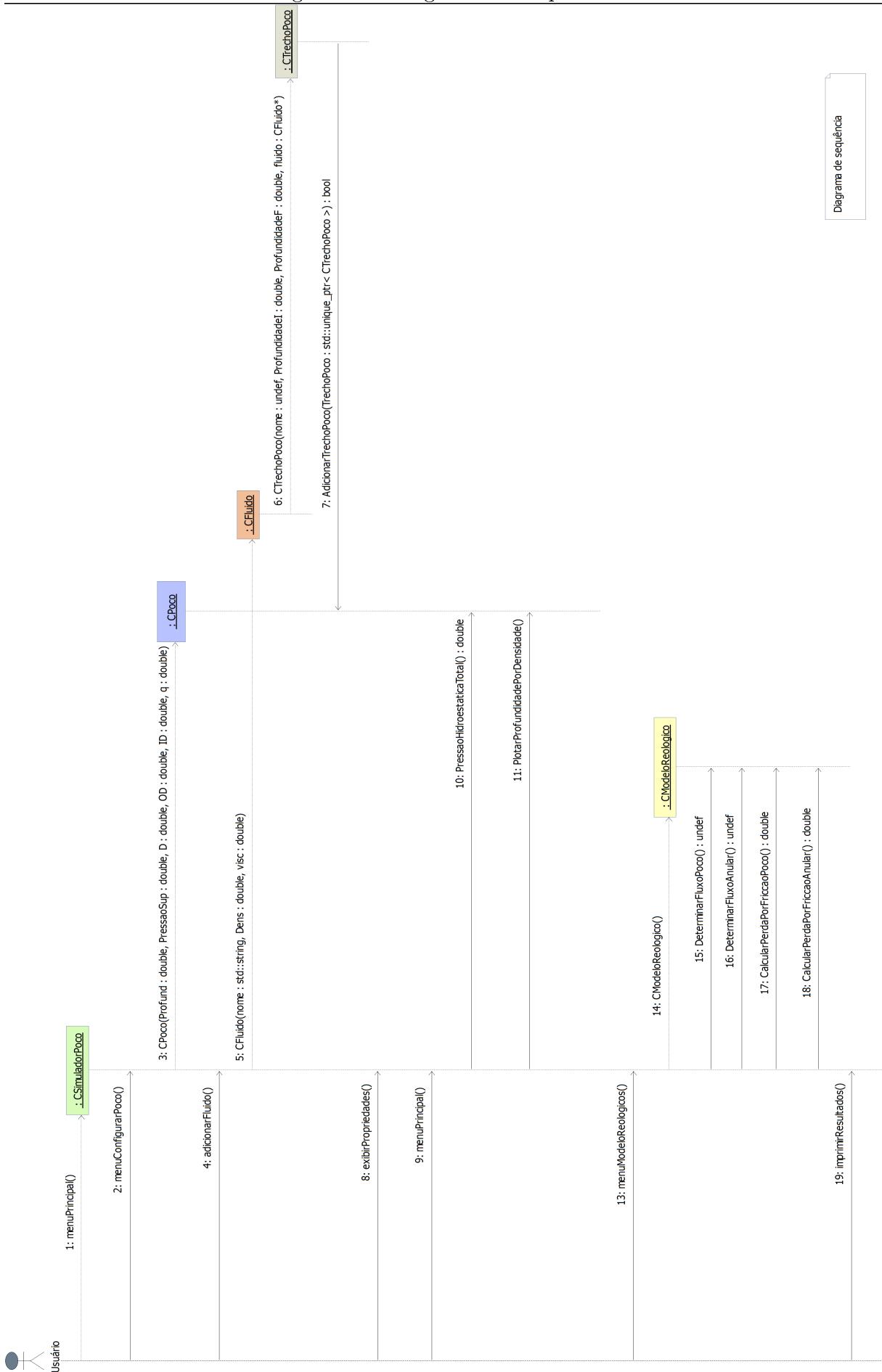
O diagrama de sequência descreve a interação entre os objetos do sistema e os elementos externos, evidenciando a ordem temporal das mensagens trocadas durante a execução de uma funcionalidade. Ele apresenta o fluxo de controle do sistema de forma cronológica, detalhando as chamadas de métodos e as respostas entre os elementos envolvidos.

Sua construção geralmente tem como base os cenários definidos nos diagramas de casos de uso. A partir disso, é possível representar a comunicação entre os participantes e os objetos do sistema, permitindo uma visualização clara da lógica de execução dos processos.

4.2.1 Diagrama de Sequência Geral

A seguir, é apresentado o diagrama de sequência geral na Figura 4.2.

Figura 4.2: Diagrama de sequência

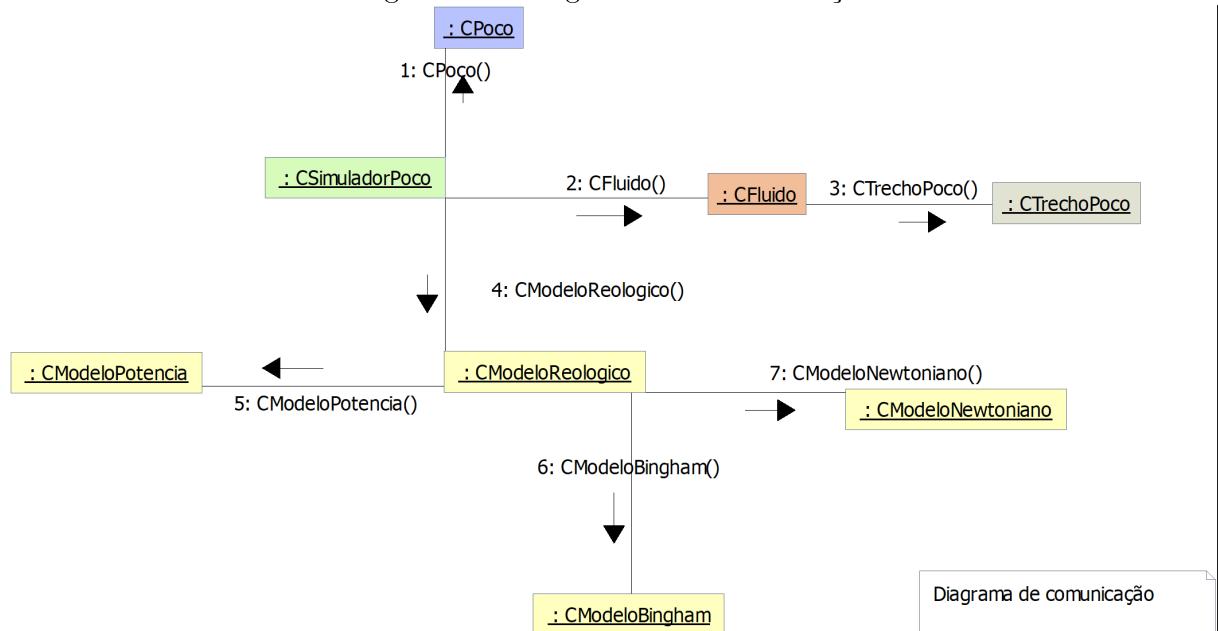


Fonte: Produzido pelo autor.

4.3 Diagrama de Comunicação – Colaboração

O diagrama de comunicação representa as interações entre os objetos do sistema em um determinado contexto, evidenciando a troca de mensagens e a sequência dos processos envolvidos. A disposição dos elementos enfatiza os relacionamentos estruturais, ao mesmo tempo em que indica a ordem numérica das mensagens trocadas durante a execução de uma tarefa específica.

Figura 4.3: Diagrama de comunicação



Fonte: Produzido pelo autor.

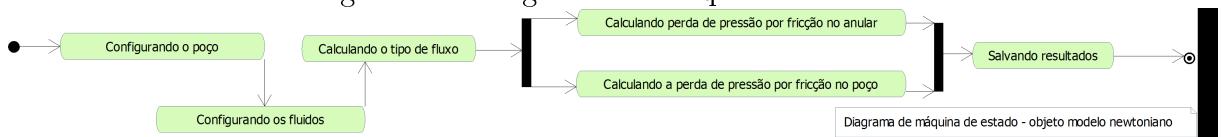
4.4 Diagrama de Máquina de Estado

O diagrama de máquina de estados descreve os diferentes estados que um objeto pode assumir ao longo de seu ciclo de vida, bem como os eventos que provocam mudanças entre esses estados. A Figura 4.4apresenta esse diagrama aplicado ao objeto relacionado ao modelo reológico newtoniano.

O processo tem início com o recebimento dos dados pela classe responsável pela simulação. A partir disso, os atributos necessários são criados e o sistema passa para a fase de definição do poço. Dependendo da configuração estabelecida, a simulação pode ser direcionada para uma única seção ou para múltiplas seções.

Na sequência, é realizada a configuração do fluido presente no poço, que pode ser do tipo gás ou óleo. Com todas as definições realizadas, os cálculos da simulação são executados a fim de determinar os parâmetros operacionais. Os resultados obtidos são, então, processados e exibidos graficamente para análise. Ao final da execução, o processo é encerrado de forma automática.

Figura 4.4: Diagrama de máquina de estado



Fonte: Produzido pelo autor.

4.5 Diagrama de Atividades

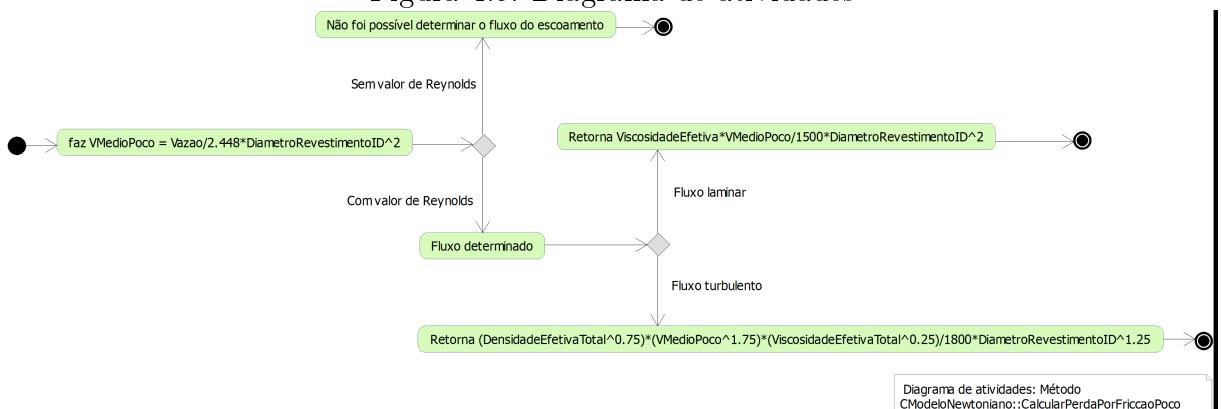
O diagrama de atividades apresentado descreve, em detalhe, a execução de uma atividade específica do sistema. No caso em questão, é representado o método **CalcularPerdaPorFriccaoPoco**, pertencente à classe **CModeloNewtoniano**.

O processo se inicia com o recebimento dos dados pela classe responsável pela simulação dos fluidos. Os atributos do objeto são atualizados conforme os valores de entrada fornecidos. O primeiro passo do método consiste no cálculo da velocidade média do fluido no poço. Em seguida, é realizada uma verificação para identificar se o número de Reynolds foi previamente determinado. Caso essa informação esteja ausente, o sistema emite uma mensagem de erro. Caso contrário, o método prossegue para a classificação do tipo de escoamento.

A partir do valor do número de Reynolds, o escoamento pode ser classificado como laminar ou turbulento, e o cálculo é ajustado conforme o regime identificado. Os procedimentos subsequentes utilizam as propriedades físicas específicas do fluido em questão para concluir os cálculos de perda de carga por fricção.

Ao final, os resultados são processados e retornados para o sistema, encerrando a execução do método.

Figura 4.5: Diagrama de atividades



Fonte: Produzido pelo autor.

Capítulo 5

Projeto

Neste capítulo, são apresentados os principais aspectos relacionados à implementação do projeto, incluindo a descrição do ambiente de desenvolvimento, as bibliotecas gráficas utilizadas e a evolução das versões do sistema ao longo do processo. Também são incluídos os diagramas de componentes e de implantação, que auxiliam na visualização da estrutura física e lógica da aplicação.

5.1 Projeto do sistema

O projeto foi desenvolvido com base no paradigma da programação orientada a objetos, o qual possibilita maior modularidade, reutilização de código e organização lógica das funcionalidades.

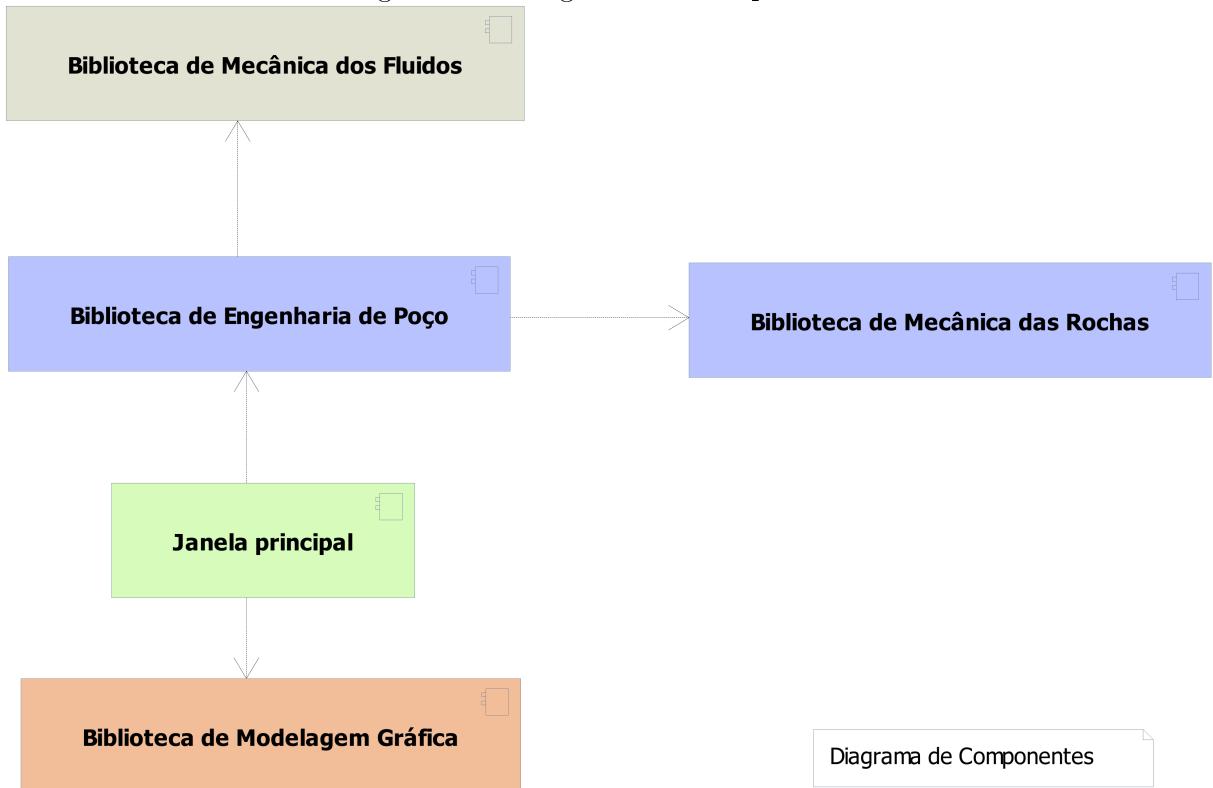
A linguagem escolhida foi o C++, em virtude de suas características que a tornam especialmente adequada para o desenvolvimento de aplicações técnicas e científicas. Os principais fatores que motivaram essa escolha incluem:

- Capacidade de alto desempenho, adequada à realização de cálculos numéricos intensivos;
- Suporte robusto ao paradigma orientado a objetos, com ampla compatibilidade com ferramentas baseadas em UML;
- Disponibilidade de bibliotecas consolidadas para gráficos (como a Gnuplot) e geração de arquivos de saída no formato .dat;
- Permite diferentes níveis de abstração, viabilizando tanto programação de baixo nível quanto de alto nível;
- Compatibilidade com diversos ambientes de desenvolvimento (*IDEs*), compiladores, depuradores e analisadores de desempenho (*profilers*);
- Acesso gratuito a compiladores e ferramentas, o que facilita a adoção da linguagem por estudantes e instituições de ensino.

5.2 Diagrama de componentes

O diagrama de componentes tem como objetivo representar a organização modular do sistema, evidenciando as dependências e relações entre os principais componentes de software. Esse diagrama é apresentado na Figura 5.1, onde é possível visualizar a estrutura lógica da aplicação e como seus módulos interagem entre si.

Figura 5.1: Diagrama de componentes



Fonte: Produzido pelo autor.

Na Figura 5.1, temos o simulador, que se comunica com a biblioteca de plotagem de gráficos e com a biblioteca de funções matemáticas. A biblioteca de estatística se comunica com a biblioteca de funções matemáticas.

Capítulo 6

Ciclos de planejamento/detalhamento

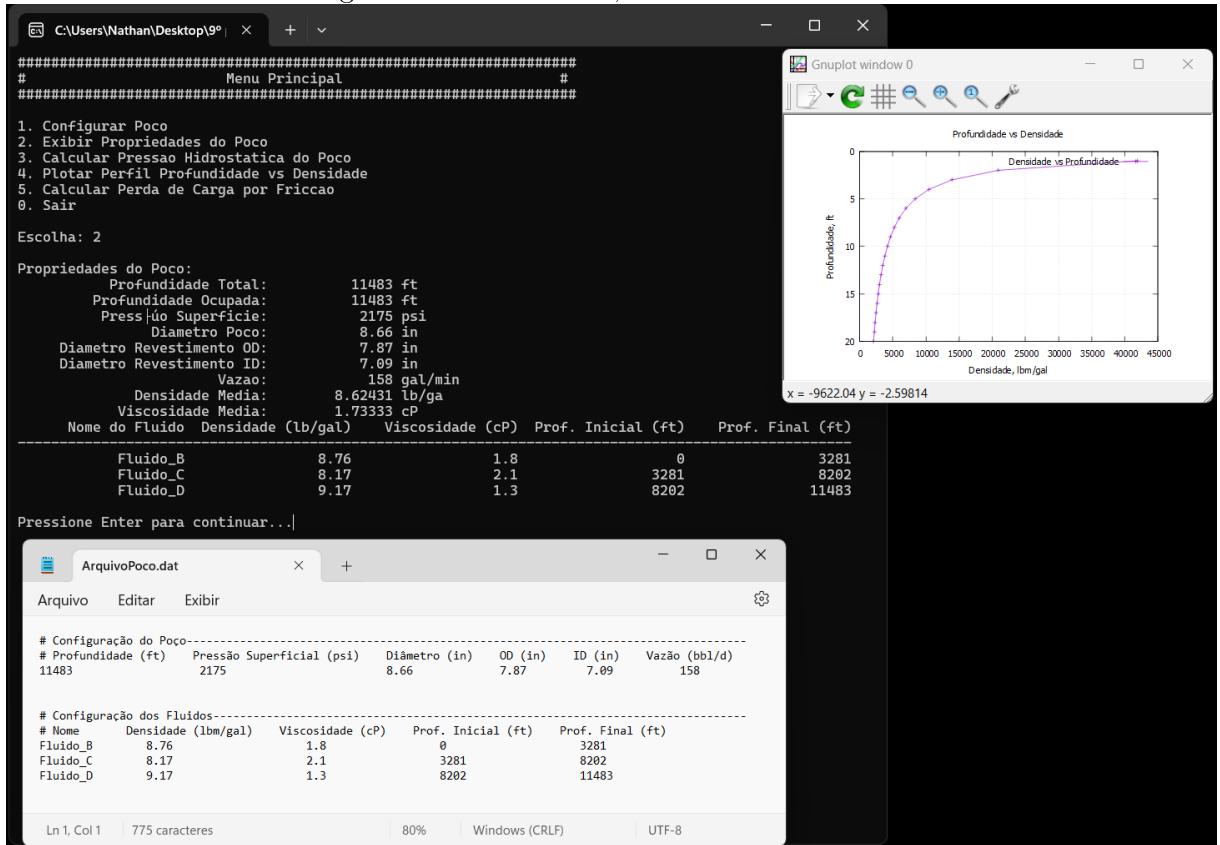
Apresenta-se neste capítulo as versões do software desenvolvido.

6.1 Versão Inicial – Interação via Terminal e Geração de Gráficos com Gnuplot

Na versão inicial do sistema, a entrada e saída de dados foi implementada exclusivamente por meio do terminal, sem o uso de bibliotecas gráficas. Para a visualização dos resultados, foi incorporado o uso do Gnuplot, permitindo a geração de gráficos de forma simples e direta, o que facilitou a apresentação dos dados ao usuário.

Essa versão foi desenvolvida em um ambiente de linha de comando, operando no sistema Windows 11. Trata-se de uma versão protótipo do software, em que a interação ocorre essencialmente por meio de texto. Mesmo com essa limitação, o usuário já era capaz de gerar gráficos e realizar simulações em diferentes cenários de forma funcional.

Figura 6.1: Versão 0.1, interface do software



Fonte: Produzido pelo autor.

6.2 Versão 0.2 – Otimização de Entrada, Validação e Armazenamento Automático de Dados

A versão 0.2 do programa introduziu melhorias significativas em termos de usabilidade, efetividade e gestão de dados, mantendo a interação orientada ao terminal e a visualização de resultados por meio do Gnuplot. Essa atualização teve como foco a correção de falhas identificadas na versão anterior e a inclusão de novas funcionalidades que aprimoraram a experiência do usuário, tornando-a mais fluida e intuitiva.

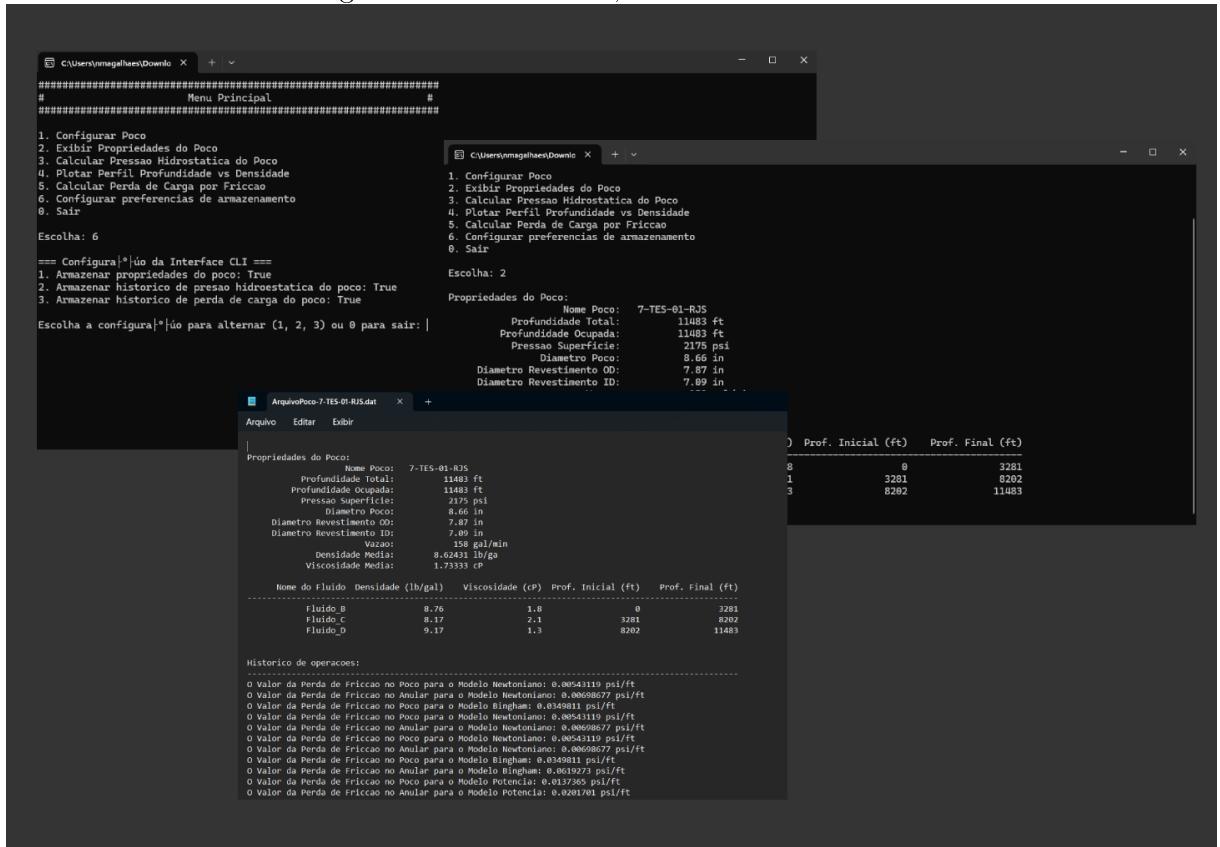
Principais aprimoramentos implementados:

- Reformulação na forma de apresentação dos dados, garantindo maior clareza e precisão na exibição dos resultados.
- Otimização da navegação no menu de opções, permitindo a seleção de comandos por meio da digitação direta de números, sem a necessidade de pressionar Enter repetidamente, o que tornou o processo mais ágil.
- Implementação de um sistema de salvamento automático, no qual os dados são armazenados em arquivos nomeados conforme o poço em questão, contendo o histórico completo das ações realizadas durante a simulação.

- Adição de um mecanismo de verificação de entradas, capaz de detectar valores inválidos ou formatos inadequados, reduzindo significativamente erros de execução e assegurando a continuidade do processo de forma estável.

Após a conclusão dos cálculos, o usuário pode acessar o histórico completo dos dados gerados durante a simulação. Esse recurso contribui para uma análise mais detalhada dos resultados, permitindo maior controle sobre as etapas executadas e facilitando a rastreabilidade das informações. A Figura 6.2 exemplifica algumas dessas melhorias implementadas na versão 0.2, evidenciando a evolução da interface textual e das funcionalidades associadas à gestão dos dados.

Figura 6.2: Versão 0.2, interface do software



Fonte: Produzido pelo autor.

6.3 Versão 1.0 – Interface Gráfica, Amigável e Intuitiva Utilizando o Framework Qt Creator

A versão 1.0 do software marcou uma transição significativa em sua arquitetura e usabilidade, ao substituir a interface baseada exclusivamente em comandos de terminal por uma interface gráfica interativa, desenvolvida com foco na acessibilidade e na experiência do usuário. Essa atualização manteve todas as funcionalidades implementadas nas versões anteriores, porém incorporou novos recursos visuais que tornaram a navegação mais

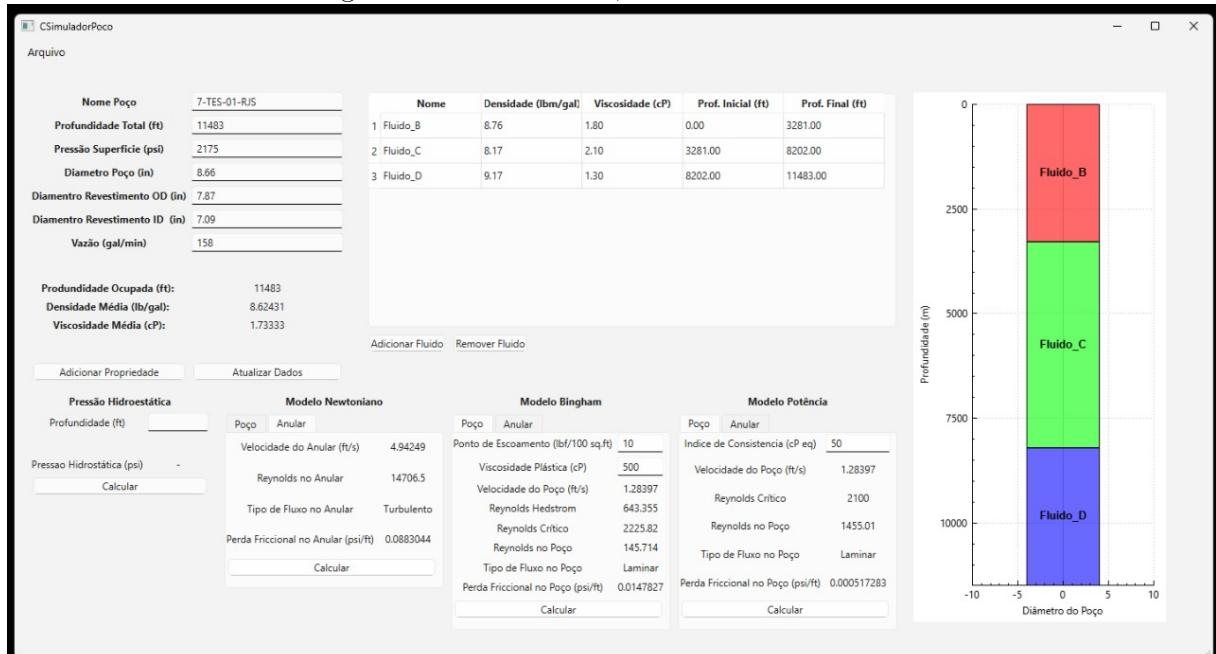
intuitiva.

Com a introdução da interface gráfica, o usuário passou a contar com as seguintes facilidades:

- Inserção de dados por meio de campos de texto e botões, dispensando a digitação direta no terminal;
- Visualização das propriedades dos fluidos do poço organizadas em tabelas, o que proporciona maior clareza e organização das informações;
- Interação com gráficos que representam o poço e os fluidos ao longo da profundidade, permitindo uma análise visual mais intuitiva da configuração do sistema.

Essa versão consolida a transição do sistema, antes concebido como um protótipo textual, para uma aplicação interativa com maior usabilidade. A nova abordagem contribui diretamente para o aprendizado dos usuários e facilita a análise de diferentes cenários de simulação relacionados à Engenharia de Poço.

Figura 6.3: Versão 1.0, interface do software



Fonte: Produzido pelo autor.

6.4 Versão 1.1 – Consolidação Visual, Barra de Tarefas e Automação de Processos

A versão 1.1 do software manteve todas as funcionalidades introduzidas na versão 1.0, porém trouxe importantes avanços no aspecto visual e na eficiência da interface. Houve uma consolidação da estrutura gráfica, com ajustes que tornaram o ambiente mais limpo, responsivo e funcional para o usuário.

Uma das principais melhorias foi a automação do processo de cálculo: o software passou a detectar alterações nas propriedades dos elementos simulados e a recalcular os parâmetros automaticamente, sem a necessidade de o usuário acionar repetidamente o botão "Calcular". Essa otimização reduziu interações redundantes e tornou o fluxo de trabalho mais ágil.

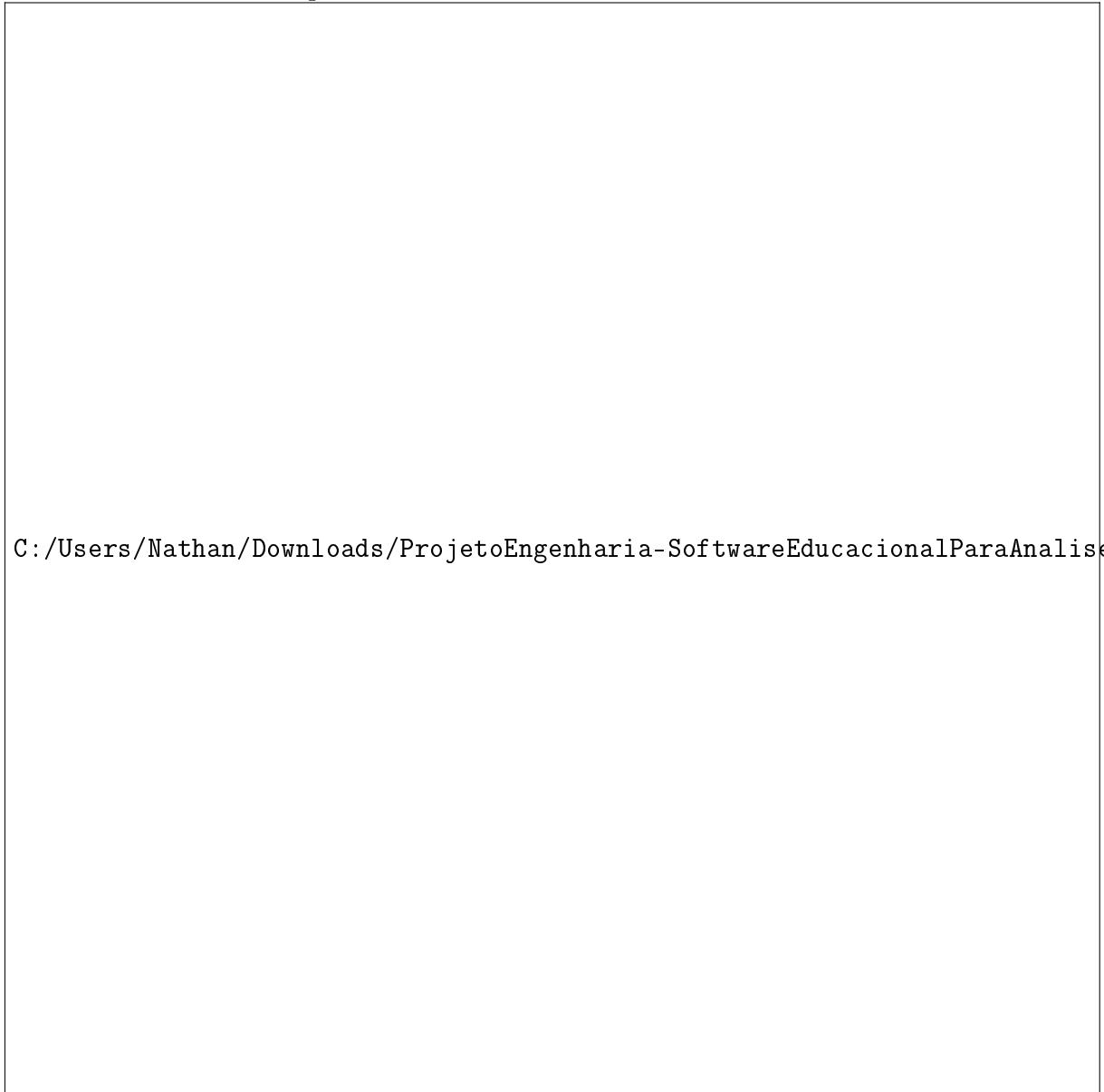
Além disso, foi implementada uma barra de tarefas com novas funcionalidades organizadas em menus acessíveis, incluindo:

- **Arquivo:** opções para iniciar nova simulação, salvar o projeto atual, importar dados e exportar a interface como imagem;
- **Referências:** acesso ao manual do usuário e à documentação dos modelos reológicos implementados;
- **Atalhos de teclado:** comandos como Ctrl+N para nova simulação e Ctrl+S para salvar, otimizando a navegação do sistema.

Outro recurso adicionado foi a possibilidade de exibir o gráfico da pressão hidrostática ao longo da profundidade do poço, oferecendo uma visualização detalhada do comportamento do fluido em função da profundidade. Essa nova ferramenta complementa os gráficos já existentes, enriquecendo a análise dos resultados simulados.

Com essas melhorias, a versão 1.1 reforça a transição do software de uma ferramenta educacional básica para uma aplicação mais robusta, interativa e alinhada às necessidades dos usuários no contexto da Engenharia de Poço.

Figura 6.4: Versão 1.1, interface do software



Fonte: Produzido pelo autor.

6.5 Versão 2.0 – Navegação por Módulos e Simulação Mecânica de Variação de Comprimento (ΔL)

A versão 2.0 do software introduziu uma das mudanças mais significativas desde o início do projeto, ao implementar um sistema de navegação baseado em módulos. Logo ao iniciar o programa, o usuário se depara com um menu principal contendo dois botões de acesso: Módulo 1 e Módulo 2, além de informações institucionais como nome do software, desenvolvedor, coordenador e contatos.

O Módulo 1 direciona para o simulador hidráulico de perfuração, que engloba todas as funcionalidades desenvolvidas até a versão 1.1, incluindo a simulação de escoamento,

cálculo de pressão hidrostática e perdas por fricção com base em modelos reológicos. Esse módulo mantém a interface gráfica interativa e os recursos de visualização consolidados nas versões anteriores.

O Módulo 2, por sua vez, representa a nova funcionalidade da versão 2.0: o módulo de análise de tensões em colunas. Essa segunda interface tem como foco principal o estudo de variações de comprimento (ΔL) de colunas de completação, levando em consideração efeitos como:

- Variações de temperatura (dilatação térmica);
- Efeito balão (ballooning);
- Força pistão gerada por packer ou crossover;
- Força restauradora;
- Pressões aplicadas ao longo da coluna.

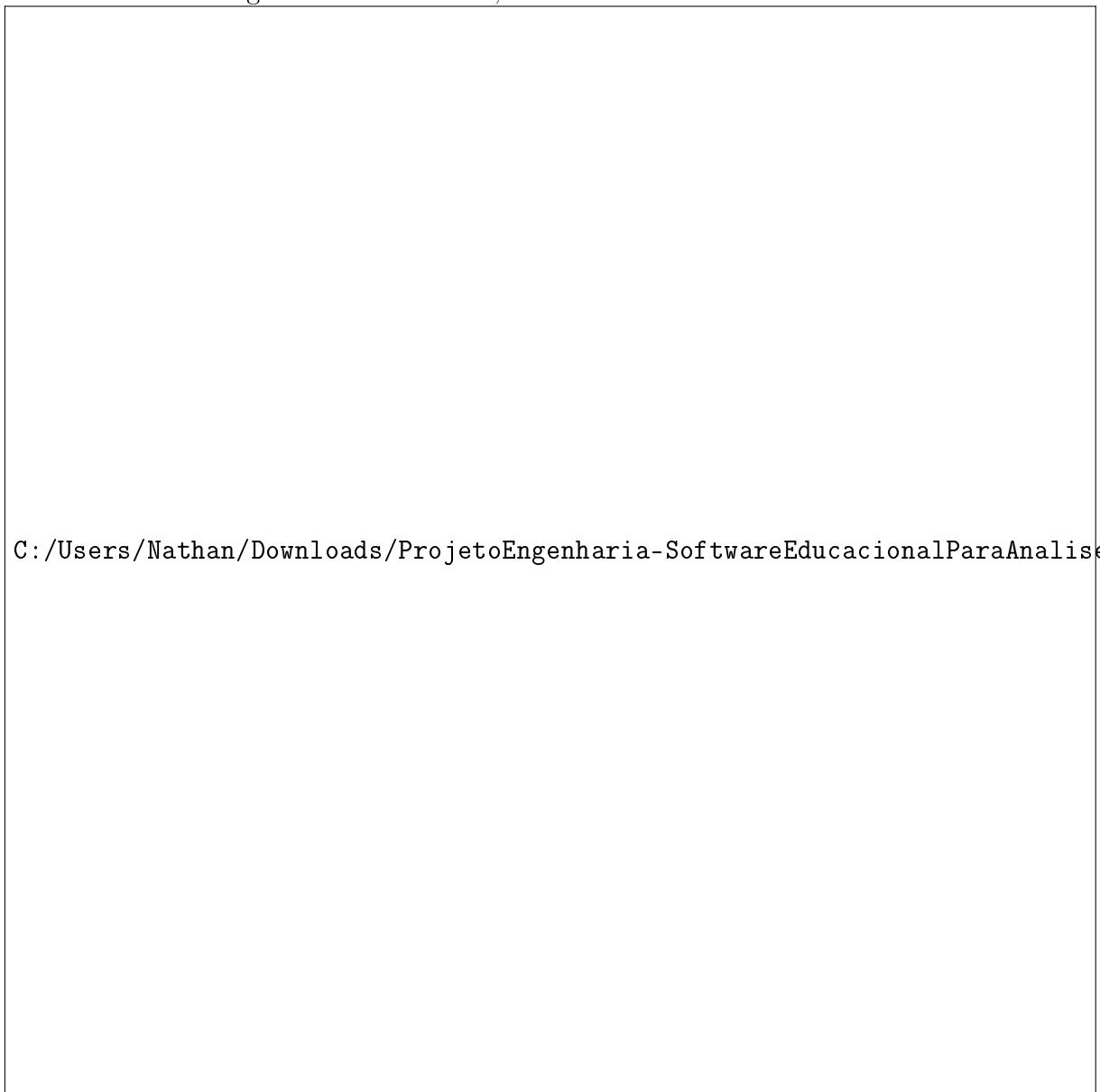
Para viabilizar essas análises, o usuário pode inserir propriedades adicionais, como coeficiente de expansão térmica, coeficiente de Poisson e módulo de elasticidade do material da coluna. A interface também permite modelar o poço com múltiplas seções, o que possibilita simulações mais precisas e segmentadas, inclusive em cenários com ou sem a presença de packer.

Além disso, o poço continua sendo visualizado graficamente conforme a profundidade, agora com suporte ao novo conjunto de propriedades exigidas pela análise mecânica.

O software mantém todos os recursos consolidados nas versões anteriores, incluindo a barra de menu com funções de nova simulação, salvamento, exportação de gráficos como imagem, além do acesso ao manual do usuário e às fórmulas utilizadas nos cálculos do sistema.

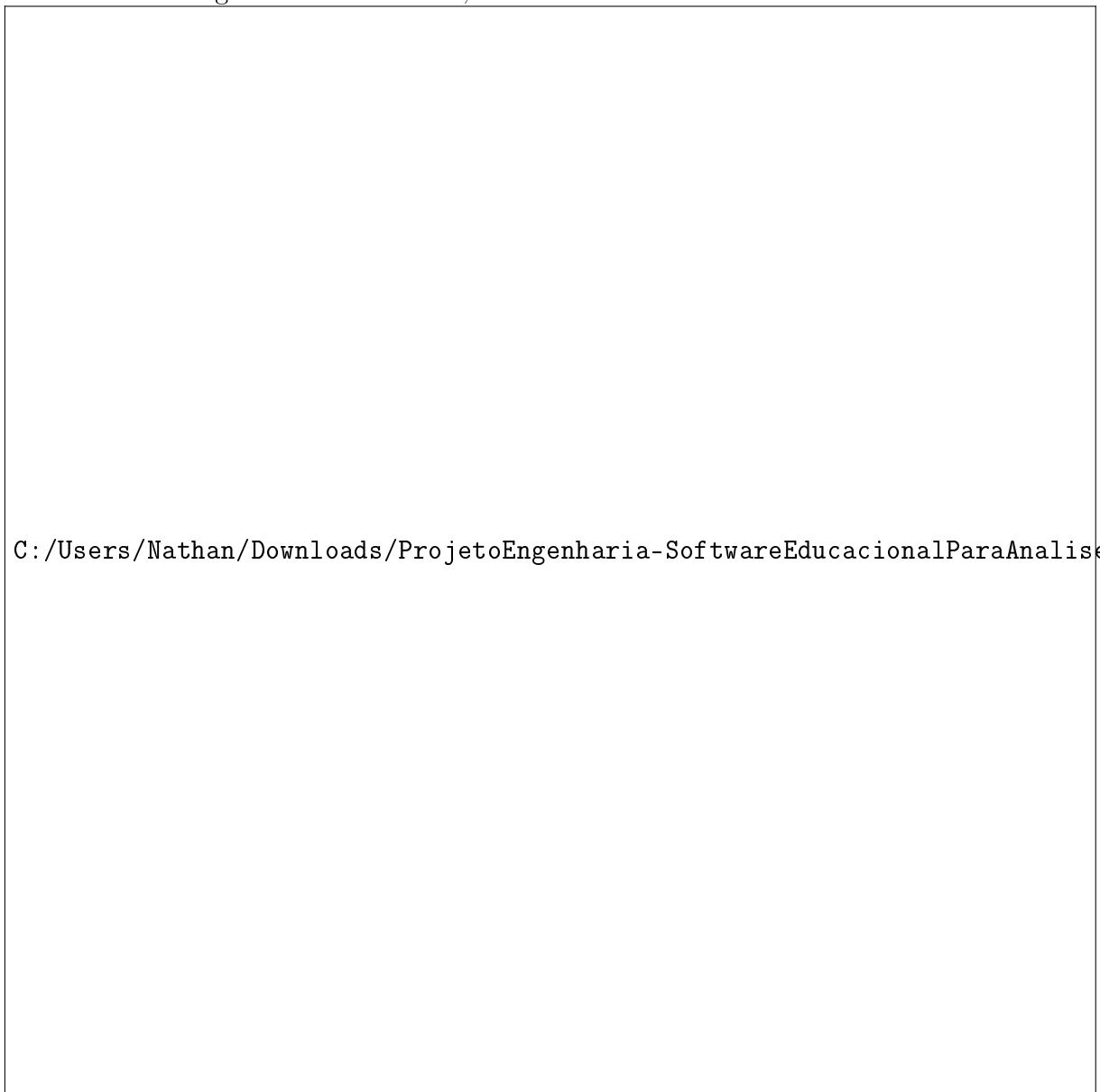
Com essa versão, o sistema passa a incorporar, além da análise hidráulica, uma abordagem mecânica de simulação, ampliando significativamente seu escopo didático e técnico na área de Engenharia de Poço.

Figura 6.5: Versão 1.1, Menu de interface do software



Fonte: Produzido pelo autor.

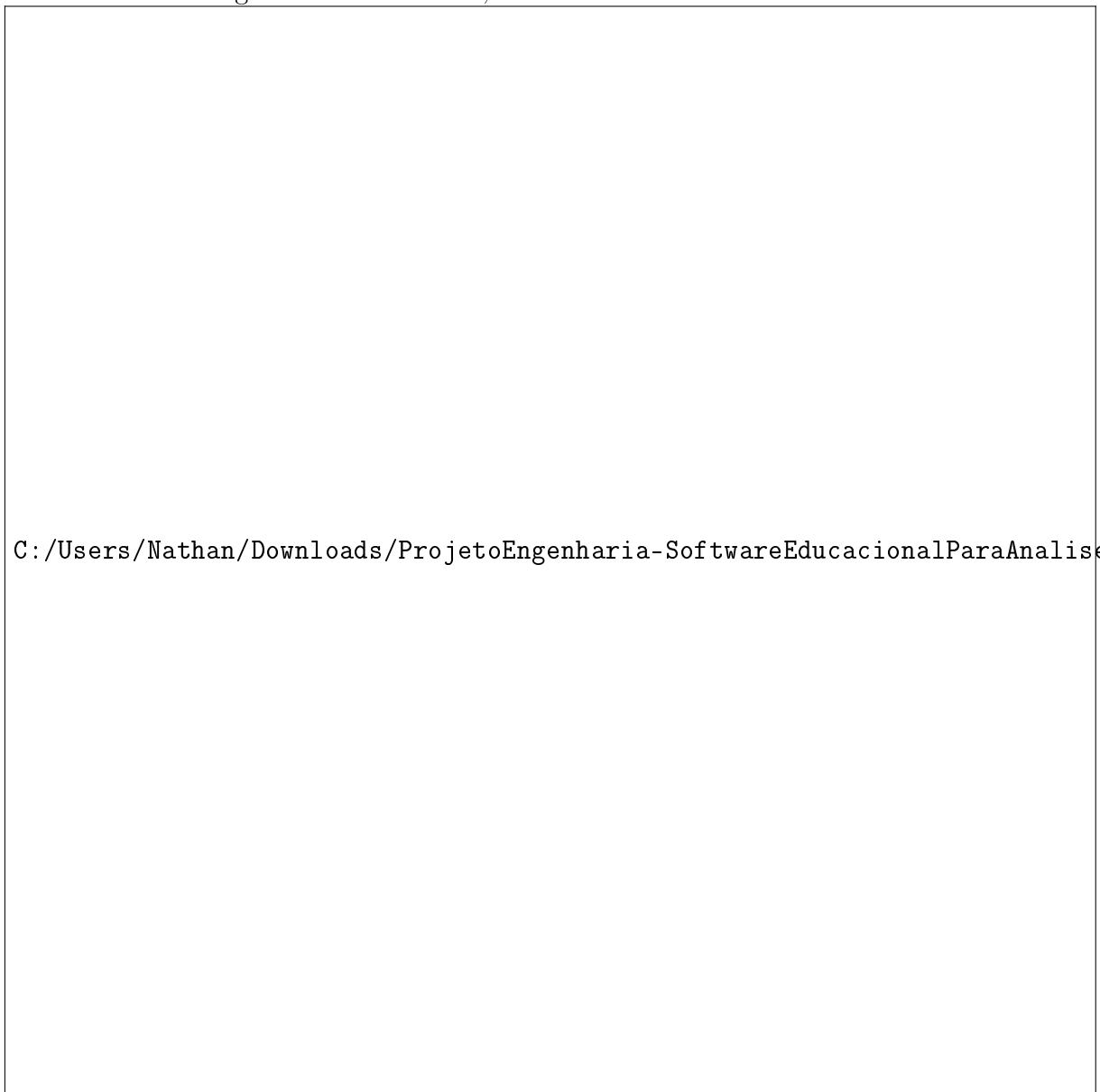
Figura 6.6: Versão 1.1, interface do modulo 01 do software



C:/Users/Nathan/Downloads/ProjetoEngenharia-SoftwareEducacionalParaAnaliseESolucao

Fonte: Produzido pelo autor.

Figura 6.7: Versão 1.1, interface do modulo 02 software



C:/Users/Nathan/Downloads/ProjetoEngenharia-SoftwareEducacionalParaAnaliseESolucao

Fonte: Produzido pelo autor.

Capítulo 7

Ciclos Construção - Implementação

Neste capítulo, são apresentados os códigos fonte implementados, além dos códigos responsáveis pela interface.

7.1 Código-Fonte (modelo)

Como visto na seção anterior, a versão 0.1 foi a última desenvolvida utilizando execução no terminal. Abaixo serão exibidas as classes necessárias para a interação via terminal.

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

Apresenta-se na listagem ?? o arquivo com código da função *main*.

Listing 7.1: Arquivo de implementação da função main

```
1 #include "CJanelaMenu.h"
2
3 #include <QApplication>
4 #include <QFile>
5
6 int main(int argc, char *argv[])
7 {
8     QApplication app(argc, argv);
9
10    QFile styleFile(":/resources/styles/lightstyle.qss");
11    styleFile.open(QFile::ReadOnly);
12    QString style(styleFile.readAll());
13    qApp->setStyleSheet(style);
14
15    QIcon appIcon(":/resources/icons/appicon.png");
16    app.setWindowIcon(appIcon);
17
18    JanelaMenu w;
```

```

19     w.setWindowIcon(appIcon);
20     w.setWindowTitle("SEAPEP - Software Educacional de Engenharia
21         de Poco");
22     w.show();
23
24     return app.exec();
25 }
```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.10 o arquivo de cabeçalho da classe CModeloReologico.

Listing 7.2: Arquivo de implementação da classe CModeloReologico

```

1 #ifndef CMODELOREOLOGICO_H
2 #define CMODELOREOLOGICO_H
3
4 #include <string>
5 #include "CObjetoPoco.h"
6
7 /*
8 Classe base abstrata para os modelos reológicos
9 Define as propriedades e métodos que devem ser implementados pelos
    modelos concretos
10 Serve como interface para modelos como newtoniano, Bingham e lei da
    potência
11 */
12
13 class CModeloReologico {
14
15 protected:
16     // Propriedades relacionadas ao escoamento e cálculos
17     double fatorFricçãoPoco = 0.0;
18     double fatorFricçãoAnular = 0.0;
19     double reynoldsPoco = 0.0;
20     double reynoldsAnular = 0.0;
21     double vMediaPoco = 0.0;
22     double vMediaAnular = 0.0;
23
24     // Tipo de fluxo (laminar ou turbulento)
25     std::string fluxoPoco;
26     std::string fluxoAnular;
27
28     // Objeto que contém as propriedades do pôco
29     CObjetoPoco* poco;
```

```

30
31 public :
32     // Construtores
33     CModeloReologico() {}
34     virtual ~CModeloReologico() {}
35     CModeloReologico(CObjetoPoco* poco) : poco(poco) {}
36
37     // Getters
38     double FatorFriccaoPoco() const { return fatorFriccaoPoco; }
39     double FatorFriccaoAnular() const { return fatorFriccaoAnular; }
40     double ReynoldsPoco() const { return reynoldsPoco; }
41     double ReynoldsAnular() const { return reynoldsAnular; }
42     double VMediaPoco() const { return vMediaPoco; }
43     double VMediaAnular() const { return vMediaAnular; }
44     std::string FluxoPoco() const { return fluxoPoco; }
45     std::string FluxoAnular() const { return fluxoAnular; }
46
47     // M todos para determinar fatores e velocidades
48     double DeterminarFatorFriccao(double re, double n);
49     double DeterminarReynoldsPoco();
50     double DeterminarReynoldsPoco(double viscosidade);
51     double DeterminarReynoldsAnular();
52     double DeterminarReynoldsAnular(double viscosidade);
53     double DeterminarVelocidadeMediaPoco();
54     double DeterminarVelocidadeMediaAnular();
55
56     // M todos puros (devem ser implementados pelas classes filhas
57     )
58     virtual std::string DeterminarFluxoPoco() = 0;
59     virtual std::string DeterminarFluxoAnular() = 0;
60     virtual double CalcularPerdaPorFriccaoPoco() = 0;
61     virtual double CalcularPerdaPorFriccaoAnular() = 0;
62 };
63 #endif // CMODELOREOLOGICO_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.11 a implementação da classe CModeloReologico.

Listing 7.3: Arquivo de implementação da classe CModeloReologico

```

1 #include <iostream>
2 #include <cmath>

```



```

34         viscosidade;
35     return reynoldsAnular;
36 }
37
38 // Calcula a velocidade m dia do fluido no interior da coluna (
39 // poco)
40 double CModeloReologico::DeterminarVelocidadeMediaPoco() {
41     vMediaPoco = poco->Vazao() / (2.448 * std::pow(poco->
42         DiametroRevestimentoID(), 2));
43     return vMediaPoco;
44 }
45
46 // Calcula a velocidade m dia no espaco anular entre a coluna e o
47 // revestimento
48 double CModeloReologico::DeterminarVelocidadeMediaAnular() {
49     vMediaAnular = poco->Vazao() /
50         (2.448 * (std::pow(poco->DiametroPoco(), 2) -
51             std::pow(poco->DiametroRevestimentoOD(), 2)));
52     ;
53     return vMediaAnular;
54 }

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.4 o arquivo de cabeçalho da classe CSimuladorPerdaTubulacao.

Listing 7.4: Arquivo de implementação da classe CModeloReologico

```

1 #ifndef CSIMULADORPERDATUBULACAO_H
2 #define CSIMULADORPERDATUBULACAO_H
3
4 #include <QMainWindow>
5 #include "CObjetoPoco.h"
6 #include "CTrechoTubulacao.h"
7 #include "CModeloNewtoniano.h"
8 #include "CModeloBingham.h"
9 #include "CModeloPotencia.h"
10 #include "qcustomplot.h" // usado pra gerar os graficos
11
12 namespace Ui {
13 class CSimuladorPerdaTubulacao;
14 }
15
16 // essa classe representa a interface principal do simulador do

```

```

    modulo 2 (perda e variacao)
17 // aqui que o usuario interage com os dados do po o , dos trechos e
    calcula L , perda, efeito balao etc
18 class CSimuladorPerdaTubulacao : public QMainWindow
19 {
20     Q_OBJECT
21
22 public:
23     // construtor e destrutor
24     explicit CSimuladorPerdaTubulacao(QWidget *parent = nullptr);
25     ~CSimuladorPerdaTubulacao();
26     QString nomeArquivo;
27     QString caminhoArquivo;
28
29 private slots:
30     // esses s o os slots que reagem aos botoes da interface
31
32     void on_btnAdicionarPropriedades_clicked();           // adiciona as
        propriedades termicas e mecanicas do fluido
33     void AtualizarDados();                                // atualiza os dados na tela
        com base no objeto do po o
34     void on_btnAdicionarTrecho_clicked();                 // adiciona um
        novo trecho de tubulacao ao po o
35     void makePlotTemperatura(double TempInicial, double TempFinal,
        profundidade, QCustomPlot* plot); // gera grafico de
        temperatura com profundidade
36     void on_btnRemoverTrecho_clicked();                   // remove um
        trecho da tubulacao
37     void makePlotPoco();                                 // desenha o
        perfil visual do po o
38     void on_btnCalcularVariacoes_clicked();             // calcula L ,
        efeito balao, forca etc
39
40     void on_actionArquivo_Dat_triggered();              // importa
        dados do arquivo .dat
41     void EditarDadosPoco();                            // edita os
        dados gerais do po o (nome, pressao etc)
42
43     // edita uma linha da tabela de fluidos ou trechos do poco
44     void EditarLinhaTabela(int row);
45
46     // opcoes do menu da interface

```

```

47     void on_actionNova_Simula_o_triggered();
48     void on_actionExportar_Como_Imagen_triggered();
49     void on_actionSobre_o_SEEP_triggered();
50     void SalvarArquivo(bool salvarComo);
51     void on_actionSalvar_como_triggered();
52     void on_actionSalvar_triggered();

53
54     //getters
55     QString NomeArquivo() { return nomeArquivo; }
56     QString CaminhoArquivo() { return caminhoArquivo; }

57
58     //setters
59     void NomeArquivo(QString nome) { nomeArquivo = nome; }
60     void CaminhoArquivo(QString caminho) { caminhoArquivo = caminho
       ; }

61
62
63 private:
64     Ui::CSimuladorPerdaTubulacao *ui; // ponteiro pra interface
       gerada pelo Qt Designer

65
66     // ponteiros para os objetos principais que compoem o modelo do
       po o
67     std::shared_ptr<COBJETOPOCO> poco = nullptr; // representa o po o como um todo
68     std::shared_ptr<CTRECHOPOCO> trechoPoco = nullptr; // trecho individual de tubulacao
69     std::shared_ptr<CFLUIDO> fluido = nullptr; // fluido associado aos trechos

70
71     // modelos reologicos usados pra calcular propriedades de
       escoamento
72     std::shared_ptr<CModeloNewtoniano> modeloNewtoniano = nullptr;
73     std::shared_ptr<CModeloBingham> modeloBingham = nullptr;
74     std::shared_ptr<CModeloPotencia> modeloPotencia = nullptr;
75 };
76
77 #endif // CSIMULADORPERDATUBULACAO_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.5 a implementação da classe CSimuladorPerdaTubulacao.

Listing 7.5: Arquivo de implementação da classe CModeloReológico

```

1 #include "CSimuladorPerdaTubulacao.h"
2 #include "ui_CSimuladorPerdaTubulacao.h"
3 #include "CJanelaAdicionarFluido.h"
4 #include "CJanelaAdicionarTrechoTubulacao.h"
5 #include "CJanelaSobreSoftware.h"
6
7 #include <iostream> // para std::cerr e std::endl
8 #include <fstream> // para std::ifstream
9 #include <sstream>
10
11 CSimuladorPerdaTubulacao::CSimuladorPerdaTubulacao(QWidget *parent)
12     : QMainWindow(parent)
13     , ui(new Ui::CSimuladorPerdaTubulacao)
14 {
15     ui->setupUi(this);
16
17
18     // Sinal para alterar os valores das caixas
19     connect(ui->editNomePoco, &QLineEdit::editingFinished, this, &
20             CSimuladorPerdaTubulacao::EditarDadosPoco);
21     connect(ui->editProfundidadeTotal, &QLineEdit::editingFinished,
22             this, &CSimuladorPerdaTubulacao::EditarDadosPoco);
23     connect(ui->editPressaoSupInicial, &QLineEdit::editingFinished,
24             this, &CSimuladorPerdaTubulacao::EditarDadosPoco);
25     connect(ui->editTemperaturaSuperiorInicial, &QLineEdit::
26             editingFinished, this, &CSimuladorPerdaTubulacao:::
27             EditarDadosPoco);
28     connect(ui->editTemperaturaFundoInicial, &QLineEdit::
29             editingFinished, this, &CSimuladorPerdaTubulacao:::
30             EditarDadosPoco);
31     connect(ui->editTemperaturaSuperiorFinal, &QLineEdit::
32             editingFinished, this, &CSimuladorPerdaTubulacao:::
33             EditarDadosPoco);
34     connect(ui->editTemperaturaFundoFinal, &QLineEdit::
35             editingFinished, this, &CSimuladorPerdaTubulacao:::
36             EditarDadosPoco);
37     connect(ui->editProfundidadeMedicao, &QLineEdit::
38             editingFinished, this, &CSimuladorPerdaTubulacao:::
39             AtualizarDados);
40
41
42     // iniciar com botões desativado

```

```

30     ui ->btnAdicionarTrecho ->setEnabled(false);
31     ui ->btnRemoverTrecho ->setEnabled(false);
32     ui ->btnCalcularVariacoes ->setEnabled(false);
33
34     connect(ui ->tblFluidos, &QTableWidget::cellChanged, this, &
35             CSimuladorPerdaTubulacao::EditarLinhaTabela);
36     connect(ui ->tblTrechos, &QTableWidget::cellChanged, this, &
37             CSimuladorPerdaTubulacao::EditarLinhaTabela);
38
39     //abrir janela no meio do monitor
40     QScreen *screen = QGuiApplication::primaryScreen();
41     QRect screenGeometry = screen->geometry();
42
43     int x = (screenGeometry.width() - this->width()) / 2;
44     int y = (screenGeometry.height() - this->height()) / 2;
45
46     this->move(x, y);
47 }
48
49 CSimuladorPerdaTubulacao::~CSimuladorPerdaTubulacao()
50 {
51     delete ui;
52 }
53
54 void CSimuladorPerdaTubulacao::EditarDadosPoco() {
55     QString nome = ui ->editNomePoco->text();
56     bool ok1, ok2, ok3, ok4, ok5, ok6, ok7, ok8;
57     double profund = ui ->editProfundidadeTotal->text().toDouble(&
58         ok1);
59     double diamPoco = ui ->editDiametroPoco->text().toDouble(&ok2);
60     double pressao = ui ->editPressaoSupInicial->text().toDouble(&
61         ok3);
62     double pressaoFim = ui ->editPressaoSupFinal->text().toDouble(&
63         ok4);
64     double temperaturaSuperiorInicial = ui ->
65         editTemperaturaSuperiorInicial->text().toDouble(&ok5);
66     double temperaturaFundoInicial = ui ->
67         editTemperaturaFundoInicial->text().toDouble(&ok6);
68     double temperaturaSuperiorFinal = ui ->
69         editTemperaturaSuperiorFinal->text().toDouble(&ok7);

```

```

64     double temperaturaFundoFinal = ui->editTemperaturaFundoFinal->
65         text().toDouble(&ok8);
66
66     if (!nome.isEmpty() && ok1 && ok2 && ok3 && ok4 && ok5 && ok6
67         && ok7 && ok8) {
68         if (!poco) {
69             // Cria o p o o
70
70             poco = std::make_unique<CObjetoPoco>(
71                 CObjetoPoco::CriarParaModulo02(nome.toStdString(),
72                     profund, diamPoco, pressao, pressaoFim,
73                     temperaturaSuperiorInicial,
74                     temperaturaFundoInicial,
75                     temperaturaSuperiorFinal, temperaturaFundoFinal,
76                     8000)
77         );
78
79         ui->btnAdicionarTrecho->setEnabled(true);
80         ui->btnRemoverTrecho->setEnabled(true);
81         ui->btnCalcularVariacoes->setEnabled(true);
82
83         ui->statusbar->showMessage("Po o criado com Sucesso!")
84             ;
85     } else {
86         // Atualiza dados do p o o j existente
87         poco->NomePoco(nome.toStdString());
88         poco->ProfundidadeTotal(profund);
89         poco->PressaoSuperficie(pressao);
90         poco->TemperaturaTopoInicial(temperaturaSuperiorInicial
91             );
92         poco->TemperaturaFundoInicial(temperaturaFundoInicial);
93         poco->TemperaturaTopoFinal(temperaturaSuperiorFinal);
94         poco->TemperaturaFundoFinal(temperaturaFundoFinal);
95         ui->statusbar->showMessage("Dados de Po o Atualizado
96             com Sucesso!");
97     }
98
99     AtualizarDados(); // Atualiza os dados calculados e a
100         interface
101 }
102 }
103 }

```

```

95 void CSimuladorPerdaTubulacao::on_btnAdicionarPropriedades_clicked
()
96 {
97     std::string nome;
98     double profundidade, diamPoco, pressaoSup, pressaoSupFim,
99         temperaturaSuperiorInicial, temperaturaFundoInicial,
100        temperaturaSuperiorFinal, temperaturaFundoFinal,
101        ProfundidadePacker;
102
103    QString text;
104
105    text = ui->editNomePoco->text();
106    nome = text.toStdString();
107
108    text = ui->editPressaoSupInicial->text();
109    pressaoSup = text.toDouble();
110
111    text = ui->editPressaoSupFinal->text();
112    pressaoSupFim = text.toDouble();
113
114    text = ui->editProfundidadeTotal->text();
115    profundidade = text.toDouble();
116
117    text = ui->editDiametroPoco->text();
118    diamPoco = text.toDouble();
119
120    text = ui->editTemperaturaSuperiorInicial->text();
121    temperaturaSuperiorInicial = text.toDouble();
122
123    text = ui->editTemperaturaFundoInicial->text();
124    temperaturaFundoInicial = text.toDouble();
125
126    text = ui->editTemperaturaSuperiorFinal->text();
127    temperaturaSuperiorFinal = text.toDouble();
128
129    text = ui->editTemperaturaFundoFinal->text();
130    temperaturaFundoFinal = text.toDouble();

131
132    if (poco) {
133        QMessageBox::StandardButton resposta = QMessageBox::
134            question(
135                this,
136                "",
137                "Ao confirmar, todos os fluidos ser\u00e3o deletados! Tem
138                certeza?",
139                QMessageBox::Yes | QMessageBox::No
140            );
141
142        if (resposta == QMessageBox::Yes) {

```

```

131     poco = std::make_unique<CObjetoPoco>(
132         CObjetoPoco::CriarParaModulo02(nome, profundidade,
133                                         diamPoco, pressaoSup, pressaoSupFim,
134                                         temperaturaSuperiorInicial,
135                                         temperaturaFundoInicial,
136                                         temperaturaSuperiorFinal, temperaturaFundoFinal,
137                                         8000)
138     );
139 }
140     AtualizarDados();
141 } else {
142
143     poco = std::make_unique<CObjetoPoco>(
144         CObjetoPoco::CriarParaModulo02(nome, profundidade,
145                                         diamPoco, pressaoSup, pressaoSupFim,
146                                         temperaturaSuperiorInicial, temperaturaFundoInicial,
147                                         temperaturaSuperiorFinal, temperaturaFundoFinal,
148                                         8000)
149     );
150 }
151 makePlotTemperatura(temperaturaSuperiorInicial,
152                     temperaturaFundoInicial, profundidade, ui->
153                     customPlotTemperaturaInicial);
154 makePlotTemperatura(temperaturaSuperiorFinal,
155                     temperaturaFundoFinal, profundidade, ui->
156                     customPlotTemperaturaFinal);
157 makePlotPoco();
158 }
159
160 void CSimuladorPerdaTubulacao::AtualizarDados()
161 {
162     ui->tblFluidos->blockSignals(true);
163     ui->tblTreichos->blockSignals(true);
164
165     if (poco){
166         // Atualiza os valores dos QLineEdits com os dados do
167         // objeto poco
168         ui->editNomePoco->setText(QString::fromStdString(poco->
169             NomePoco()));           // Profundidade total do po o
170         ui->editProfundidadeTotal->setText(QString::number(poco->
171             ProfundidadeTotal()));           // Profundidade total do

```

```

    p o o
157 ui ->editDiametroPoco ->setText (QString :: number (poco ->
    DiametroPoco ()));
158 ui ->editPressaoSupInicial ->setText (QString :: number (poco ->
    PressaoSuperficie ()); // Profundidade ocupada
159 ui ->editPressaoSupFinal ->setText (QString :: number (poco ->
    PressaoSuperficieFim ()));
160 ui ->editTemperaturaSuperiorInicial ->setText (QString :: number
    (poco ->TemperaturaTopoInicial ())); // Di metro do po o
161 ui ->editTemperaturaFundoInicial ->setText (QString :: number (
    poco ->TemperaturaFundoInicial ()); // Di metro externo do revestimento (OD)
162 ui ->editTemperaturaSuperiorFinal ->setText (QString :: number (
    poco ->TemperaturaTopoFinal ()); // Di metro interno do revestimento (ID)
163 ui ->editTemperaturaFundoFinal ->setText (QString :: number (poco
    ->TemperaturaFundoFinal ()));
    // Vaz o do fluido no po o
164 if (poco ->Packer () == true) {
165     ui ->checkBoxPacker ->setChecked (true);
166 } else {
167     ui ->checkBoxPacker ->setChecked (false);
168 }
169
170
171
172 // Atualizar QTableWidget com os dados dos trechos
173 ui ->tblTrechos ->setRowCount (static_cast < int > (poco ->Trechos
    ().size ()));
174 ui ->tblFluidos ->setRowCount (static_cast < int > (poco ->Trechos
    ().size ()));
175 int row = 0;
176 for (const auto & trecho : poco ->Trechos ()) {
177
178     ui ->tblTrechos ->setItem (row, 0, new QTableWidgetItem (
        QString :: fromStdString (trecho ->Nome ())));
179     ui ->tblTrechos ->setItem (row, 1, new QTableWidgetItem (
        QString :: number (trecho ->ProfundidadeInicial (), 'f',
        2)));
180     ui ->tblTrechos ->setItem (row, 2, new QTableWidgetItem (
        QString :: number (trecho ->ProfundidadeFinal (), 'f', 2))

```

```

        ));
181    ui->tblTrechos->setItem(row, 3, new QTableWidgetItem(
        QString::number(trecho->DiametroExterno(), 'f', 2)));
        ;
182    ui->tblTrechos->setItem(row, 4, new QTableWidgetItem(
        QString::number(trecho->DiametroInterno(), 'f', 2)));
        ;
183    ui->tblTrechos->setItem(row, 5, new QTableWidgetItem(
        QString::number(trecho->CoeficientePoisson(), 'f',
2)));
184    ui->tblTrechos->setItem(row, 6, new QTableWidgetItem(
        QString::number(trecho->CoeficienteExpancaoTermica()
, 'f', 8)));
185    ui->tblTrechos->setItem(row, 7, new QTableWidgetItem(
        QString::number(trecho->ModuloElasticidade(), 'f',
2)));
186    ui->tblTrechos->setItem(row, 8, new QTableWidgetItem(
        QString::number(trecho->PesoUnidade(), 'f', 2)));
187
188    ui->tblFluidos->setItem(row, 0, new QTableWidgetItem(
        QString::fromStdString(trecho->Fluido()->Nome())));
189    ui->tblFluidos->setItem(row, 1, new QTableWidgetItem(
        QString::number(trecho->Fluido()->Densidade(), 'f',
2)));
190    ui->tblFluidos->setItem(row, 2, new QTableWidgetItem(
        QString::number(trecho->Fluido()->Viscosidade(), 'f',
, 2)));
191
192    ++row;
193 }
194 }
195 makePlotTemperatura(poco->TemperaturaTopoInicial(), poco->
TemperaturaFundoInicial(), poco->ProfundidadeTotal(), ui->
customPlotTemperaturaInicial);
196 makePlotTemperatura(poco->TemperaturaTopoFinal(), poco->
TemperaturaFundoFinal(), poco->ProfundidadeTotal(), ui->
customPlotTemperaturaFinal);
197 makePlotPoco();
198
199 ui->tblFluidos->blockSignals(false);
200 ui->tblTrechos->blockSignals(false);
201 }

```

```

202
203 void CSimuladorPerdaTubulacao::on_btnAdicionarTrecho_clicked()
204 {
205     ui->tblTrechos->setEditTriggers(QAbstractItemView::
206         NoEditTriggers);
207     ui->tblFluidos->setEditTriggers(QAbstractItemView::
208         NoEditTriggers);
209
210     if (!poco) {
211         QMessageBox::warning(this, "Erro", "As propriedades do o
212             precisaest preenchida!");
213     }
214
215     else{
216         CJanelaAdicionarTrechoTubulacao JanelaTrecho;
217         JanelaTrecho.exec();
218
219         if (JanelaTrecho.Trecho() != "" &&
220             JanelaTrecho.ProfundidadeInicial() != "" &&
221             JanelaTrecho.ProfundidadeFinal() != "" &&
222             JanelaTrecho.DiametroExterno() != "" &&
223             JanelaTrecho.DiametroInterno() != "" &&
224             JanelaTrecho.CoeficientePoisson() != "" &&
225             JanelaTrecho.CoeficienteExpansaoTermica() != "" &&
226             JanelaTrecho.ModuloElasticidade() != "" &&
227             JanelaTrecho.PesoUnidade() != "" &&
228             JanelaTrecho.NomeFluido() != "" &&
229             JanelaTrecho.Densidade() != "" &&
230             JanelaTrecho.Viscosidade() != ""){

231
232         int numLinhas = ui->tblTrechos->rowCount();
233
234         ui->tblTrechos->insertRow(numLinhas);
235         ui->tblTrechos->setItem(numLinhas, 0, new
236             QTableWidgetItem(JanelaTrecho.Trecho()));
237         ui->tblTrechos->setItem(numLinhas, 1, new
238             QTableWidgetItem(JanelaTrecho.ProfundidadeInicial()))
239             );
240         ui->tblTrechos->setItem(numLinhas, 2, new
241             QTableWidgetItem(JanelaTrecho.ProfundidadeFinal()));
242         ui->tblTrechos->setItem(numLinhas, 3, new

```

```

236         QTableWidgetItem(JanelaTrecho.DiametroExterno()));
237     ui->tblTrechos->setItem(numLinhas, 4, new
238             QTableWidgetItem(JanelaTrecho.DiametroInterno()));
239     ui->tblTrechos->setItem(numLinhas, 5, new
240             QTableWidgetItem(JanelaTrecho.CoefficientePoisson()))
241             ;
242     ui->tblTrechos->setItem(numLinhas, 6, new
243             QTableWidgetItem(JanelaTrecho.
244                 CoeficienteExpansaoTermica()));
245     ui->tblTrechos->setItem(numLinhas, 7, new
246             QTableWidgetItem(JanelaTrecho.ModuloElasticidade()))
247             ;
248     ui->tblTrechos->setItem(numLinhas, 8, new
249             QTableWidgetItem(JanelaTrecho.PesoUnidade()));

250
251     ui->tblFluidos->insertRow(numLinhas);
252     ui->tblFluidos->setItem(numLinhas, 0, new
253             QTableWidgetItem(JanelaTrecho.NomeFluido()));
254     ui->tblFluidos->setItem(numLinhas, 1, new
255             QTableWidgetItem(JanelaTrecho.Densidade()));
256     ui->tblFluidos->setItem(numLinhas, 2, new
257             QTableWidgetItem(JanelaTrecho.Viscosidade()));

258
259     std::string NomeTrecho = JanelaTrecho.Trecho().
260             toStdString();
261     double profundInicial = JanelaTrecho.
262             ProfundidadeInicial().toDouble();
263     double profundFinal = JanelaTrecho.ProfundidadeFinal().
264             toDouble();
265     double diametroExterno = JanelaTrecho.DiametroExterno()
266             .toDouble();
267     double diametroInterno = JanelaTrecho.DiametroInterno()
268             .toDouble();
269     double coeficientePoisson = JanelaTrecho.
270             CoeficientePoisson().toDouble();
271     double coeficienteExpansaoTermica = JanelaTrecho.
272             CoeficienteExpansaoTermica().toDouble();
273     double moduloElasticidade = JanelaTrecho.
274             ModuloElasticidade().toDouble();
275     double pesoUnidade = JanelaTrecho.PesoUnidade().
276             toDouble();

277

```

```

258         std::string nome = JanelaTrecho.NomeFluido().
259             toStdString();
260         double densidade = JanelaTrecho.Densidade().toDouble();
261         double viscosidade = JanelaTrecho.Viscosidade().
262            toDouble();
263
264         auto fluido = std::make_unique<CFluido>(nome, densidade
265             , viscosidade);
266         auto trechoPoco = std::make_unique<CTrechoPoco>(
267             NomeTrecho, profundInicial, profundFinal, std::move(
268                 fluido), diametroExterno, diametroInterno,
269                 coeficientePoisson, coeficienteExpansaoTermica,
270                 moduloElasticidade, pesoUnidade);
271         poco->AdicionarTrechoPoco(std::move(trechoPoco));
272
273     }
274
275     AtualizarDados();
276 }
277
278 }
279
280 void CSimuladorPerdaTubulacao::makePlotTemperatura(double
281 TempInicial, double TempFinal, double profundidade, QCustomPlot*
282 plot)
283 {
284     // Limpa graficos anteriores
285     plot->clearItems();
286     plot->clearPlottables();
287
288     // Configura eixos
289     plot->xAxis->setLabel("Temperatura ( F )");
290     plot->yAxis->setLabel("Profundidade (ft)");
291     plot->yAxis->setRangeReversed(true); // profundidade cresce pra
292                                         baixo
293
294     // Adiciona "respiro" nos extremos
295     double margemProfundidade = profundidade * 0.05; // 5% de
296                                         margem visual
297
298     // Define pontos
299     double tempMeio = (TempInicial + TempFinal) / 2.0;
300     QVector<double> temperaturas = {TempInicial, tempMeio,
301                                     TempFinal};

```

```

288     QVector<double> profundidades = {0.0 + margemProfundidade,
289                                     profundidade / 2.0, profundidade - margemProfundidade};
290
291     // Ajuste de range
292     double tempMin = *std::min_element(temperaturas.begin(),
293                                       temperaturas.end());
294     double tempMax = *std::max_element(temperaturas.begin(),
295                                       temperaturas.end());
296
297     plot->xAxis->setRange(tempMin - 5, tempMax + 5);
298     plot->yAxis->setRange(0.0, profundidade); // Mantem 0 a
299                                         profundidade total, o respiro e visual apenas
300
301     // Grid leve
302     plot->xAxis->grid()->setVisible(true);
303     plot->yAxis->grid()->setVisible(true);
304     plot->xAxis->grid()->setPen(QPen(QColor(220, 220, 220)));
305     plot->yAxis->grid()->setPen(QPen(QColor(220, 220, 220)));
306
307     // Linha do perfil
308     QCPIGraph *perfilTemp = plot->addGraph();
309     perfilTemp->setData(temperaturas, profundidades);
310     perfilTemp->setPen(QPen(QColor(200, 0, 0), 2));
311
312     // Marcar e rotular os 3 pontos (topo, meio e fundo)
313     for (int i = 0; i < temperaturas.size(); ++i) {
314         QCPIItemEllipse *ponto = new QCPIItemEllipse(plot);
315         ponto->topLeft->setCoords(temperaturas[i] - 2,
316                                     profundidades[i] - 20);
317         ponto->bottomRight->setCoords(temperaturas[i] + 2,
318                                         profundidades[i] + 20);
319         ponto->setPen(Qt::NoPen);
320         ponto->setBrush(QBrush(Qt::darkGray));
321
322         QCPIItemText *rotulo = new QCPIItemText(plot);
323         rotulo->position->setCoords(temperaturas[i] + 4,
324                                       profundidades[i]);
325         rotulo->setText(QString::number(temperaturas[i], 'f', 1) +
326                           " ° F ");
327         rotulo->setFont(QFont("Arial", 8));
328         rotulo->setColor(Qt::darkGray);
329         // Ajusta alinhamento do rotulo do ultimo ponto para dentro

```

```

            do grafico
322     if (i == temperaturas.size() - 1)
323         rotulo->setPositionAlignment(Qt::AlignRight | Qt::
324             AlignVCenter); // alinha para a esquerda do ponto
325             final
326     else
327         rotulo->setPositionAlignment(Qt::AlignLeft | Qt::
328             AlignVCenter);
329     }
330
331     // Limpeza estatica
332     plot->legend->setVisible(false);
333     plot->setBackground(Qt::white);
334     plot->xAxis->setBasePen(QPen(Qt::black));
335     plot->yAxis->setBasePen(QPen(Qt::black));
336     plot->xAxis->setTickPen(QPen(Qt::black));
337     plot->yAxis->setTickPen(QPen(Qt::black));
338     plot->xAxis->setTickLabelColor(Qt::black);
339     plot->yAxis->setTickLabelColor(Qt::black);
340 }
341
342
343
344
345 void CSimuladorPerdaTubulacao::on_btnRemoverTrecho_clicked()
346 {
347     // pega qual linha ta selecionada em cada tabela
348     int linhaSelecionadaTrecho = ui->tblTrechos->currentRow();
349     int linhaSelecionadaFluido = ui->tblFluidos->currentRow();
350
351     // se a selecao for feita na tabela de trecho, faz a remocao
352     if (linhaSelecionadaTrecho >= 0) {
353         QMessageBox::StandardButton resposta = QMessageBox::
354             question(
355                 this,
356                 "Confirma o",
357                 "Deseja\u00e7 remover\u00e7 o\u00e1trecho\u00e1 e\u00e1o\u00e1 fluido\u00e1 associado?",
358                 QMessageBox::Yes | QMessageBox::No
359             );

```

```

359
360     if (resposta == QMessageBox::Yes) {
361         // pega o nome do trecho pela tabela (pra remover do
362         // objeto poco)
363         QString nomeTrecho = ui->tblTreichos->item(
364             linhaSelecionadaTrecho, 0)->text();
365
366         // remove a mesma linha das duas tabelas
367         ui->tblTreichos->removeRow(linhaSelecionadaTrecho);
368         ui->tblFluidos->removeRow(linhaSelecionadaTrecho);
369
370         // remove o trecho (e junto o fluido) do objeto poco
371         poco->RemoverTrechoPoco(nomeTrecho.toStdString());
372
373         // atualiza visualmente os dados
374         AtualizarDados();
375         ui->statusbar->showMessage("Trecho removido com sucesso
376                                     !");
377     }
378
379 } else if (linhaSelecionadaFluido >= 0) {
380     // se clicou so na tabela de fluido, avisa que deve usar a
381     // outra
382     QMessageBox::warning(this, "Aviso", "A remoção deve ser
383                           feita pela tabela de trechos.");
384 } else {
385     // nenhuma linha foi selecionada
386     QMessageBox::warning(this, "Erro", "Nenhuma linha foi
387                           selecionada.");
388 }
389
390
391 void CSimuladorPerdaTubulacao::makePlotPoco()
392 {
393     ui->customPlotPoco->clearItems();
394     ui->customPlotPoco->xAxis->setLabel("Dimetro (pol)");
395     ui->customPlotPoco->yAxis->setLabel("Profundidade (pol)");
396     ui->customPlotPoco->yAxis->setRangeReversed(true);
397
398     if (!poco || poco->Trechos().empty())
399         return;

```

```

395
396     // 1. Profundidade máxima e maior diâmetro externo
397     double profundidadeMaxima = 0.0;
398     double maiorDiametroExterno = 0.0;
399     for (const auto& trecho : poco->Trechos()) {
400         profundidadeMaxima = std::max(profundidadeMaxima, trecho->
401             ProfundidadeFinal());
402         maiorDiametroExterno = std::max(maiorDiametroExterno,
403             trecho->DiametroExterno());
404     }
405
406     // 2. Buraco = maior diâmetro externo + 3 polegadas
407     double diametroBuraco = maiorDiametroExterno + 3.0;
408
409     // 3. Ajuste visual no eixo X: 1.5 vezes o furo
410     double larguraGrafico = diametroBuraco * 1.5;
411     ui->customPlotPoco->xAxis->setRange(-larguraGrafico / 2.0,
412         larguraGrafico / 2.0);
413     ui->customPlotPoco->yAxis->setRange(0, profundidadeMaxima);
414
415     // 4. Cores dos fluidos
416     QMap<QString, QColor> mapaCores;
417     QVector<QColor> coresDisponiveis = {
418         QColor(70, 130, 180, 180), QColor(255, 0, 0, 150),
419         QColor(0, 255, 0, 150), QColor(0, 0, 255, 150),
420         QColor(255, 165, 0, 150), QColor(128, 0, 128, 150)
421     };
422     int corIndex = 0;
423
424     // 5. Desenho dos trechos
425     for (const auto& trecho : poco->Trechos()) {
426         double z1 = trecho->ProfundidadeInicial();
427         double z2 = trecho->ProfundidadeFinal();
428         double dExt = trecho->DiametroExterno();
429         QString nomeFluido = QString::fromStdString(trecho->Fluido
430             ()->Nome());
431
432         if (!mapaCores.contains(nomeFluido)) {
433             mapaCores[nomeFluido] = coresDisponiveis[corIndex++ %
434                 coresDisponiveis.size()];
435         }
436         QColor corFluido = mapaCores[nomeFluido];

```

```

432
433     // === Retângulo cinza (buraco do pô) ===
434     QCPIItemRect *rectBuraco = new QCPIItemRect(ui->
435         customPlotPoco);
436     rectBuraco->topLeft->setCoords(-diametroBuraco / 2.0, z1);
437     rectBuraco->bottomRight->setCoords(diametroBuraco / 2.0, z2
438         );
439     rectBuraco->setPen(QPen(Qt::black));
440     rectBuraco->setBrush(QBrush(QColor(150, 150, 150, 100)));
441
442     // === Retângulo da seção (fluído) ===
443     QCPIItemRect *rectSecao = new QCPIItemRect(ui->customPlotPoco
444         );
445     rectSecao->topLeft->setCoords(-dExt / 2.0, z1);
446     rectSecao->bottomRight->setCoords(dExt / 2.0, z2);
447     rectSecao->setPen(QPen(Qt::black));
448     rectSecao->setBrush(QBrush(corFluido));
449
450     // === Rótulo ===
451     QCPIItemText *label = new QCPIItemText(ui->customPlotPoco);
452     label->position->setCoords(0, (z1 + z2) / 2.0);
453     label->setText(nomeFluido);
454     label->setFont(QFont("Arial", 10, QFont::Bold));
455     label->setColor(Qt::black);
456     label->setPositionAlignment(Qt::AlignCenter);
457 }
458
459     // 6. Desenhar packer se existir
460     bool haPacker = poco->Packer();
461     if (haPacker == true) {
462
463         // Pega a profundidade do último trecho como profundidade do
464         // packer
465         double profundidadePacker = 0.0;
466         if (!poco->Trechos().empty()) {
467             profundidadePacker = poco->Trechos().back()->
468                 ProfundidadeFinal();
469         }
470
471         double alturaPacker = std::max(profundidadeMaxima * 0.01, 12.0)
472             ;
473         double zTop, zBottom;
474
475     }

```

```

468     // Se o packer estiver exatamente na profundidade máxima ,
469     // desenha ele todo acima
470
471     if (std::abs(profundidadePacker - profundidadeMaxima) < 1e-3) {
472         zBottom = profundidadePacker;
473         zTop = profundidadePacker - alturaPacker;
474     } else {
475         zTop = profundidadePacker - alturaPacker / 2.0;
476         zBottom = profundidadePacker + alturaPacker / 2.0;
477     }
478
479     // Encontrar o trecho correspondente      profundidade do packer
480     double diametroNoPacker = 0.0;
481
482     for (const auto& trecho : poco->Trechos()) {
483         if (profundidadePacker >= trecho->ProfundidadeInicial() &&
484             profundidadePacker <= trecho->ProfundidadeFinal()) {
485             diametroNoPacker = trecho->DiametroExterno();
486             break;
487         }
488     }
489
490     // Se n o achou trecho correspondente, usa o maior conhecido
491     // como fallback
492     if (diametroNoPacker == 0.0)
493         diametroNoPacker = maiorDiametroExterno;
494
495     // Coordenadas horizontais para os quadrados laterais
496     double xEsq1 = -diametroBuraco / 2.0;
497     double xEsq2 = -diametroNoPacker / 2.0;
498     double xDir1 = diametroNoPacker / 2.0;
499     double xDir2 = diametroBuraco / 2.0;
500
501     // === Quadrado esquerdo ===
502     QCPIItemRect* rectPackerEsq = new QCPIItemRect(ui->
503             customPlotPoco);
504     rectPackerEsq->topLeft->setCoords(xEsq1, zTop);
505     rectPackerEsq->bottomRight->setCoords(xEsq2, zBottom);
506     rectPackerEsq->setPen(QPen(Qt::red, 1.5));
507     rectPackerEsq->setBrush(Qt::NoBrush);
508
509     // === Quadrado direito ===
510     QCPIItemRect* rectPackerDir = new QCPIItemRect(ui->
511             customPlotPoco);

```

```

506     rectPackerDir->topLeft->setCoords(xDir1, zTop);
507     rectPackerDir->bottomRight->setCoords(xDir2, zBottom);
508     rectPackerDir->setPen(QPen(Qt::red, 1.5));
509     rectPackerDir->setBrush(Qt::NoBrush);

510
511     // === X vermelho esquerdo ===
512     QCPItemLine* linha1Esq = new QCPItemLine(ui->customPlotPoco
513         );
514     linha1Esq->start->setCoords(xEsq1, zTop);
515     linha1Esq->end->setCoords(xEsq2, zBottom);
516     linha1Esq->setPen(QPen(Qt::red, 1.5));

517     QCPItemLine* linha2Esq = new QCPItemLine(ui->customPlotPoco
518         );
519     linha2Esq->start->setCoords(xEsq2, zTop);
520     linha2Esq->end->setCoords(xEsq1, zBottom);
521     linha2Esq->setPen(QPen(Qt::red, 1.5));

522     // === X vermelho direito ===
523     QCPItemLine* linha1Dir = new QCPItemLine(ui->customPlotPoco
524         );
525     linha1Dir->start->setCoords(xDir1, zTop);
526     linha1Dir->end->setCoords(xDir2, zBottom);
527     linha1Dir->setPen(QPen(Qt::red, 1.5));

528     QCPItemLine* linha2Dir = new QCPItemLine(ui->customPlotPoco
529         );
530     linha2Dir->start->setCoords(xDir2, zTop);
531     linha2Dir->end->setCoords(xDir1, zBottom);
532     linha2Dir->setPen(QPen(Qt::red, 1.5));
533 }

534 // 7. Linha tracejada indicando profundidade de medi o , se
535 // v lida
536 bool ok = false;
537 double profundidadeMedicao = ui->editProfundidadeMedicao->text
538     () .toDouble(&ok);

539 // So desenha se a conversao foi bem-sucedida e o valor for
540 // maior que zero
541 if (ok && profundidadeMedicao > 0.0) {
542     QCPItemLine* linhaMedicao = new QCPItemLine(ui->

```

```

        customPlotPoco);

541     linhaMedicao->start->setCoords(-larguraGrafico / 2.0,
542         profundidadeMedicao);
543     linhaMedicao->end->setCoords(larguraGrafico / 2.0,
544         profundidadeMedicao);

545     // Define o estilo como linha tracejada preta
546     QPen pen(Qt::black);
547     pen.setStyle(Qt::DashLine);
548     pen.setWidthF(1.5);
549     linhaMedicao->setPen(pen);
550 }

551     ui->customPlotPoco->replot();
552 }

553

554 void CSimuladorPerdaTubulacao::on_btnCalcularVariacoes_clicked()
555 {
556
557     double pressaoCabeca = ui->editPressaoSupFinal->text().toDouble
558         ();
559
560     QString profundidadeStr = ui->editProfundidadeMedicao->text();
561     double profundidade = profundidadeStr.toDouble();
562
563     ui->lbnPressaoHidroestatica->setText(QString::number( (poco->
564         PressaoHidroestaticaNoPonto(profundidade)) ));
565     ui->lbnCargaInicial->setText(QString::number(poco->Carga(
566         profundidade, true)));
567
568     ui->lbnTituloDeltaLTemperatura->setText(QString::number(poco->
569         DeltaLTemperatura(profundidade)));
570     ui->lbnCargaInjecaoColunaFixa->setText(QString::number(poco->
571         CargaInjecao(profundidade)));
572     ui->lbnCargaInjecaoColunaLivre->setText(QString::number(poco->
573         Carga(profundidade, false)));
574     ui->lbnDeltaLPistaoPacker->setText(QString::number(poco->
575         DeltaLPistaoPacker(profundidade, pressaoCabeca)));
576     ui->lbnTituloDeltaLBalao->setText(QString::number(poco->
577         DeltaLEfeitoBalao(profundidade)));
578     ui->lbnDeltaLPistaoCrossover->setText(QString::number(poco->
579         DeltaLPistaoCrossover(profundidade, pressaoCabeca)));

```

```

571     ui->lbnDeltaLForcaRestauradora->setText(QString::number(poco->
572         DeltaLForcaRestauradora(profundidade, pressaoCabeca)));
573
574
575 void CSimuladorPerdaTubulacao::on_actionArquivo_Dat_triggered()
576 {
577     QString caminhoDoArquivo = QFileDialog::getOpenFileName(
578         this,
579         "Selecione um arquivo",
580         "",
581         "Todos os arquivos (*.*)"
582     );
583
584     std::string caminhoDoArquivoStr = caminhoDoArquivo.toStdString()
585         ();
586     std::ifstream file(caminhoDoArquivoStr);
587
588     if (!file.is_open()) {
589         ui->statusbar->showMessage("Falha ao abrir o arquivo!");
590         return;
591     }
592
593     std::string linha;
594     bool lendoTreichos = false;
595
596     while (std::getline(file, linha)) {
597         if (linha.find("Configuracao dos Fluidos") != std::string::
598             npos) {
599             lendoTreichos = true;
600             continue;
601         }
602
603         if (linha.empty() || linha[0] == '#') {
604             continue;
605
606             if (!lendoTreichos) {
607                 // Leitura dos dados do po o
608                 std::istringstream iss(linha);
609                 std::string nome;
610                 double profundidade, diamPoco, pressaoSup,

```

```

    pressaoSupFim;
610
    double temperaturaSuperiorInicial ,
       temperaturaFundoInicial;
611
    double temperaturaSuperiorFinal , temperaturaFundoFinal;
612
    bool haPacker;

613
614     if (iss >> nome >> profundidade >> diamPoco >>
615         pressaoSup >> pressaoSupFim
616         >> temperaturaSuperiorInicial >>
617         temperaturaFundoInicial
618         >> temperaturaSuperiorFinal >>
619         temperaturaFundoFinal >> haPacker) {
620
621
622         if (haPacker == false){
623             ui->btnAdicionarTrecho->setEnabled(true);
624             ui->btnRemoverTrecho->setEnabled(true);
625             ui->btnCalcularVariacoes->setEnabled(true);
626
627
628         poco = std::make_unique<CObjetoPoco>(
629             CObjetoPoco::CriarParaModulo02(nome,
630                 profundidade, diamPoco, pressaoSup,
631                 pressaoSupFim,
632
633                 temperaturaSuperiorInicial
634
635                 ,
636                 temperaturaFundoInicial
637
638                 ,
639                 temperaturaSuperiorFinal
640
641                 ,
642                 temperaturaFundoFinal
643                 ,
644                 haPacker)
645             );
646
647         } else {
648             std::cerr << "Erro ao ler linha de po o : " <<
649             linha << std::endl;
650
651         }
652
653     } else {
654
655         // Leitura dos dados dos trechos

```

```

638         std::istringstream iss(linha);
639         std::string nomeTrecho, nomeFluido;
640         double profundInicial, profundFinal;
641         double diametroExterno, diametroInterno;
642         double coeficientePoisson, coeficienteExpansaoTermica;
643         double moduloElasticidade, pesoUnidade;
644         double densidade, viscosidade;
645
646         if (iss >> nomeTrecho >> profundInicial >> profundFinal
647             >> diametroExterno >> diametroInterno
648             >> coeficientePoisson >> coeficienteExpansaoTermica
649             >> moduloElasticidade >> pesoUnidade
650             >> nomeFluido >> densidade >> viscosidade) {
651
652             auto fluido = std::make_unique<CFluido>(nomeFluido,
653                 densidade, viscosidade);
654             auto trechoPoco = std::make_unique<CTrechoPoco>(
655                 nomeTrecho,
656                 profundInicial, profundFinal, std::move(fluido)
657
658                 ,
659                 diametroExterno, diametroInterno,
660                 coeficientePoisson, coeficienteExpansaoTermica,
661                 moduloElasticidade, pesoUnidade
662             );
663
664             if (!poco->AdicionarTrechoPoco(std::move(trechoPoco))
665             )) {
666                 std::cerr << "Falha ao adicionar trecho ao
667                 poco.\n";
668             }
669
670         file.close();
671         AtualizarDados();
672         ui->statusbar->showMessage("Dados importados com sucesso!");
673     }

```

```

674
675
676 void CSimuladorPerdaTubulacao::on_actionNova_Simula_o_triggered()
677 {
678     QMessageBox::StandardButton resposta = QMessageBox::question(
679         this,
680         "",
681         "Tem certeza que deseja iniciar uma nova simula o ?",
682         QMessageBox::Yes | QMessageBox::No
683     );
684
685     if (resposta == QMessageBox::Yes) {
686         CSimuladorPerdaTubulacao *newWindow = new
687             CSimuladorPerdaTubulacao();
688         newWindow->show();
689         this->close();
690     }
691
692
693 void CSimuladorPerdaTubulacao::
694     on_actionExportar_Como_Imagem_triggered()
695 {
696     QString fileName = QFileDialog::getSaveFileName(this, "Salvar
697         imagem", "", "PNG (*.png);;JPEG (*.jpg)");
698
699     if (!fileName.isEmpty()) {
700         QPixmap pixmap = this->grab();
701         pixmap.save(fileName);
702     }
703
704 void CSimuladorPerdaTubulacao::on_actionSobre_o_SEEP_triggered()
705 {
706     CJanelaSobreSoftware janelaSobre;
707     janelaSobre.setWindowTitle("Sobre o Software");
708     janelaSobre.exec();
709 }
710
711
712 void CSimuladorPerdaTubulacao::SalvarArquivo(bool salvarComo)

```

```

713 {
714     QString caminho;
715
716     // Se for salvarComo ou ainda não tiver caminho, abrir o
717     // di logo
718     if (salvarComo || CaminhoArquivo().isEmpty()) {
719         caminho = QFileDialog::getSaveFileName(this, "Salvar"
720             " Arquivo", "", "Arquivo.dat (*.dat)");
721         if (caminho.isEmpty()) return; // usuário cancelou
722         CaminhoArquivo(caminho);
723         NomeArquivo(QFileInfo(caminho).fileName());
724     } else {
725         caminho = CaminhoArquivo(); // salva direto
726     }
727
728     QFile arquivo(caminho);
729     if (!arquivo.open(QIODevice::WriteOnly | QIODevice::Text)) {
730         QMessageBox::warning(this, "Erro", "Não foi possível salvar"
731             " o arquivo.");
732         return;
733     }
734
735     QTextStream out(&arquivo);
736
737     out << "# Configuração do Poco"
738     -----
739     n";
740     out << "#"
741         << QString("Nome").leftJustified(35, ' ')
742         << QString("Profundidade(ft)").leftJustified(35, ' ')
743         << QString("Diâmetro do Pó p(ft)").leftJustified(35, ' ')
744         << QString("Pressão Sup. Inicial(psi)").leftJustified(35,
745             ' ')
746         << QString("Pressão Sup. Final(psi)").leftJustified(35, ' ')
747         << QString("Tempo Sup. Inicial(F)").leftJustified(35, ' ')
748             )
749         << QString("Tempo Fund. Inicial(F)").leftJustified(35, ' ')
750             )
751         << QString("Tempo Sup. Final(F)").leftJustified(35, ' ')
752         << QString("Tempo Fund. Final(F)").leftJustified(35, ' ')
753         << QString("Profundidade(ft)").leftJustified(35, ' ')

```

```

746     << "\n";
747
748     out << " "
749     << ui->editNomePoco->text().leftJustified(35, ' ')
750     << ui->editProfundidadeTotal->text().leftJustified(35, ' ')
751     << ui->editDiametroPoco->text().leftJustified(35, ' ')
752     << ui->editPressaoSupInicial->text().leftJustified(35, ' ')
753     << ui->editPressaoSupFinal->text().leftJustified(35, ' ')
754     << ui->editTemperaturaSuperiorInicial->text().leftJustified
755         (35, ' ')
756     << ui->editTemperaturaFundoInicial->text().leftJustified
757         (35, ' ')
758     << ui->editTemperaturaSuperiorFinal->text().leftJustified
759         (35, ' ')
760     << ui->editTemperaturaFundoFinal->text().leftJustified(35,
761         ' ');
762
763     // Checkbox de saida
764     if (ui->checkBoxPacker->isChecked()) {
765         out << "true";
766     } else {
767         out << "false";
768     }
769
770     out << "\n";
771
772     // Escreve os dados dos fluidos
773     out << "\n\n#\u2014Configuracao\u2014dos\u2014Fluidos\u2014
774
775     -----
776
777     n";
778     out << "#"
779     << QString("Nome\u2014Trecho").leftJustified(25, ' ')
780     << QString("Prof.\u2014Inicial\u2014(ft)").leftJustified(25, ' ')
781     << QString("Prof.\u2014Final\u2014(ft)").leftJustified(25, ' ')
782     << QString("Diam.\u2014externo\u2014(in)").leftJustified(25, ' ')
783     << QString("Diam.\u2014interno\u2014(in)").leftJustified(25, ' ')
784     << QString("Coef.\u2014Poisson").leftJustified(25, ' ')
785     << QString("Coef.\u2014Exp.\u2014Term.\u2014(1/F)").leftJustified(25, ' ')
786     << QString("Mod.\u2014Elast.\u2014(psi)").leftJustified(25, ' ')
787     << QString("Peso/unid\u2014(lb/ft)").leftJustified(25, ' ')
788     << QString("Nome\u2014fluido").leftJustified(25, ' ')

```

```

782     << QString("Densidade (lbm/gal)").leftJustified(25, ' ')
783     << QString("Viscosidade (cP)").leftJustified(25, ' ')
784     << "\n";
785
786     int linhas = ui->tblFluidos->rowCount();
787     for (int i = 0; i < linhas; ++i) {
788         out << " " // recuo
789         << ui->tblTrechos->item(i, 0)->text().leftJustified(25,
790             ' ')
791         << ui->tblTrechos->item(i, 1)->text().leftJustified(25,
792             ' ')
793         << ui->tblTrechos->item(i, 2)->text().leftJustified(25,
794             ' ')
795         << ui->tblTrechos->item(i, 3)->text().leftJustified(25,
796             ' ')
797         << ui->tblTrechos->item(i, 4)->text().leftJustified(25,
798             ' ')
799         << ui->tblTrechos->item(i, 5)->text().leftJustified(25,
800             ' ')
801         << ui->tblTrechos->item(i, 6)->text().leftJustified(25,
802             ' ')
803         << ui->tblTrechos->item(i, 7)->text().leftJustified(25,
804             ' ')
805
806     // Atualiza o caminho salvo apenas se for novo
807     if (CaminhoArquivo().isEmpty())
808     {
809         CaminhoArquivo(caminho);
810         NomeArquivo(QFileInfo(caminho).fileName());
811     }

```

```

812
813     QMessageBox::information(this, "Salvo", "Arquivo salvo com
814     sucesso!");
815 }
816 void CSimuladorPerdaTubulacao::on_actionSalvar_triggered()
817 {
818     SalvarArquivo(false); // salvar direto
819 }
820
821 void CSimuladorPerdaTubulacao::on_actionSalvar_como_triggered()
822 {
823     SalvarArquivo(true); // for ar abrir QFileDialog
824 }
825
826 // essa funcao edita os dados do fluido e do trecho com base na
827 // linha da tabela de fluidos
828 void CSimuladorPerdaTubulacao::EditarLinhaTabela(int row)
829 {
830     // garante que o ndice da linha v lido
831     if (row < 0 || row >= poco->Trechos().size()) {
832         return;
833     }
834     // obtém o trecho e fluido da mesma linha
835     CTrechoPoco* trecho = poco->Trechos().at(row);
836     CFluido* fluido = trecho->Fluido();
837
838     if (!fluido) return;
839
840     trecho->Nome(ui->tblTrechos->item(row, 0)->text().toStdString())
841         ;
842     trecho->ProfundidadeInicial(ui->tblTrechos->item(row, 1)->text()
843         .toDouble());
844     trecho->ProfundidadeFinal(ui->tblTrechos->item(row, 2)->text()
845         .toDouble());
846     trecho->DiametroExterno(ui->tblTrechos->item(row, 3)->text()
847         .toDouble());
848     trecho->DiametroInterno(ui->tblTrechos->item(row, 4)->text()
849         .toDouble());
850     trecho->CoeficientePoisson(ui->tblTrechos->item(row, 5)->text()
851         .toDouble());

```

```

846     trecho->CoeficienteExpancaoTermica(ui->tblTreichos->item(row, 6)
847         ->text().toDouble());
848     trecho->ModuloElasticidade(ui->tblTreichos->item(row, 7)->text()
849         .toDouble());
850     trecho->PesoUnidade(ui->tblTreichos->item(row, 8)->text().
851         toDouble());
852
853     fluido->Nome(ui->tblFluidos->item(row, 0)->text().toStdString())
854         );
855     fluido->Densidade(ui->tblFluidos->item(row, 1)->text().toDouble()
856         );
857     fluido->Viscosidade(ui->tblFluidos->item(row, 2)->text().
858         toDouble());
859
860     // atualiza valores globais do simulador
861     AtualizarDados();
862
863     ui->statusbar->showMessage("Fluido e profundidades atualizados com sucesso!");
864 }

```

Fonte: produzido pelo autor.

Apresenta-se na listagem ?? o arquivo de cabeçalho da classe CObjetoPoco.

Listing 7.6: Arquivo de implementação da classe CObjetoPoco

```

1 #ifndef COBJETOPOCO_H
2 #define COBJETOPOCO_H
3
4 #include <vector>
5 #include <memory>
6 #include "CTrechoTubulacao.h"
7
8 // Classe CObjetoPoco representa o objeto principal que armazena os
9 // dados do poço e permite a execução de cálculos e
10 // simulações
11
12 class CObjetoPoco {
13 protected:
14     std::string nomePoco;
15     double profundidadeFinal = 0.0;
16     double profundidadeOcupada = 0.0;
17     double pressaoSuperficie = 0.0;
18     double diametroPoco = 0.0;

```

```

17     double diametroRevestimentoOD = 0.0;
18     double diametroRevestimentoID = 0.0;
19     double vazao = 0.0;
20     std::vector<std::unique_ptr<CTrechoPoco>> trechos;
21
22     double pressaoSuperficieFim = 0.0;
23     double temperaturaTopoInicial = 0.0;
24     double temperaturaFundoInicial = 0.0;
25     double temperaturaTopoFinal = 0.0;
26     double temperaturaFundoFinal = 0.0;
27     double packer = true;
28
29 public:
30     // Construtor e destrutor padro
31     CObjetoPoco() = default;
32     ~CObjetoPoco() = default;
33
34     // Evita c pia (pra evitar duplica o desnecessaria dos
35     // trechos)
36     CObjetoPoco(const CObjetoPoco&) = delete;
37     CObjetoPoco& operator=(const CObjetoPoco&) = delete;
38
39     // Permite movimenta o (move semantics)
40     CObjetoPoco(CObjetoPoco&&) = default;
41     CObjetoPoco& operator=(CObjetoPoco&&) = default;
42
43     // M todos de cria o est ticos, separando claramente os
44     // dados exigidos por cada m dulo
45     static CObjetoPoco CriarParaModulo01(std::string Nome, double
46         Profund, double PressaoSup, double D, double OD, double ID,
47         double q);
48     static CObjetoPoco CriarParaModulo02(std::string Nome, double
49         Profund, double diamPoco, double PressaoSup, double
50         PressaoSupFinal, double TempTopoInicial, double
51         TempFundoInicial, double TempTopoFinal, double
52         TempFundoFinal, bool haPacker);
53
54     // Getters para acessar os atributos de forma segura
55     std::string NomePoco() const { return nomePoco; }
56     double ProfundidadeTotal() const { return profundidadeFinal; }
57     double ProfundidadeOcupada() const { return profundidadeOcupada
58         ; }

```

```

50     double PressaoSuperficie() const { return pressaoSuperficie; }
51     double PressaoSuperficieFim() const { return
52         pressaoSuperficieFim; }
53     double DiametroPoco() const { return diametroPoco; }
54     double DiametroRevestimentoOD() const { return
55         diametroRevestimentoOD; }
56     double DiametroRevestimentoID() const { return
57         diametroRevestimentoID; }
58     double Vazao() const { return vazao; }
59     double TemperaturaTopoInicial() const { return
60         temperaturaTopoInicial; }
61     double TemperaturaFundoInicial() const { return
62         temperaturaFundoInicial; }
63     double TemperaturaTopoFinal() const { return
64         temperaturaTopoFinal; }
65     double TemperaturaFundoFinal() const { return
66         temperaturaFundoFinal; }
67     bool Packer() const { return packer; }

68     // Retorna os trechos adicionados ao poço (em forma de
69     // ponteiros brutos)
70     std::vector<CTrechoPoco*> Trechos() const;

71     // Setters permitem modificar os atributos do poço
72     void NomePoco(std::string Nome) { nomePoco = Nome; }
73     void ProfundidadeTotal(double Profundidade) { profundidadeFinal
74         = Profundidade; }
75     void ProfundidadeOcupada(double Profundidade) {
76         profundidadeOcupada = Profundidade; }
77     void PressaoSuperficie(double PressaoSuperior) {
78         pressaoSuperficie = PressaoSuperior; }
79     void PressaoSuperficieFim(double PressaoSuperior) {
80         pressaoSuperficieFim = PressaoSuperior; }
81     void DiametroPoco(double D) { diametroPoco = D; }
82     void DiametroRevestimentoOD(double DiametroExterno) {
83         diametroRevestimentoOD = DiametroExterno; }
84     void DiametroRevestimentoID(double DiametroInterno) {
85         diametroRevestimentoID = DiametroInterno; }
86     void Vazao(double q) { vazao = q; }
87     void TemperaturaTopoInicial(double temperatura) {
88         temperaturaTopoInicial = temperatura; }
89     void TemperaturaFundoInicial(double temperatura) {
90

```

```

        temperaturaFundoInicial = temperatura; }
77 void TemperaturaTopoFinal(double temperatura) {
    temperaturaTopoFinal = temperatura; }
78 void TemperaturaFundoFinal(double temperatura) {
    temperaturaFundoFinal = temperatura; }
79 void Packer(bool haPacker) { packer = haPacker; }

80

81 // Métodos de cálculo
82 double PressaoHidroestaticaTotal() const;
83 double PressaoHidroestaticaNoPonto(double profundidade) const;
84 double DensidadeEfetivaTotal() const;
85 double ViscosidadeEfetivaTotal() const;
86 bool VerificarPreenchimentoColuna();
87 double Carga(double profundidade, bool inicio) const;
88 double DeltaLTemperatura(double profundidade) const;
89 double DeltaLEfeitoBalão(double profundidade) const;
90 double VariacaoCargaDevidoCrossover(double profundidade, double
    pressaoCabecaPoco = -1) const;
91 double VariacaoCargaEfeitoPistão(double profundidade, double ID
    , double OD) const;
92 double DeltaLPistãoPacker(double profundidade, double
    CargaPistão) const;
93 double DeltaLPistãoCrossover(double profundidade, double
    pressaoCabecaPoco) const;
94 double DeltaLForçaRestauradora(double profundidade, double
    pressaoCabecaPoco) const;
95 double CargaInjeção(double profundidade) const;
96 double TemperaturaNoPonto(double profundidade, double T_topo,
    double T_Fundo) const;

97

98 bool AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco>
    trechoParaAdicionar); // Função para adicionar um novo
    trecho ao poço
99 void RemoverFluidoPoco(const std::string& nomeFluido); // Remove
    trecho do poço com base no nome do fluido
100 void RemoverTrechoPoco(const std::string& nomeTrecho);

101

102 // Métodos que retornam dados para gráficos
103 std::pair<std::vector<double>, std::vector<double>>
    PlotarProfundidadePorPressão();
104 std::pair<std::vector<double>, std::vector<double>>

```

```

    PlotarProfundidadePorPressaoMedia();
106 };
107
108 #endif

```

Fonte: produzido pelo autor.

Apresenta-se na listagem ?? a implementação da classe CObjetoPoco.

Listing 7.7: Arquivo de implementação da classe CObjetoPoco

```

1 #include "CObjetoPoco.h"
2 #include <iostream>
3 #include <vector>
4 #include <fstream>
5 #include <cstdlib>
6 #include <numbers>
7 #include <math.h>
8 #include <QDebug>
9
10
11 CObjetoPoco CObjetoPoco::CriarParaModulo01(std::string nomeDoPoco,
12                                              double profundidadeFinalDoPoco, double pressaoNaSuperficie,
13                                              double diametroDoPoco,
14                                              double
15                                              diametroRevestimentoExterno
16                                              , double
17                                              diametroRevestimentoInterno
18                                              , double vazaoDoPoco)
19 {
20     CObjetoPoco objetoPoco;
21
22     objetoPoco.nomePoco = nomeDoPoco;
23     objetoPoco.profundidadeFinal = profundidadeFinalDoPoco;
24     objetoPoco.pressaoSuperficie = pressaoNaSuperficie;
25     objetoPoco.diametroPoco = diametroDoPoco;
26     objetoPoco.diametroRevestimentoOD = diametroRevestimentoExterno
27         ;
28     objetoPoco.diametroRevestimentoID = diametroRevestimentoInterno
29         ;
30     objetoPoco.vazao = vazaoDoPoco;
31
32     return objetoPoco;
33 }
34

```

```

26
27 CObjetoPoco CObjetoPoco::CriarParaModulo02(std::string nomeDoPoco ,
28     double profundidadeFinalDoPoco, double diamPoco, double
29     pressaoNaSuperficie, double pressaoNaSuperficieFim,
30     double
31     temperaturaTopoInicial
32     , double
33     temperaturaFundoInicial
34     ,
35     double
36     temperaturaTopoFinal ,
37     double
38     temperaturaFundoFinal
39     , bool haPacker) {
40
41     CObjetoPoco objetoPoco;
42
43     objetoPoco.nomePoco = nomeDoPoco;
44     objetoPoco.profundidadeFinal = profundidadeFinalDoPoco;
45     objetoPoco.diametroPoco = diamPoco;
46     objetoPoco.pressaoSuperficie = pressaoNaSuperficie;
47     objetoPoco.pressaoSuperficieFim = pressaoNaSuperficieFim;
48     objetoPoco.temperaturaTopoInicial = temperaturaTopoInicial;
49     objetoPoco.temperaturaFundoInicial = temperaturaFundoInicial;
50     objetoPoco.temperaturaTopoFinal = temperaturaTopoFinal;
51     objetoPoco.temperaturaFundoFinal = temperaturaFundoFinal;
52     objetoPoco.haPacker = packer;
53
54     return objetoPoco;
55 }
56
57
58
59 std::vector<CTrechoPoco*> CObjetoPoco::Trechos() const {
60     std::vector<CTrechoPoco*> vetorDePonteirosParaTrechos;
61
62     // percorre todos os trechos armazenados no p o o
63     for (const auto& trechoUnico : trechos) {
64         // adiciona o ponteiro cru ( n o nico ) de cada trecho ao
65         // vetor de sa da
66         vetorDePonteirosParaTrechos.push_back(trechoUnico.get());
67     }
68
69     // retorna o vetor contendo os ponteiros dos trechos

```

```

57     return vetorDePonteirosParaTrechos;
58 }
59
60 bool CObjetoPoco::AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco>
61                                         trechoParaAdicionar) {
62     // calcula o comprimento do trecho com base nas profundidades
63     // inicial e final
64     double comprimentoDoTrecho = trechoParaAdicionar->
65         ProfundidadeFinal() - trechoParaAdicionar->
66         ProfundidadeInicial();
67
68     // move o trecho para dentro do vetor principal do p o o
69     // trechos.push_back(std::move(trechoParaAdicionar));
70
71     // atualiza a profundidade total ocupada no p o o somando o
72     // novo trecho
73     profundidadeOcupada += comprimentoDoTrecho;
74
75     return true; // opera o realizada com sucesso
76 }
77
78
79
80
81
82
83
84 }
85
86 void CObjetoPoco::RemoverFluidoPoco(const std::string& nomeFluido)
87 {
88     for (auto it = trechos.begin(); it != trechos.end();) {
89         if ((*it)->Fluido()->Nome() == nomeFluido) {
90             double comprimento = (*it)->ProfundidadeFinal() - (*it)
91                         ->ProfundidadeInicial();
92             it = trechos.erase(it);
93             profundidadeOcupada -= comprimento;
94         } else {
95             ++it;
96         }
97     }
98 }
99
100 void CObjetoPoco::RemoverTrechoPoco(const std::string& nomeTrecho)
101 {
102     for (auto it = trechos.begin(); it != trechos.end();) {
103         if ((*it)->Nome() == nomeTrecho) {
104             double comprimento = (*it)->ProfundidadeFinal() - (*it)
105                         ->ProfundidadeInicial();
106         }
107     }
108 }
```

```

90         it = trechos.erase(it);
91         profundidadeOcupada -= comprimento;
92     } else {
93         ++it;
94     }
95 }
96 }
97
98 double CObjetoPoco::PressaoHidroestaticaTotal() const {
99     double pressaoTotalHidrostatica = 0.0;
100
101    // soma a pressao hidrostatica de todos os trechos do po o
102    for (const auto& trechoAtual : trechos) {
103        pressaoTotalHidrostatica += trechoAtual->
104            PressaoHidroestatica();
105    }
106
107    // adiciona a pressao da superficie para obter a pressao total
108    return pressaoTotalHidrostatica + pressaoSuperficie;
109 }
110
111 double CObjetoPoco::PressaoHidroestaticaNoPonto(double
112     profundidadeDesejada) const {
113     double pressaoAcumulada = pressaoSuperficie;
114     double profundidadeJaCalculada = 0.0;
115
116     // percorre cada trecho verificando se ele contem a
117     // profundidade desejada
118     for (const auto& trechoAtual : trechos) {
119         double comprimentoDoTrecho = trechoAtual->ProfundidadeFinal
120             () - trechoAtual->ProfundidadeInicial();
121
122         // se a profundidade estiver dentro do trecho atual
123         if (profundidadeDesejada <= profundidadeJaCalculada +
124             comprimentoDoTrecho) {
125             double profundidadeDentroDoTrecho =
126                 profundidadeDesejada - profundidadeJaCalculada;
127
128             // adiciona somente a parte proporcional da pressao no
129             // trecho
130             pressaoAcumulada += trechoAtual->PressaoHidroestatica(
131                 profundidadeDentroDoTrecho);

```

```

124         break;
125     } else {
126         // adiciona a pressao total do trecho completo
127         pressaoAcumulada += trechoAtual->PressaoHidroestatica()
128         ;
129         profundidadeJaCalculada += comprimentoDoTrecho;
130     }
131 }
132 return pressaoAcumulada;
133 }

134

135 bool CObjetoPoco::VerificarPreenchimentoColuna() {
136     double profundidadeNaoPreenchida = profundidadeFinal -
137         profundidadeOcupada;
138
139     if (profundidadeNaoPreenchida > 0) {
140         std::cout << "Uma coluna de " << profundidadeNaoPreenchida
141             << " uft de fluido precisa ser adicionada!" << std::endl;
142         return false; // coluna incompleta
143     } else {
144         std::cout << "A coluna de fluidos equivale a profundade
145             total de " << std::endl;
146         return true; // coluna completamente preenchida
147     }
148 }

149 double CObjetoPoco::DensidadeEfetivaTotal() const {
150     double somaDensidadePonderada = 0.0;
151     double comprimentoTotalDaColuna = 0.0;
152
153     for (const auto& trechoAtual : trechos) {
154         double comprimentoTrecho = trechoAtual->ProfundidadeFinal()
155             - trechoAtual->ProfundidadeInicial();
156
157         // multiplica a densidade equivalente pelo comprimento do
158         // trecho para ponderar
159         somaDensidadePonderada += trechoAtual->DensidadeEquivalente
160             () * comprimentoTrecho;
161         comprimentoTotalDaColuna += comprimentoTrecho;
162     }

```

```

159
160     return somaDensidadePonderada / comprimentoTotalDaColuna;
161 }
162
163 double CObjetoPoco::ViscosidadeEfetivaTotal() const {
164     double somaDasViscosidades = 0.0;
165
166     // percorre todos os trechos para somar as viscosidades dos
167     // fluidos
168     for (const auto& trechoAtual : trechos) {
169         somaDasViscosidades += trechoAtual->Fluido()->Viscosidade()
170         ;
171     }
172
173     // retorna a media simples das viscosidades
174     return somaDasViscosidades / trechos.size();
175 }
176
177 std::pair<std::vector<double>, std::vector<double>> CObjetoPoco::
178 PlotarProfundidadePorPressaoMedia() {
179     std::vector<double> vetorDeProfundidades;
180     std::vector<double> vetorDePressoes;
181
182     // percorre cada metro ao longo da profundidade total do po o
183     for (double profundidadeAtual = 0.0; profundidadeAtual <=
184         ProfundidadeTotal(); profundidadeAtual += 1.0) {
185         vetorDeProfundidades.push_back(profundidadeAtual);
186
187         // calcula a pressao no ponto atual usando a funcao
188         // apropriada
189         double pressaoNoPonto = PressaoHidroestaticaNoPonto(
190             profundidadeAtual);
191         vetorDePressoes.push_back(pressaoNoPonto);
192     }
193
194     // retorna as duas listas para serem usadas na geracao do
195     // grafico
196     return std::make_pair(vetorDeProfundidades, vetorDePressoes);
197 }
198
199 #include <QDebug>
200
201

```

```

194 double CObjetoPoco::Carga(double profundidade, bool inicio) const {
195     double cargaTotal = 0.0;
196     double pi = 3.141592653589793;
197
198     if (trechos.empty())
199         return 0.0;
200
201     // 1. Pressao no fundo do poco (base)
202     double profundidadeBase = 0.0;
203     double OD_base = 0.0;
204     double ID_base = 0.0;
205
206     for (const auto& trecho : trechos) {
207         if (trecho->ProfundidadeFinal() > profundidadeBase) {
208             profundidadeBase = trecho->ProfundidadeFinal();
209             OD_base = trecho->DiametroExterno();
210             ID_base = trecho->DiametroInterno();
211         }
212     }
213
214     // 2. Carga por pressao no fundo (negativa)
215     double pressaoBase = PressaoHidroestaticaNoPonto(
216         profundidadeBase);
217     if (inicio != false){
218         pressaoBase = PressaoHidroestaticaNoPonto(profundidadeBase)
219             + PressaoSuperficieFim();
220     }
221     double areaBase = (pi / 4.0) * (OD_base * OD_base - ID_base *
222         ID_base);
223     double cargaPressao = -1.0 * pressaoBase * areaBase;
224
225     cargaTotal += cargaPressao;
226
227     // 3. Soma pesos de trechos abaixo da profundidade
228     for (const auto& trecho : trechos) {
229         double z_i = trecho->ProfundidadeInicial();
230         double z_f = trecho->ProfundidadeFinal();
231         double pesoUnit = trecho->PesoUnidade();
232
233         // Caso 1: trecho totalmente abaixo da profundidade
234         if (z_i >= profundidade) {
235             double L_total = z_f - z_i;

```

```

233         double cargaPeso = L_total * pesoUnit;
234         cargaTotal += cargaPeso;
235     }
236
237     // Caso 2: profundidade dentro do trecho
238     else if (profundidade > z_i && profundidade < z_f) {
239         double L_parcial = z_f - profundidade;
240         double cargaPeso = L_parcial * pesoUnit;
241         cargaTotal += cargaPeso;
242     }
243
244 }
245
246 // 4. considerando as Cargas por efeito pist o
247
248 if (inicio != false){
249     cargaTotal += VariacaoCargaDevidoCrossover(profundidade ,
250             PressaoSuperficieFim());
251 }
252 else {
253     cargaTotal += VariacaoCargaDevidoCrossover(profundidade );
254 }
255
256 return cargaTotal;
257
258 double CObjetoPoco::DeltaLTemperatura(double profundidade) const {
259     double deltaLTotal = 0.0;
260
261     for (const auto& trecho : trechos) {
262         double z_i = trecho->ProfundidadeInicial();
263         double z_f = trecho->ProfundidadeFinal();
264         double ct = trecho->CoeficienteExpancaoTermica();
265
266         // Ignora trechos mais profundos que a profundidade
267         // analisada
268         if (z_i >= profundidade)
269             continue;
270
271         double tMedioInicial = (
272             TemperaturaNoPonto(z_i ,
273             TemperaturaTopoInicial() ,

```

```

272                                     TemperaturaFundoInicial() +
273                                     TemperaturaNoPonto(z_f,
274                                         TemperaturaTopoInicial(),
275                                         TemperaturaFundoInicial())
276                                         ) / 2.0;

277
278     double tMedioFinal = (
279         TemperaturaNoPonto(z_i,
280             TemperaturaTopoFinal(),
281             TemperaturaFundoFinal()) +
282             TemperaturaNoPonto(z_f,
283                 TemperaturaTopoFinal(),
284                 TemperaturaFundoFinal())
285             ) / 2.0;

286
287     double deltaT = tMedioFinal - tMedioInicial;
288     double L = 0.0;

289     // Profundidade dentro do trecho      considera parte de zi
290     // ate profundidade
291     if (profundidade > z_i && profundidade < z_f) {
292         L = profundidade - z_i;
293     }
294
295     // Trecho completamente acima da profundidade      usa L
296     // total
297     else if (z_f <= profundidade) {
298         L = z_f - z_i;
299     }
300
301     deltaLTotal += ct * L * deltaT;
302 }

303     return deltaLTotal;
304 }

305
306     double CObjetoPoco::TemperaturaNoPonto(double profundidade, double
307 T_topo, double T_Fundo) const {
308
309         double inclinacao = (T_Fundo - T_topo) / ProfundidadeOcupada();

```

```

304     double temperatura = T_topo + inclinacao * profundidade;
305     return temperatura;
306 }
308
309 double CObjetoPoco::VariacaoCargaDevidoCrossover(double
310     profundidade, double pressaoCabecaPoco) const {
311     double pi = 3.141592653589793;
312     double variacaoTotal = 0.0;
313     double variacaoCarga;
314
315     if (trechos.size() < 2)
316         return 0.0;
317
318     // Varre todas as interfaces abaixo da profundidade fornecida
319     for (size_t i = 1; i < trechos.size(); ++i) {
320         const auto& trechoAcima = trechos[i - 1];
321         const auto& trechoAbaixo = trechos[i];
322
323         double z_interface = trechoAbaixo->ProfundidadeInicial();
324
325         if ((z_interface - profundidade) > 1e-6) {
326             double ID_acima = trechoAcima->DiametroInterno();
327             double ID_abaixo = trechoAbaixo->DiametroInterno();
328             double OD_acima = trechoAcima->DiametroExterno();
329             double OD_abaixo = trechoAbaixo->DiametroExterno();
330
331             if (std::abs(ID_acima - ID_abaixo) < 1e-6 &&
332                 std::abs(OD_acima - OD_abaixo) < 1e-6) {
333                 continue;
334             }
335
336             // Diferenca de area: sempre acima - abaiixo
337             double Ai = (pi / 4.0) * (ID_acima * ID_acima -
338                                         ID_abaixo * ID_abaixo);
339             double Ao = (pi / 4.0) * (OD_acima * OD_acima -
340                                         OD_abaixo * OD_abaixo);
341             double pressao = PressaoHidroestaticaNoPonto(
342                 z_interface);
343
344             if (pressaoCabecaPoco != -1){
345                 variacaoCarga = ((pressao + pressaoCabecaPoco -

```

```

            PressaoSuperficie()) * Ai) - ((pressao -
            PressaoSuperficie()) * Ao);
        }
        else{
            variacaoCarga = pressao * (Ai - Ao);
        }
    }
    variacaoTotal += variacaoCarga;
}
}

return variacaoTotal;
}

#include <QDebug>

double CObjetoPoco::VariacaoCargaEfeitoPistao(double profundidade,
    double ID, double OD) const {
    const double pi = 3.141592653589793;

    // rea da parede interna do tubo
    double Ain = (pi / 4.0) * (OD * OD - ID * ID);

    double Aout = Ain;

    // Press es internas
    double Pin_inicio = PressaoHidroestaticaNoPonto(profundidade) +
        PressaoSuperficie();
    double Pin_fim = PressaoHidroestaticaNoPonto(profundidade) +
        PressaoSuperficieFim();
    double deltaPin = Pin_fim - Pin_inicio;

    // Press es externas
    double Pout_inicio;
    double Pout_fim;

    if (Packer() == true) {
        Pout_inicio = PressaoHidroestaticaNoPonto(PressaoSuperficie
            ());
        Pout_fim = PressaoHidroestaticaNoPonto(PressaoSuperficie())
            ;
    } else {

```

```

377     Pout_inicio = PressaoHidroestaticaNoPonto(PressaoSuperficie
378         ());
379     Pout_fim = PressaoHidroestaticaNoPonto(PressaoSuperficieFim
380         ());
381 }
382
383     double deltaPout = Pout_fim - Pout_inicio;
384
385     // Resultado final
386     if (Packer() == true) {
387         return -deltaPin * Ain - deltaPout * Aout;
388     } else {
389         return deltaPin * Ain - deltaPout * Aout;
390     }
391 }
392 double CObjetoPoco::DeltaLPistaoPacker(double profundidade, double
393     pressaoCabecaPoco) const {
394     double pi = 3.141592653589793;
395     double deltaL_total = 0.0;
396
397     if (trechos.empty() || pressaoCabecaPoco < 0.0)
398         return 0.0;
399
400     for (size_t i = 0; i < trechos.size(); ++i) {
401         const auto& trecho = trechos[i];
402         double z_i = trecho->ProfundidadeInicial();
403         double z_f = trecho->ProfundidadeFinal();
404         double OD = trecho->DiametroExterno();
405         double ID = trecho->DiametroInterno();
406         double E = trecho->ModuloElasticidade();
407
408         double area_anular = (pi / 4.0) * (OD * OD - ID * ID);
409
410         if (E <= 0.0 || area_anular <= 0.0)
411             continue;
412
413         double L = 0.0;
414
415         if (profundidade > z_i && profundidade < z_f) {
416             // trecho parcialmente acima da profundidade

```

```

416         L = profundidade - z_i;
417     } else if (z_f <= profundidade) {
418         // trecho totalmente acima da profundidade
419         L = z_f - z_i;
420     } else {
421         continue; // trecho abaixo da profundidade, ignora
422     }
423
424     double CargaPistao = VariacaoCargaEfeitoPistao(z_i, ID,
425             DiametroPoco());
426     double deltaL_trecho = (CargaPistao * (L)) / (E *
427             area_anular); // o x12      uma unidade de conversao de
428             ft para in
429     deltaL_total += deltaL_trecho;
430 }
431
432
433 double CObjetoPoco::DeltaLEfeitoBalão(double profundidade) const {
434     double pi = 3.141592653589793;
435     double deltaL_total = 0.0;
436
437     for (size_t i = 0; i < trechos.size(); ++i) {
438         const auto& trecho = trechos[i];
439         double z_i = trecho->ProfundidadeInicial();
440         double z_f = trecho->ProfundidadeFinal();
441         double OD = trecho->DiametroExterno();
442         double ID = trecho->DiametroInterno();
443         double E = trecho->ModuloElasticidade();
444         double v = trecho->CoeficientePoisson();
445
446         double area_anular = (pi / 4.0) * (OD * OD - ID * ID);
447
448         if (E <= 0.0 || area_anular <= 0.0)
449             continue;
450
451         double L = 0.0;
452
453         if (profundidade > z_i && profundidade < z_f) {
454             // trecho parcialmente acima da profundidade

```

```

455         L = profundidade - z_i;
456     } else if (z_f <= profundidade) {
457         // trecho totalmente acima da profundidade
458         L = z_f - z_i;
459     } else {
460         continue; // trecho abaixo da profundidade, ignora
461     }
462
463     double CargaPistao = VariacaoCargaEfeitoPistao(z_i, 0, ID);
464     double deltaL_trecho = 2 * v * (CargaPistao * (L)) / (E *
465         area_anular); // o x12 uma unidade de conversao de
466         ft para in
467
468     deltaL_total += deltaL_trecho;
469 }
470
471
472 double CObjetoPoco::DeltaLPistaoCrossover(double profundidade,
473     double pressaoCabecaPoco) const {
474     double pi = 3.141592653589793;
475     double deltaL_total = 0.0;
476
477     if (trechos.size() < 2)
478         return 0.0;
479
480     for (size_t i = 1; i < trechos.size(); ++i) {
481         const auto& trechoAcima = trechos[i - 1];
482         const auto& trechoAbaixo = trechos[i];
483
484         double z_interface = trechoAbaixo->ProfundidadeInicial();
485         if (z_interface < profundidade)
486             continue;
487
488         double OD_acima = trechoAcima->DiametroExterno();
489         double ID_acima = trechoAcima->DiametroInterno();
490         double OD_abai xo = trechoAbaixo->DiametroExterno();
491
492         if (std::abs(OD_acima - OD_abai xo) < 1e-6)
493             continue;

```

```

494     double cargaPistao = VariacaoCargaEfeitoPistao(z_interface
495         - 100, 1, 1);
496     double L = z_interface;
497
498     double E1 = trechoAcima->ModuloElasticidade();
499     double E2 = trechoAbaixo->ModuloElasticidade();
500     double E_medio = (E1 + E2) / 2.0;
501
502     double area_secao = (pi / 4.0) * (OD_acima * OD_acima -
503         ID_acima * ID_acima);
504
505     if (E_medio <= 0.0 || area_secao <= 0.0)
506         continue;
507
508     deltaL = (cargaPistao * L) / (E_medio * area_secao);
509     deltaL_total += deltaL;
510 }
511 }
512
513 double CObjetoPoco::DeltaLForcaRestauradora(double profundidade,
514     double pressaoCabecaPoco) const{
515     double deltaLTemperatura = DeltaLTemperatura(profundidade);
516     double deltaLBalao = DeltaLEfeitoBalao(profundidade);
517     double deltaLPacker = DeltaLPistaoPacker(profundidade,
518         pressaoCabecaPoco);
519     double deltaLCrossover = DeltaLPistaoCrossover(profundidade,
520         pressaoCabecaPoco);
521
522     return deltaLTemperatura + deltaLBalao + deltaLPacker +
523         deltaLCrossover;
524 }
525
526 double CObjetoPoco::CargaInjecao(double profundidade) const {
527     double numerador = 0.0;
528     double denominador = 0.0;
529     const double pi = 3.141592653589793;
530
531     for (const auto& trecho : trechos) {

```

```

530     double z_i = trecho->ProfundidadeInicial();
531     double z_f = trecho->ProfundidadeFinal();
532
533     // Pular trechos que estao acima da profundidade desejada
534     if (z_i > profundidade)
535         continue;
536
537     // Determinar quanto do trecho esta dentro da profundidade
538     // desejada
539     double limiteSuperior = std::min(z_f, profundidade);
540     double comprimentoUtil = limiteSuperior - z_i;
541
542     if (comprimentoUtil <= 0.0)
543         continue;
544
545     double E = trecho->ModuloElasticidade(); // modulo de
546     elasticidade
547     double F = DeltaLForcaRestauradora((double)profundidade,
548                                         PressaoSuperficieFim()); // forca restauradora da
549                                         classe
550     double OD = trecho->DiametroExterno(); // diametro
551     interno para calcular area
552     double ID = trecho->DiametroInterno(); // diametro
553     interno para calcular area
554
555     // Area da secao circular interna (em m , se diametro
556     // estiver em metros)
557     double area = (pi / 4.0) * (OD * OD - ID * ID);
558
559     numerador += E * F;
560     denominador += comprimentoUtil / area;
561 }
562
563     // Evita divisao por zero
564     if (denominador == 0.0)
565         return 0.0;
566
567     return - numerador / denominador;
568 }
```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.8 o arquivo de cabeçalho da classe CFluido.

Listing 7.8: Arquivo de implementação da classe CFluido

```
1 #ifndef CFLUIDO_H
2 #define CFLUIDO_H
3
4 #include <string>
5
6 /*
7 Classe que representa um fluido qualquer do sistema
8 Aqui sao armazenados nome, densidade e viscosidade
9 Essas informacoes sao usadas nos calculos de pressao, perda de
   carga, etc
10 */
11
12 class CFluido {
13 protected:
14     std::string nomeFluido;           // nome do fluido (ex: oleo ou
                                         agua)
15     double densidadeFluido = 0.0;    // densidade em lbm/gal
16     double viscosidadeFluido = 0.0; // viscosidade em cP
17
18 public:
19     CFluido() {}
20     ~CFluido() {}
21
22     // Construtor para inicializar com todos os dados
23     CFluido(std::string nome, double densidade, double viscosidade)
24         : nomeFluido(nome), densidadeFluido(densidade),
25           viscosidadeFluido(viscosidade) {}
26
27     // getters
28     std::string Nome() const { return nomeFluido; }
29     double Densidade() const { return densidadeFluido; }
30     double Viscosidade() const { return viscosidadeFluido; }
31
32     // setters
33     void Nome(const std::string& novoNome) { nomeFluido = novoNome;
34     }
35     void Densidade(double novaDensidade) { densidadeFluido =
36         novaDensidade; }
37     void Viscosidade(double novaViscosidade) { viscosidadeFluido =
38         novaViscosidade; }
39 };
40 }
```

```
37 #endif
```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.9 a implementação da classe CFluido.

Listing 7.9: Arquivo de implementação da classe CFluido

```
1 // Arquivo fonte da classe CFluido
2 // Atualmente todos os metodos sao implementados no proprio
   cabecalho (.h)
3 // Este arquivo existe apenas por organizacao, podendo ser removido
   se desejado
4
5 #include "CFluido.h"
```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.10 o arquivo de cabeçalho da classe CModeloReologico.

Listing 7.10: Arquivo de implementação da classe CModeloReologico

```
1 ifndef CMODELOREOLOGICO_H
2 define CMODELOREOLOGICO_H
3
4 include <string>
5 include "CObjetoPoco.h"
6
7 /*
8 Classe base abstrata para os modelos reologicos
9 Define as propriedades e metodos que devem ser implementados pelos
   modelos concretos
10 Serve como interface para modelos como newtoniano, Bingham e lei da
    potencia
11 */
12
13 class CModeloReologico {
14
15 protected:
16     // Propriedades relacionadas ao escoamento e calculos
17     double fatorFriccaoPoco = 0.0;
18     double fatorFriccaoAnular = 0.0;
19     double reynoldsPoco = 0.0;
20     double reynoldsAnular = 0.0;
21     double vMediaPoco = 0.0;
22     double vMediaAnular = 0.0;
23
```

```

24     // Tipo de fluxo (laminar ou turbulento)
25     std::string fluxoPoco;
26     std::string fluxoAnular;
27
28     // Objeto que contem as propriedades do poco
29     CObjetoPoco* poco;
30
31 public:
32     // Construtores
33     CModeloReologico() {}
34     virtual ~CModeloReologico() {}
35     CModeloReologico(CObjetoPoco* poco) : poco(poco) {}

36
37     // Getters
38     double FatorFriccaoPoco() const { return fatorFriccaoPoco; }
39     double FatorFriccaoAnular() const { return fatorFriccaoAnular; }
40     double ReynoldsPoco() const { return reynoldsPoco; }
41     double ReynoldsAnular() const { return reynoldsAnular; }
42     double VMediaPoco() const { return vMediaPoco; }
43     double VMediaAnular() const { return vMediaAnular; }
44     std::string FluxoPoco() const { return fluxoPoco; }
45     std::string FluxoAnular() const { return fluxoAnular; }

46
47     // M todos para determinar fatores e velocidades
48     double DeterminarFatorFriccao(double re, double n);
49     double DeterminarReynoldsPoco();
50     double DeterminarReynoldsPoco(double viscosidade);
51     double DeterminarReynoldsAnular();
52     double DeterminarReynoldsAnular(double viscosidade);
53     double DeterminarVelocidadeMediaPoco();
54     double DeterminarVelocidadeMediaAnular();

55
56     // M todos puros (devem ser implementados pelas classes filhas
57     )
58     virtual std::string DeterminarFluxoPoco() = 0;
59     virtual std::string DeterminarFluxoAnular() = 0;
60     virtual double CalcularPerdaPorFriccaoPoco() = 0;
61     virtual double CalcularPerdaPorFriccaoAnular() = 0;
62
63 #endif // CMODELOREOLOGICO_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.11 a implementação da classe CModeloReologico.

Listing 7.11: Arquivo de implementação da classe CModeloReologico

```
1 #include <iostream>
2 #include <cmath>
3 #include <iomanip>
4
5 #include "CModeloReologico.h"
6
7 // Calcula o numero de Reynolds no poto usando a viscosidade total
8 // do fluido
9 double CModeloReologico::DeterminarReynoldsPoco() {
10     reynoldsPoco = (928 * poco->DensidadeEfetivaTotal() *
11                     vMediaPoco * poco->DiametroRevestimentoID()) /
12                     poco->ViscosidadeEfetivaTotal();
13     return reynoldsPoco;
14 }
15 // Mesmo calculo, mas permitindo passar a viscosidade como
16 // parametro
17 double CModeloReologico::DeterminarReynoldsPoco(double viscosidade)
18 {
19     reynoldsPoco = (928 * poco->DensidadeEfetivaTotal() *
20                     vMediaPoco * poco->DiametroRevestimentoID()) /
21                     viscosidade;
22     return reynoldsPoco;
23 }
24
25 // Calcula o numero de Reynolds no espaco anular
26 double CModeloReologico::DeterminarReynoldsAnular() {
27     reynoldsAnular = (757 * poco->DensidadeEfetivaTotal() *
28                     vMediaAnular *
29                     (poco->DiametroPoco() - poco->
30                     DiametroRevestimentoOD()))) /
31                     poco->ViscosidadeEfetivaTotal();
32     return reynoldsAnular;
33 }
34
35 // Mesmo calculo para o anular, com viscosidade recebida
36 // externamente
37 double CModeloReologico::DeterminarReynoldsAnular(double
```

```

    viscosidade) {
32     reynoldsAnular = (757 * poco->DensidadeEfetivaTotal() *
33         vMediaAnular *
34             (poco->DiametroPoco() - poco->
35                 DiametroRevestimentoOD())) /
36                 viscosidade;
37
38 // Calcula a velocidade m dia do fluido no interior da coluna (
39 poco)
40 double CModeloReologico::DeterminarVelocidadeMediaPoco() {
41     vMediaPoco = poco->Vazao() / (2.448 * std::pow(poco->
42         DiametroRevestimentoID(), 2));
43
44     return vMediaPoco;
45 }
46
47 // Calcula a velocidade m dia no espaco anular entre a coluna e o
48 revestimento
49 double CModeloReologico::DeterminarVelocidadeMediaAnular() {
50     vMediaAnular = poco->Vazao() /
51         (2.448 * (std::pow(poco->DiametroPoco(), 2) -
52             std::pow(poco->DiametroRevestimentoOD(), 2)))
53             ;
54
55     return vMediaAnular;
56 }

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.12 o arquivo de cabeçalho da classe CModeloNewtoniano.

Listing 7.12: Arquivo de implementação da classe CModeloNewtoniano

```

1 #ifndef CMODELONEWTONIANO_H
2 #define CMODELONEWTONIANO_H
3
4 #include "CModeloReologico.h"
5
6 /*
7 Classe que representa o modelo reológico newtoniano
8 Nesse caso, a viscosidade é constante e independe da taxa de
9 deformação
10 A classe herda de CModeloReologico e implementa os métodos
11 específicos para esse tipo de fluido
12 */

```

```

11
12 class CModeloNewtoniano : public CModeloReologico {
13
14 public:
15     // Construtores
16     CModeloNewtoniano() {}
17     ~CModeloNewtoniano() {}

18
19     // Construtor que recebe o objeto do poco
20     CModeloNewtoniano(CObjetoPoco* poco) : CModeloReologico(poco) {
21         DeterminarFluxoPoco();
22         DeterminarFluxoAnular();
23     }

24
25     // Metodos obrigatorios sobrescritos do modelo base
26     std::string DeterminarFluxoPoco() override;
27     std::string DeterminarFluxoAnular() override;
28     double CalcularPerdaPorFriccaoPoco() override;
29     double CalcularPerdaPorFriccaoAnular() override;
30 };

31
32 #endif // CMODELONEWTONIANO_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.13 a implementação da classe CModeloNewtoniano.

Listing 7.13: Arquivo de implementação da classe CModeloNewtoniano

```

1 #include "CModeloNewtoniano.h"
2 #include <cmath>
3
4 // Determina o tipo de fluxo no poco com base no numero de Reynolds
5 std::string CModeloNewtoniano::DeterminarFluxoPoco() {
6     DeterminarVelocidadeMediaPoco();
7     DeterminarReynoldsPoco();
8
9     // Valor limite de 2100 usado para separar regime laminar e
10    // turbulento
11    fluxoPoco = (reynoldsPoco <= 2100) ? "Laminar" : "Turbulento";
12    return fluxoPoco;
13}
14 // Determina o tipo de fluxo no espaco anular com base no numero de
15 // Reynolds

```

```

15 std::string CModeloNewtoniano::DeterminarFluxoAnular() {
16     DeterminarVelocidadeMediaAnular();
17     DeterminarReynoldsAnular();
18
19     fluxoAnular = (reynoldsAnular <= 2100) ? "Laminar" : "
20         Turbulento";
21     return fluxoAnular;
22 }
23 // Calcula a perda de carga por friccao no poco para regime laminar
24 // ou turbulento
24 double CModeloNewtoniano::CalcularPerdaPorFriccaoPoco() {
25     if (fluxoPoco.empty()) {
26         DeterminarFluxoPoco();
27     }
28
29     if (fluxoPoco == "Laminar") {
30         return (poco->ViscosidadeEfetivaTotal() * vMediaPoco) /
31             (1500 * std::pow(poco->DiametroRevestimentoID(), 2))
32             ;
33 } else {
34     return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std
35         ::pow(vMediaPoco, 1.75) *
36         std::pow(poco->ViscosidadeEfetivaTotal(), 0.25)) /
37             (1800 * std::pow(poco->DiametroRevestimentoID(),
38                 1.25));
39 }
40 }
41
42 // Calcula a perda de carga por friccao no espaco anular
43 double CModeloNewtoniano::CalcularPerdaPorFriccaoAnular() {
44     if (fluxoAnular.empty()) {
45         DeterminarFluxoAnular();
46     }
47
48     double diametroAnular = poco->DiametroPoco() - poco->
49         DiametroRevestimentoOD();
50
51     if (fluxoAnular == "Laminar") {
52         return (poco->ViscosidadeEfetivaTotal() * vMediaAnular) /
53             (1000 * std::pow(diametroAnular, 2));
54 } else {

```

```

51         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std
52             ::pow(vMediaAnular, 1.75) *
53                 std::pow(poco->ViscosidadeEfetivaTotal(), 0.25)) /
54                 (1396 * std::pow(diametroAnular, 1.25));
55     }
56 }

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.14 o arquivo de cabeçalho da classe CModeloBingham.

Listing 7.14: Arquivo de implementação da classe CModeloBingham

```

1 #ifndef CMODELOBINGHAM_H
2 #define CMODELOBINGHAM_H
3
4 #include "CModeloReologico.h"
5
6 /*
7 Classe que representa o modelo reológico de Bingham
8 Esse modelo é usado para fluidos com um ponto de escoamento
9 definido
10 A implementação baseia-se na herança da classe CModeloReologico
11 */
12 class CModeloBingham : public CModeloReologico {
13
14 protected:
15     // Parâmetros específicos do modelo de Bingham
16     double viscosidadePlastica = 0.0;
17     double pontoDeEscoamento = 0.0;
18
19     // Valores críticos de Reynolds (poco e anular)
20     double reynoldsCriticoPoco = 0.0;
21     double reynoldsCriticoAnular = 0.0;
22
23     // Números de Hedström (relacionados ao tipo de escoamento)
24     double reynoldsHedstromPoco = 0.0;
25     double reynoldsHedstromAnular = 0.0;
26
27 public:
28     // Construtores
29     CModeloBingham() {}
30     ~CModeloBingham() {}
31

```

```

32     // Construtor com ponteiro para o objeto CObjetoPoco
33     CModeloBingham(CObjetoPoco* poco) : CModeloReologico(poco) {}
34
35     // Construtor completo com parametros do modelo
36     CModeloBingham(CObjetoPoco* poco, double viscosidadePlastica,
37                     double pontoDeEscoamento)
38         : CModeloReologico(poco),
39           viscosidadePlastica(viscosidadePlastica),
40           pontoDeEscoamento(pontoDeEscoamento)
41     {
42         DeterminarFluxoPoco();
43         DeterminarFluxoAnular();
44     }
45
46     // Getters
47     double ViscosidadePlastica() const { return viscosidadePlastica
48         ; }
49     double PontoDeEscoamento() const { return pontoDeEscoamento; }
50     double ReynoldsCriticoPoco() const { return reynoldsCriticoPoco
51         ; }
52     double ReynoldsCriticoAnular() const { return
53         reynoldsCriticoAnular; }
54     double ReynoldsHedstronPoco() const { return
55         reynoldsHedstronPoco; }
56     double ReynoldsHedstronAnular() const { return
57         reynoldsHedstronAnular; }
58
59     // Setters
60     void ViscosidadePlastica(double valor) { viscosidadePlastica =
61         valor; }
62     void PontoDeEscoamento(double valor) { pontoDeEscoamento =
63         valor; }
64
65     // Metodos especificos do modelo de Bingham
66     double DeterminarReynoldsCritico(double hedstron);
67     double DeterminarReynoldsHedstronPoco();
68     double DeterminarReynoldsHedstronAnular();
69     std::string DeterminarFluxoPoco() override;
70     std::string DeterminarFluxoAnular() override;
71     double CalcularPerdaPorFriccaoPoco() override;
72     double CalcularPerdaPorFriccaoAnular() override;
73 };

```

```
66  
67 #endif // CMODELOBINGHAM_H
```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.15 a implementação da classe CModeloBingham.

Listing 7.15: Arquivo de implementação da classe CModeloBingham

```
1 #include "CModeloBingham.h"  
2 #include <cmath>  
3 #include <QApplication>  
4 #include <QFileDialog>  
5 #include <QString>  
6 #include <QMessageBox>  
7  
8 // Determina o valor de Reynolds critico com base no numero de  
// Hedstron  
9 // Utiliza metodo numerico de Newton-Raphson para resolver a  
// equacao implicita  
10 double CModeloBingham::DeterminarReynoldsCritico(double hedstron) {  
11     // f(x) = ((x / (1 - x)^3) * 16800) - Hedstron  
12     auto func = [hedstron](double x) {  
13         return ((x / std::pow(1 - x, 3)) * 16800) - hedstron;  
14     };  
15  
16     // Derivada de f(x)  
17     auto derivadaFunc = [] (double x) {  
18         return (16800 * (1 + x) / std::pow(1 - x, 4));  
19     };  
20  
21     // Metodo de Newton-Raphson para aproximar a raiz  
22     auto newtonRaphson = [&] (double x) {  
23         for (int i = 0; i < 10000; ++i) {  
24             x = x - func(x) / derivadaFunc(x);  
25             if (fabs(func(x)) < 1e-2) break;  
26         }  
27         return x;  
28     };  
29  
30     // Chute inicial  
31     double y = newtonRaphson(0.99);  
32  
33     // Retorna o Reynolds critico com base em y encontrado  
34     return ((1 - ((4.0 / 3.0) * y) + ((1.0 / 3.0) * std::pow(y, 4)))
```

```

35 }
36
37 // Calcula o numero de Hedstron para o escoamento no poco
38 double CModeloBingham::DeterminarReynoldsHedstronPoco() {
39     reynoldsHedstronPoco =
40         (37100 * poco->DensidadeEfetivaTotal() * pontoDeEscoamento
41             *
42             std::pow(poco->DiametroRevestimentoID(), 2)) /
43             std::pow(viscosidadePlastica, 2);
44
45 }
46
47 // Calcula o numero de Hedstron para o escoamento no espaco anular
48 double CModeloBingham::DeterminarReynoldsHedstronAnular() {
49     double diametroAnular = poco->DiametroPoco() - poco->
50         DiametroRevestimentoOD();
51
52     reynoldsHedstronAnular =
53         (24700 * poco->DensidadeEfetivaTotal() * pontoDeEscoamento
54             *
55             std::pow(diametroAnular, 2)) /
56             std::pow(viscosidadePlastica, 2);
57
58
59 // Determina o tipo de fluxo no poco (laminar ou turbulento)
60 std::string CModeloBingham::DeterminarFluxoPoco() {
61     DeterminarVelocidadeMediaPoco();
62     DeterminarReynoldsPoco(viscosidadePlastica);
63     DeterminarReynoldsHedstronPoco();
64
65     reynoldsCriticoPoco = DeterminarReynoldsCritico(
66         reynoldsHedstronPoco);
67
68     fluxoPoco = (reynoldsPoco <= reynoldsCriticoPoco) ? "Laminar" :
69         "Turbulento";
70
71     return fluxoPoco;
72 }
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
747
748
749
749
750
751
752
753
754
755
756
757
757
758
759
759
760
761
762
763
764
765
766
767
767
768
769
769
770
771
772
773
774
775
776
777
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
827
828
829
829
830
831
832
833
834
835
836
837
837
838
839
839
840
841
842
843
844
845
846
847
847
848
849
849
850
851
852
853
854
855
856
857
857
858
859
859
860
861
862
863
864
865
866
867
867
868
869
869
870
871
872
873
874
875
876
877
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
907
908
909
909
910
911
912
913
914
915
915
916
917
917
918
919
919
920
921
922
923
924
925
926
926
927
928
928
929
930
931
932
933
934
935
936
937
937
938
939
939
940
941
942
943
944
945
945
946
947
947
948
949
949
950
951
952
953
954
955
956
957
957
958
959
959
960
961
962
963
964
965
966
966
967
968
968
969
969
970
971
972
973
974
975
976
976
977
978
978
979
980
981
982
983
984
985
986
986
987
988
988
989
989
990
991
992
993
994
995
995
996
997
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1005
1006
1007
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1015
1016
1017
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1025
1026
1027
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1035
1036
1037
1037
1038
1039
1039
1040
1041
1042
1043
1044
1044
1045
1046
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1054
1055
1056
1056
1057
1058
1058
1059
1060
1061
1062
1063
1064
1064
1065
1066
1066
1067
1068
1068
1069
1070
1071
1072
1073
1073
1074
1075
1075
1076
1077
1077
1078
1079
1079
1080
1081
1082
1083
1084
1084
1085
1086
1086
1087
1088
1088
1089
1090
1091
1092
1092
1093
1094
1094
1095
1096
1096
1097
1098
1098
1099
1100
1101
1102
1102
1103
1104
1104
1105
1106
1106
1107
1108
1108
1109
1110
1111
1112
1112
1113
1114
1114
1115
1116
1116
1117
1118
1118
1119
1120
1120
1121
1122
1122
1123
1124
1124
1125
1126
1126
1127
1128
1128
1129
1130
1130
1131
1132
1132
1133
1134
1134
1135
1136
1136
1137
1138
1138
1139
1140
1140
1141
1142
1142
1143
1144
1144
1145
1146
1146
1147
1148
1148
1149
1150
1150
1151
1152
1152
1153
1154
1154
1155
1156
1156
1157
1158
1158
1159
1160
1160
1161
1162
1162
1163
1164
1164
1165
1166
1166
1167
1168
1168
1169
1170
1170
1171
1172
1172
1173
1174
1174
1175
1176
1176
1177
1178
1178
1179
1180
1180
1181
1182
1182
1183
1184
1184
1185
1186
1186
1187
1188
1188
1189
1190
1190
1191
1192
1192
1193
1194
1194
1195
1196
1196
1197
1198
1198
1199
1200
1200
1201
1202
1202
1203
1204
1204
1205
1206
1206
1207
1208
1208
1209
1210
1210
1211
1212
1212
1213
1214
1214
1215
1216
1216
1217
1218
1218
1219
1220
1220
1221
1222
1222
1223
1224
1224
1225
1226
1226
1227
1228
1228
1229
1230
1230
1231
1232
1232
1233
1234
1234
1235
1236
1236
1237
1238
1238
1239
1240
1240
1241
1242
1242
1243
1244
1244
1245
1246
1246
1247
1248
1248
1249
1250
1250
1251
1252
1252
1253
1254
1254
1255
1256
1256
1257
1258
1258
1259
1260
1260
1261
1262
1262
1263
1264
1264
1265
1266
1266
1267
1268
1268
1269
1270
1270
1271
1272
1272
1273
1274
1274
1275
1276
1276
1277
1278
1278
1279
1280
1280
1281
1282
1282
1283
1284
1284
1285
1286
1286
1287
1288
1288
1289
1290
1290
1291
1292
1292
1293
1294
1294
1295
1296
1296
1297
1298
1298
1299
1300
1300
1301
1302
1302
1303
1304
1304
1305
1306
1306
1307
1308
1308
1309
1310
1310
1311
1312
1312
1313
1314
1314
1315
1316
1316
1317
1318
1318
1319
1320
1320
1321
1322
1322
1323
1324
1324
1325
1326
1326
1327
1328
1328
1329
1330
1330
1331
1332
1332
1333
1334
1334
1335
1336
1336
1337
1338
1338
1339
1340
1340
1341
1342
1342
1343
1344
1344
1345
1346
1346
1347
1348
1348
1349
1350
1350
1351
1352
1352
1353
1354
1354
1355
1356
1356
1357
1358
1358
1359
1360
1360
1361
1362
1362
1363
1364
1364
1365
1366
1366
1367
1368
1368
1369
1370
1370
1371
1372
1372
1373
1374
1374
1375
1376
1376
1377
1378
1378
1379
1380
1380
1381
1382
1382
1383
1384
1384
1385
1386
1386
1387
1388
1388
1389
1390
1390
1391
1392
1392
1393
1394
1394
1395
1396
1396
1397
1398
1398
1399
1400
1400
1401
1402
1402
1403
1404
1404
1405
1406
1406
1407
1408
1408
1409
1410
1410
1411
1412
1412
1413
1414
1414
1415
1416
1416
1417
1418
1418
1419
1420
1420
1421
1422
1422
1423
1424
1424
1425
1426
1426
1427
1428
1428
1429
1430
1430
1431
1432
1432
1433
1434
1434
1435
1436
1436
1437
1438
1438
1439
1440
1440
1441
1442
1442
1443
1444
1444
1445
1446
1446
1447
1448
1448
1449
1450
1450
1451
1452
1452
1453
1454
1454
1455
1456
1456
1457
1458
1458
1459
1460
1460
1461
1462
1462
1463
1464
1464
1465
1466
1466
1467
1468
1468
1469
1470
1470
1471
1472
1472
1473
1474
1474
1475
1476
1476
1477
1478
1478
1479
1480
1480
1481
1482
1482
1483
1484
1484
1485
1486
1486
1487
1488
1488
1489
1490
1490
1491
1492
1492
1493
1494
1494
1495
1496
1496
1497
1498
1498
1499
1500
1500
1501
1502
1502
1503
1504
1504
1505
1506
1506
1507
1508
1508
1509
1510
1510
1511
1512
1512
1513
1514
1514
1515
1516
1516
1517
1518
1518
1519
1520
1520
1521
1522
1522
1523
1524
1524
1525
1526
1526
1527
1528
1528
1529
1530
1530
1531
1532
1532
1533
1534
1534
1535
1536
1536
1537
1538
1538
1539
1540
1540
1541
1542
1542
1543
1544
1544
1545
1546
1546
1547
1548
1548
1549
1550
1550
1551
1552
1552
1553
1554
1554
1555
1556
1556
1557
1558
1558
1559
1560
1560
1561
1562
1562
1563
1564
1564
1565
1566
1566
1567
1568
1568
1569
1570
1570
1571
1572
1572
1573
1574
1574
1575
1576
1576
1577
1578
1578
1579
1580
1580
1581
1582
1582
1583
1584
1584
1585
1586
1586
1587
1588
1588
1589
1590
1590
1591
1592
1592
1593
1594
1594
1595
1596
1596
1597
1598
1598
1599
1600
1600
1601
1602
1602
1603
1604
1604
1605
1606
1606
1607
1608
1608
1609
1610
1610
1611
1612
1612
1613
1614
1614
1615
1616
1616
1617
1618
1618
1619
1620
1620
1621
1622
1622
1623
1624
1624
1625
1626
1626
1627
1628
1628
1629
1630
1630
1631
1632
1632
1633
1634
1634
1635
1636
1636
1637
1638
1638
1639
1640
1640
1641
1642
1642
1643
1644
1644
1645
1646
1646
1647
1648
1648
1649
1650
1650
1651
1652
1652
1653
1654
1654
1655
1656
1656
1657
1658
1658
1659
1660
1660
1661
1662
1662
1663
1664
1664
1665
1666
1666
1667
1668
1668
1669
1670
1670
1671
1672
1672
1673
1674
1674
1675
1676
1676
1677
1678
1678
1679
1680
1680
1681
1682
1682
1683
1684
1684
1685
1686
1686
1687
1688
1688
1689
1690
1690
1691
1692
1692
1693
1694
1694
1695
1696
1696
1697
1698
1698
1699
1700
1700
1701
1702
1702
1703
1704
1704
1705
1706
1706
1707
1708
1708
1709
1710
1710
1711
1712
1712
1713
1714
1714
1715
1716
1716
1717
1718
1718
1719
1720
1720
1721
1722
1722
1723
1724
1724
1725
1726
1726
1727
1728
1728
1729
1730
1730
1731
1732
1732
1733
1734
1734
1735
1736
1736
1737
1738
1738
1739
1740
1740
1741
1742
1742
1743
1744
1744
1745
1746
1746
1747
1748
1748
1749
1750
1750
1751
1752
1752
1753
1754
1754
1755
1756
1756
1757
1758
1758
1759
1760
1760
1761
1762
1762
1763
1764
1764
1765
1766
1766
1767
1768
1768
1769
1770
1770
1771
1772
1772
1773
1774
1774
1775
1776
1776
1777
1778
1778
1779
1780
1780
1781
1782
1782
1783
1784
1784
1785
1786
1786
1787
1788
1788
1789
1790
1790
1791
1792
1792
1793
1794
1794
1795
1796
1796
1797
1798
1798
1799
1800
1800
1801
1802
1802
1803
1804
1804
1805
1806
1806
1807
1808
1808
1809
1810
1810
1811
1812
1812
1813
1814
1814
1815
1816
1816
1817
1818
1818
1819
1820
1820
1821
1822
1822
1823
1824
1824
1825
1826
1826
1827
1828
1828
1829
1830
1830
1831
1832
1832
1833
1834
1834
1835
1836
1836
1837
1838
1838
1839
1840
1840
1841
1842
1842
1843
1844
1844
1845
1846
1846
1847
1848
1848
1849
1850
1850
1851
1852
1852
1853
1854
1854
1855
1856
1856
1857
1858
1858
1859
1860
1860
1861
1862
1862
1863
1864
1864
1865
1866
1866
1867
1868
1868
1869
1870
1870
1871
1872
1872
1873
1874
1874
1875
1876
1876
1877
1878
1878
1879
1880
1880
1881
1882
1882
1883
1884
1884
1885
1886
1886
1887
1888
1888
1889
1890
1890
1891
1892
1892
1893
1894
1894
1895
1896
1896
1897
1898
1898
1899
1900
1900
1901
1902
1902
1903
1904
1904
1905
1906
1906
1907
1908
1908
1909
1910
1910
1911
1912
1912
1913
1914
1914
1915
1916
1916
1917
1918
1918
1919
1920
1920
1921
1922
1922
1923
1924
1924
1925
1926
1926
1927
1928
1928
1929
1930
1930
1931
1932
1932
1933
1934
1934
1935
1936
1936
1937
1938
1938
1939
1940
1940
1941
1942
1942
1943
1944
1944
1945
1946
1946
1947
1948
1948
1949
1950
1950
1951
1952
1952
1953
1954
1954
1955
1956
1956
1957
1958
1958
1959
1960
1960
1961
1962
1962
1963
1964
1964
1965
1966
1966
1967
1968
1968
1969
1970
1970
1971
1972
1972
1973
1974
1974
1975
1976
1976
1977
1978
1978
1979
1980
1980
1981
1982
1982
1983
1984
1984
1985
1986
1986
1987
1988
1988
1989
1990
1990
1991
1992
1992
1993
1994
1994
1995
1996
1996
1997
1998
1998
1999
2000
2000
2001
2002
2002
2003
2004
2004
2005
2006
2006
2007
2008
2008
2009
2010
2010
2011
2012
2012
2013
2014
2014
2015
2016
2016
2017
2018
2018
2019
2020
2020
2021
2022
2022
2023
2024
2024
2025
2026
2026
2027
2028
2028
2029
2030
2030
2031
2032
2032
2033
2034
2034
2035
2036
2036
2037
2038
2038
2039
2040
2040
2041
2042
2042
2043
2044
2044
2045
2046
2046
2047
2048
2048
2049
2050
2050
2051
2052
2052
2053
2054
2054
2055
2056
2056
2057
2058
2058
2059
2060
2060
2061
2062
2062
2063
2064

```

```

71 // Determina o tipo de fluxo no espaco anular
72 std::string CModeloBingham::DeterminarFluxoAnular() {
73     DeterminarVelocidadeMediaAnular();
74     DeterminarReynoldsAnular(viscosidadePlastica);
75     DeterminarReynoldsHedstronAnular();
76
77     reynoldsCriticoAnular = DeterminarReynoldsCritico(
78         reynoldsHedstronAnular);
79
80     fluxoAnular = (reynoldsAnular <= reynoldsCriticoAnular) ? "
81         Laminar" : "Turbulento";
82
83     return fluxoAnular;
84 }
85
86 // Calcula a perda de carga por friccao no poco
87 double CModeloBingham::CalcularPerdaPorFriccaoPoco() {
88     if (fluxoPoco == "Laminar") {
89         return ((viscosidadePlastica * vMediaPoco) /
90                 (1500 * std::pow(poco->DiametroRevestimentoID(), 2)
91                  )) +
92                 (pontoDeEscoamento / (225 * poco->
93                     DiametroRevestimentoID())));
94     } else {
95         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std
96                 ::pow(vMediaPoco, 1.75) *
97                 std::pow(viscosidadePlastica, 0.25)) /
98                 (1800 * std::pow(poco->DiametroRevestimentoID(),
99                     1.25));
100    }
101 }
102
103 // Calcula a perda de carga por friccao no espaco anular
104 double CModeloBingham::CalcularPerdaPorFriccaoAnular() {
105     double diametroAnular = poco->DiametroPoco() - poco->
106         DiametroRevestimentoOD();
107
108     if (fluxoAnular == "Laminar") {
109         return ((viscosidadePlastica * vMediaAnular) /
110                 (1000 * std::pow(diametroAnular, 2))) +
111                 (pontoDeEscoamento / (200 * diametroAnular));
112     } else {
113         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std

```

```

        ::pow(vMediaAnular, 1.75) *
106      std::pow(viscosidadePlastica, 0.25)) /
107      (1396 * std::pow(diametroAnular, 1.25));
108  }
109 }

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.18 o arquivo de cabeçalho da classe CModeloPotencia.

Listing 7.16: Arquivo de implementação da classe CModeloPotencia

```

1 #ifndef CMODELOPOTENCIA_H
2 #define CMODELOPOTENCIA_H
3
4 #include "CModeloReologico.h"
5
6 /*
7 Classe que representa o modelo reológico da Lei da Potencia
8 Esse modelo é aplicado a fluidos pseudoplásticos ou dilatantes (n
9   diferente de 1)
10  A classe herda de CModeloReologico e implementa os métodos
11    específicos do modelo
12 */
13
14 class CModeloPotencia : public CModeloReologico {
15
16 protected:
17     // Parâmetros do modelo da potência
18     double indiceDeConsistencia;           // K
19     double indiceDeComportamento;          // n
20
21     // Reynolds crítico arbitrário (2100 como base de separação)
22     double reynoldsCriticoPoco;
23     double reynoldsCriticoAnular;
24
25 public:
26     // Construtores
27     CModeloPotencia() {}
28     ~CModeloPotencia() {}
29
30     // Construtor com inicialização do pôco e do índice de
31     // consistência
32     CModeloPotencia(CObjetoPoco* poco, double indiceDeConsistencia,
33                      double indiceDeComportamento)

```

```

30         : CModeloReologico(poco),
31         indiceDeConsistencia(indiceDeConsistencia),
32         indiceDeComportamento(indiceDeComportamento)
33
34     {
35         DeterminarFluxoPoco();
36         DeterminarFluxoAnular();
37         IndiceDeComportamento(indiceDeComportamento);
38     }
39
40     // Getters
41     double ReynoldsCriticoPoco() const { return reynoldsCriticoPoco
42         ; }
43     double ReynoldsCriticoAnular() const { return
44         reynoldsCriticoAnular; }
45     double IndiceDeConsistencia() const { return
46         indiceDeConsistencia; }
47     double IndiceDeComportamento() const { return
48         indiceDeComportamento; }
49     std::string FluxoPoco() const { return fluxoPoco; }
50     std::string FluxoAnular() const { return fluxoAnular; }
51
52     // Setters
53     void IndiceDeConsistencia(double valor) { indiceDeConsistencia
54         = valor; }
55     void IndiceDeComportamento(double valor) {
56         indiceDeComportamento = valor; }
57     void FluxoPoco(const std::string& fluxo) { fluxoPoco = fluxo; }
58     void FluxoAnular(const std::string& fluxo) { fluxoAnular =
59         fluxo; }
60
61     // Metodos especificos do modelo de potencia
62     double DeterminarFatorFriccao(double reynolds, double n);
63     double DeterminarReynoldsCritico(double reynolds);
64     double DeterminarReynoldsPoco();
65     double DeterminarReynoldsAnular();
66     std::string DeterminarFluxoPoco() override;
67     std::string DeterminarFluxoAnular() override;
68     double CalcularPerdaPorFriccaoPoco() override;
69     double CalcularPerdaPorFriccaoAnular() override;
70 };
71
72

```

```
65 #endif // CMODELOPOTENCIA_H
```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.17 a implementação da classe CModeloPotencia.

Listing 7.17: Arquivo de implementação da classe CModeloPotencia

```
1 #include "CModeloPotencia.h"
2 #include <cmath>
3 #include <stdexcept>
4
5 // Retorna o fator de atrito para fluido da lei da pot ncia com
6 // base em NRe e n
7 double CModeloPotencia::DeterminarFatorFriccao(double NRe, double n
8 ) {
9
10    // Coeficientes obtidos via regress o log-log com base nos
11    // dados reais do gr fico
12    const double n_vals[] = {1.0, 0.8, 0.6, 0.4, 0.2};
13    const double a_vals[] = {-0.23285, -0.25049, -0.24695,
14                           -0.20192, -0.30519};
15    const double b_vals[] = {-1.14079, -1.12562, -1.25944,
16                           -1.66278, -1.33815};
17
18    // Interpola o linear entre os dois pontos mais pr ximos
19    double a = 0.0, b = 0.0;
20    for (int i = 1; i < 5; ++i) {
21        if (n <= n_vals[i - 1] && n >= n_vals[i]) {
22            double t = (n - n_vals[i]) / (n_vals[i - 1] - n_vals[i
23                ]);
24            a = a_vals[i] + t * (a_vals[i - 1] - a_vals[i]);
25            b = b_vals[i] + t * (b_vals[i - 1] - b_vals[i]);
26            break;
27        }
28    }
29
30    double logF = a * std::log10(NRe) + b;
31    return std::pow(10.0, logF);
32}
33
34 // Funcao generica para Reynolds critico (nao esta sendo usada
35 // diretamente aqui)
36 double CModeloPotencia::DeterminarReynoldsCritico(double Reynolds)
37 {
```

```

30     return 183.13 * std::pow(Reynolds, 0.3185);
31 }
32
33 // Calcula o numero de Reynolds para o escoamento no poco
34 double CModeloPotencia::DeterminarReynoldsPoco() {
35     reynoldsPoco =
36         ((89100 * poco->DensidadeEfetivaTotal() * std::pow(
37             vMediaPoco, 2 - indiceDeComportamento)) /
38             indiceDeConsistencia) *
39             (std::pow((0.0416 * poco->DiametroRevestimentoID()) / (3 +
40                 (1 / indiceDeComportamento)), indiceDeComportamento));
41
42     reynoldsCriticoPoco = DeterminarReynoldsCritico(reynoldsPoco);
43 }
44
45 // Calcula o numero de Reynolds no espaco anular
46 double CModeloPotencia::DeterminarReynoldsAnular() {
47     reynoldsAnular =
48         ((109000 * poco->DensidadeEfetivaTotal() * std::pow(
49             vMediaAnular, 2 - indiceDeComportamento)) /
50             indiceDeConsistencia) *
51             (std::pow((0.0208 * (poco->DiametroPoco() - poco->
52                 DiametroRevestimentoID())) / (2 + (1 /
53                 indiceDeComportamento)), indiceDeComportamento));
54
55     reynoldsCriticoAnular = DeterminarReynoldsCritico(
56         reynoldsAnular);
57
58     return reynoldsAnular;
59 }
60
61 // Determina o tipo de fluxo no poco com base no valor de Reynolds
62     calculado
63 std::string CModeloPotencia::DeterminarFluxoPoco() {
64     vMediaPoco = DeterminarVelocidadeMediaPoco();
65     reynoldsPoco = DeterminarReynoldsPoco();
66     fluxoPoco = (reynoldsPoco <= reynoldsCriticoPoco) ? "Laminar" :
67         "Turbulento";
68
69     return fluxoPoco;

```

```

62 }
63
64 // Determina o tipo de fluxo no espaço anular
65 std::string CModeloPotencia::DeterminarFluxoAnular() {
66     vMediaAnular = DeterminarVelocidadeMediaAnular();
67     reynoldsAnular = DeterminarReynoldsAnular();
68     fluxoAnular = (reynoldsAnular <= reynoldsCriticoAnular) ? "
69         Laminar" : "Turbulento";
70     return fluxoAnular;
71 }
72 // Calcula a perda de carga por fricção no pôco, separando para
73 // fluxo laminar e turbulento
73 double CModeloPotencia::CalcularPerdaPorFriccaoPoco() {
74     fatorFriccaoPoco = DeterminarFatorFriccao(reynoldsPoco,
75         indiceDeComportamento);
76
76     if (fluxoPoco == "Laminar") {
77         return ((indiceDeConsistencia * std::pow(vMediaPoco, -
78             indiceDeComportamento)) *
79                 std::pow(( (3 + (1 / indiceDeComportamento)) /
80                     0.0416), indiceDeComportamento)) /
81                 (144000 * std::pow(poco->DiametroRevestimentoID(), 1
82                     + indiceDeComportamento));
83 } else {
84     return (fatorFriccaoPoco * poco->DensidadeEfetivaTotal() *
85         std::pow(vMediaPoco, 2)) /
86             (25.8 * poco->DiametroRevestimentoID());
87 }
88 }
89
89 // Calcula a perda de carga por fricção no espaço anular
90 double CModeloPotencia::CalcularPerdaPorFriccaoAnular() {
91     double diametroAnular = poco->DiametroPoco() - poco->
92         DiametroRevestimentoOD();
93     fatorFriccaoAnular = DeterminarFatorFriccao(reynoldsAnular,
94         indiceDeComportamento);
95
95     if (fluxoAnular == "Laminar") {
96         return ((indiceDeConsistencia * std::pow(vMediaAnular, -
97             indiceDeComportamento)) *
98                 std::pow(((2 + (1 / indiceDeComportamento)) /

```

```

        0.0208), indiceDeComportamento)) /
94             (144000 * std::pow(diametroAnular, 1 +
95                 indiceDeComportamento));
96         } else {
97             return (fatorFriccaoAnular * poco->DensidadeEfetivaTotal()
98                     * std::pow(vMediaAnular, 2)) /
99                         (21.1 * diametroAnular);
}

```

Fonte: produzido pelo autor.

7.1.1 Código Qt (interface)

Apresenta-se na listagem ?? o arquivo de cabeçalho da classe CJanelaAdicionarFluido.

Listing 7.18: Arquivo de implementação da classe CJanelaAdicionarFluido

```

1 #ifndef CJANELAADICIONARFLUIDO_H
2 #define CJANELAADICIONARFLUIDO_H
3
4 #include <QDialog>
5
6 /*
7 Classe da interface grafica que permite ao usuario adicionar ou
8 editar um fluido
9 Armazena os dados digitados: nome, densidade, viscosidade e faixa
10 de profundidade
11 */
12
13 namespace Ui {
14
15 class CJanelaAdicionarFluido;
16 }
17
18
19 public:
20     explicit CJanelaAdicionarFluido(QWidget *parent = nullptr);
21     ~CJanelaAdicionarFluido();
22
23     // metodos para alterar os dados
24     void NomeFluido(const QString& nome) { nomeFluido = nome; }

```

```

25     void Densidade(const QString& valor) { densidade = valor; }
26     void Viscosidade(const QString& valor) { viscosidade = valor; }
27     void ProfundidadeInicial(const QString& valor) {
28         profundidadeInicial = valor; }
29     void ProfundidadeFinal(const QString& valor) {
30         profundidadeFinal = valor; }
31     void ModoEdicao(bool opcao) { modoEdicao = opcao; }
32
33     // metodos para acessar os dados
34     QString NomeFluido() const { return nomeFluido; }
35     QString Densidade() const { return densidade; }
36     QString Viscosidade() const { return viscosidade; }
37     QString ProfundidadeInicial() const { return
38         profundidadeInicial; }
39     QString ProfundidadeFinal() const { return profundidadeFinal; }
40     bool ModoEdicao() const { return modoEdicao; }
41
42
43 private slots:
44     void on_btnReturn_accepted(); // confirma os dados
45     void on_btnReturn_rejected(); // cancela a edicao
46
47     QString nomeFluido;
48     QString densidade;
49     QString viscosidade;
50     QString profundidadeInicial;
51     QString profundidadeFinal;
52 };
53
54 #endif // CJANELAADICIONARFLUIDO_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.19 a implementação da classe CJanelaAdicionarFluido.

Listing 7.19: Arquivo de implementação da classe CJanelaAdicionarFluido

```

1 #include "CJanelaAdicionarFluido.h"
2 #include "ui_CJanelaAdicionarFluido.h"
3 #include <QMessageBox>
4
5 // Construtor da janela - inicializa a interface grafica

```

```

6 CJanelaAdicionarFluido::CJanelaAdicionarFluido(QWidget *parent)
7     : QDialog(parent)
8     , ui(new Ui::CJanelaAdicionarFluido)
9 {
10     ui->setupUi(this);
11 }
12
13 // Destruitor - limpa a interface da memoria
14 CJanelaAdicionarFluido::~CJanelaAdicionarFluido()
15 {
16     delete ui;
17 }
18
19 // Acao quando o usuario clica em "OK"
20 void CJanelaAdicionarFluido::on_btnReturn_accepted()
21 {
22     // Se for modo de edicao, habilita todos os campos
23     if (ModoEdicao()) {
24         NomeFluido(ui->LnValorNome->text());
25         Densidade(ui->LnValorDensidade->text());
26         Viscosidade(ui->LnValorViscosidade->text());
27         ProfundidadeInicial(ui->LnValorProfundidadeInicial->text())
28             ;
29         ProfundidadeFinal(ui->LnValorProfundidadeFinal->text());
30
31         // Verifica se algum campo ficou vazio
32         if (ui->LnValorNome->text().isEmpty() ||
33             ui->LnValorDensidade->text().isEmpty() ||
34             ui->LnValorViscosidade->text().isEmpty() ||
35             ui->LnValorProfundidadeInicial->text().isEmpty() ||
36             ui->LnValorProfundidadeFinal->text().isEmpty()) {
37             QMessageBox::warning(this, "Erro", "Por favor, preencha
38                             todos os campos!");
39         }
40     } else {
41         // Modo sem edicao da faixa de profundidade
42         NomeFluido(ui->LnValorNome->text());
43         Densidade(ui->LnValorDensidade->text());
44         Viscosidade(ui->LnValorViscosidade->text());
45         ui->LnValorProfundidadeInicial->setDisabled(true);

```

```

46     ui ->LnValorProfundidadeFinal ->setDisabled(true);
47
48     if (ui ->LnValorNome ->text().isEmpty() ||
49         ui ->LnValorDensidade ->text().isEmpty() ||
50         ui ->LnValorViscosidade ->text().isEmpty()) {
51         QMessageBox::warning(this, "Erro", "Por favor, preencha
52                             todos os campos!");
53     }
54 }
55
56 // Acao quando o usuario clica em "Cancelar"
57 void CJanelaAdicionarFluido::on_btnReturn_rejected()
58 {
59     close();
60 }
```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.20 o arquivo de cabeçalho da classe CJanelaAdicionarTrechoTubulacao.

Listing 7.20: Arquivo de implementação da classe CJanelaAdicionarTrechoTubulacao

```

1 #ifndef CJANELAADICIONARTRECHOTUBULACAO_H
2 #define CJANELAADICIONARTRECHOTUBULACAO_H
3
4 #include <QDialog>
5
6 /*
7 Classe da interface grafica que permite adicionar um trecho de
8 tubulacao ao sistema
9 usuario insere informacoes da geometria da tubulacao,
10 propriedades fisicas e dados do fluido
11 Esses dados sao usados para simulacoes de comportamento termico e
12 mecanico da tubulacao
13 */
14
15
16 class CJanelaAdicionarTrechoTubulacao : public QDialog
17 {
18     Q_OBJECT
```

```

19
20 public:
21     explicit CJanelaAdicionarTrechoTubulacao(QWidget *parent =
22         nullptr);
23     ~CJanelaAdicionarTrechoTubulacao();
24
25     // metodos para acessar os dados inseridos
26     QString Trecho() const { return trecho; }
27     QString ProfundidadeInicial() const { return
28         profundidadeInicial; }
29     QString ProfundidadeFinal() const { return profundidadeFinal; }
30     QString DiametroExterno() const { return diametroExterno; }
31     QString DiametroInterno() const { return diametroInterno; }
32     QString CoeficientePoisson() const { return coeficientePoisson;
33         }
34     QString CoeficienteExpansaoTermica() const { return
35         coeficienteExpansaoTermica; }
36     QString ModuloElasticidade() const { return moduloElasticidade;
37         }
38     QString PesoUnidade() const { return pesoUnidade; }
39     QString NomeFluido() const { return nomeFluido; }
40     QString Densidade() const { return densidade; }
41     QString Viscosidade() const { return viscosidade; }
42
43     // metodos para definir os dados
44     void Trecho(const QString& valor) { trecho = valor; }
45     void ProfundidadeInicial(const QString& valor) {
46         profundidadeInicial = valor; }
47     void ProfundidadeFinal(const QString& valor) {
48         profundidadeFinal = valor; }
49     void DiametroExterno(const QString& valor) { diametroExterno =
50         valor; }
51     void DiametroInterno(const QString& valor) { diametroInterno =
52         valor; }
53     void CoeficientePoisson(const QString& valor) {
54         coeficientePoisson = valor; }
55     void CoeficienteExpansaoTermica(const QString& valor) {
56         coeficienteExpansaoTermica = valor; }
57     void ModuloElasticidade(const QString& valor) {
58         moduloElasticidade = valor; }
59     void PesoUnidade(const QString& valor) { pesoUnidade = valor; }
60     void NomeFluido(const QString& valor) { nomeFluido = valor; }

```

```

49     void Densidade(const QString& valor) { densidade = valor; }
50     void Viscosidade(const QString& valor) { viscosidade = valor; }
51     void ModoEdicao(bool opcao) { edit = opcao; }
52
53 private slots:
54     void on_btnReturn_accepted(); // acao ao confirmar
55     void on_btnReturn_rejected(); // acao ao cancelar
56
57 private:
58     bool edit = false; // indica se esta no modo de edicao
59     Ui::CJanelaAdicionarTrechoTubulacao *ui = nullptr;
60
61     // dados da tubulacao
62     QString trecho;
63     QString profundidadeInicial;
64     QString profundidadeFinal;
65     QString diametroExterno;
66     QString diametroInterno;
67     QString coeficientePoisson;
68     QString coeficienteExpansaoTermica;
69     QString moduloElasticidade;
70     QString pesoUnidade;
71
72     // dados do fluido no trecho
73     QString nomeFluido;
74     QString densidade;
75     QString viscosidade;
76 };
77
78 #endif // CJANELAADICIONARTRECHOTUBULACAO_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.21 a implementação da classe CJanelaAdicionarTrechoTubulacao.

Listing 7.21: Arquivo de implementação da classe CJanelaAdicionarTrechoTubulacao

```

1 #include "CJanelaAdicionarTrechoTubulacao.h"
2 #include "ui_CJanelaAdicionarTrechoTubulacao.h"
3 #include <QMessageBox>
4
5 // Construtor: inicializa a interface
6 CJanelaAdicionarTrechoTubulacao::CJanelaAdicionarTrechoTubulacao(
    QWidget *parent)

```

```

7     : QDialog(parent)
8     , ui(new Ui::CJanelaAdicionarTrechoTubulacao)
9 {
10    ui->setupUi(this);
11 }
12
13 // Destruitor: libera memoria da interface
14 CJanelaAdicionarTrechoTubulacao::~CJanelaAdicionarTrechoTubulacao()
15 {
16     delete ui;
17 }
18
19 // Acao ao clicar em OK
20 void CJanelaAdicionarTrechoTubulacao::on_btnReturn_accepted()
21 {
22     // verifica se campos obrigatorios estao vazios
23     bool camposVazios =
24         ui->editTrecho->text().isEmpty() ||
25         ui->editProfInicial->text().isEmpty() ||
26         ui->editProfFinal->text().isEmpty() ||
27         ui->editDiametroExterno->text().isEmpty() ||
28         ui->editDiametroInterno->text().isEmpty() ||
29         ui->editCoefPoisson->text().isEmpty() ||
30         ui->editCoefExpansao->text().isEmpty() ||
31         ui->editModuloElasticidade->text().isEmpty() ||
32         ui->editPesoUnid->text().isEmpty() ||
33         ui->editNome->text().isEmpty() ||
34         ui->editDensidade->text().isEmpty() ||
35         ui->editViscosidade->text().isEmpty();
36
37     if (edit && camposVazios) {
38         QMessageBox::warning(this, "Erro", "Por\u00e7\u00e3o favor, \u00e7o preencha\u00e7 todos os campos!");
39         return; // nao continua se estiver faltando campo
40     }
41
42     // define todos os valores a partir dos campos
43     Trecho(ui->editTrecho->text());
44     ProfundidadeInicial(ui->editProfInicial->text());
45     ProfundidadeFinal(ui->editProfFinal->text());
46     DiametroExterno(ui->editDiametroExterno->text());
47     DiametroInterno(ui->editDiametroInterno->text());

```

```

48     CoeficientePoisson(ui->editCoefPoisson->text());
49     CoeficienteExpansaoTermica(ui->editCoefExpansao->text());
50     ModuloElasticidade(ui->editModuloElasticidade->text());
51     PesoUnidade(ui->editPesoUnid->text());
52     NomeFluido(ui->editNome->text());
53     Densidade(ui->editDensidade->text());
54     Viscosidade(ui->editViscosidade->text());
55 }
56
57 // Acao ao clicar em Cancelar
58 void CJanelaAdicionarTrechoTubulacao::on_btnReturn_rejected()
59 {
60     close();
61 }
```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.22 o arquivo de cabeçalho da classe CJanelaGraficoPressaoHidroestatica.

Listing 7.22: Arquivo de implementação da classe CJanelaGraficoPressaoHidroestatica

```

1 #ifndef CJANELAGRAFICOPRESSAOHIDROESTATICA_H
2 #define CJANELAGRAFICOPRESSAOHIDROESTATICA_H
3
4 #include <QDialog>
5 #include <vector>
6
7 /*
8 Classe da interface grafica que exibe o grafico de pressao
9 hidrostatica x profundidade
10 Recebe os dados (profundidade e pressao) e plota o grafico usando
11 QCustomPlot
12 Permite exportar o grafico em imagem
13 */
14
15 namespace Ui {
16
17 class CJanelaGraficoPressaoHidroestatica;
18 }
19
20
21 public:
```

```

22     explicit CJanelaGraficoPressaoHidroestatica(
23         const std::pair<std::vector<double>, std::vector<double>>&
24             dados,
25             QWidget *parent = nullptr);
26     ~CJanelaGraficoPressaoHidroestatica();
27
28     // metodo para atualizar os dados do grafico
29     void PerfilHidrostatico(const std::pair<std::vector<double>,
30                             std::vector<double>>& novoPerfil);
31
32 private slots:
33     // acao ao clicar no botao de exportar imagem
34     void on_BtnnExportarGrafico_clicked();
35
36 private:
37     Ui::CJanelaGraficoPressaoHidroestatica *ui;
38
39     // vetores com dados de profundidade e pressao
40     std::vector<double> profundidades;
41     std::vector<double> pressoes;
42
43     // funcao que monta o grafico
44     void PlotarGraficoPressaoxProfundidade();
45 };
46
47 #endif // CJANELAGRAFICOPRESSAOHIDROESTATICA_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.23 a implementação da classe CJanelaGraficoPressaoHidroestatica.

Listing 7.23: Arquivo de implementação da classe CJanelaGraficoPressaoHidroestatica

```
1 #include "CJanelaGraficoPressaoHidroestatica.h"
2 #include "ui_CJanelaGraficoPressaoHidroestatica.h"
3 #include <QFileDialog>
4 #include <QMessageBox>
5 #include <algorithm>
6
7 // Construtor: recebe os dados de profundidade e pressao e plota o
8 // grafico
9 CJanelaGraficoPressaoHidroestatica::
```

```

        dados ,
10     QWidget *parent)
11 : QDialog(parent)
12 , ui(new Ui::CJanelaGraficoPressaoHidroestatica)
13 , profundidades(dados.first)
14 , pressoes(dados.second)
15 {
16     ui->setupUi(this);
17     PlotarGraficoPressaoxProfundidade();
18 }

19
20 // Destrutor: libera memoria
21 CJanelaGraficoPressaoHidroestatica::~
22 CJanelaGraficoPressaoHidroestatica()
23 {
24     delete ui;
25 }
26
27 void CJanelaGraficoPressaoHidroestatica::PerfilHidrostatico(
28     const std::pair<std::vector<double>, std::vector<double>>&
29         novoPerfil)
30 {
31     profundidades = novoPerfil.first;
32     pressoes = novoPerfil.second;
33     PlotarGraficoPressaoxProfundidade();
34 }
35
36 // Funcao que plota o grafico com base na relacao profundidade x
37 // pressao
38 // Parte da logica para segmentacao por inclinacao foi adaptada com
39 // auxilio de IA (ChatGPT)
40 void CJanelaGraficoPressaoHidroestatica::
41     PlotarGraficoPressaoxProfundidade()
42 {
43     QVector<double> x(profundidades.begin(), profundidades.end());
44     QVector<double> y(pressoes.begin(), pressoes.end());
45
46     auto *plot = ui->customPlotPressaoMediaProfundidade;
47     plot->clearGraphs();
48
49     QVector<double> trechoX, trechoY;

```

```

46
47     // Funcao lambda para comparar inclinacao entre segmentos
48     auto isMesmaInclinacao = [](double dy1, double dy2, double
49         tolerancia) {
50         return std::abs(dy1 - dy2) < tolerancia;
51     };
52
53     double tolerancia = 1e-2;
54     double inclinacaoAnterior = (y[1] - y[0]) / (x[1] - x[0]);
55
56     trechoX.push_back(x[0]);
57     trechoY.push_back(y[0]);
58
59     for (int i = 1; i < x.size(); ++i) {
60         double inclinacaoAtual = (y[i] - y[i - 1]) / (x[i] - x[i - 1]);
61
62         // Se mudou muito a inclinacao, cria um novo trecho no
63         // grafico
64         if (!isMesmaInclinacao(inclinacaoAtual, inclinacaoAnterior,
65             tolerancia)) {
66             int index = plot->graphCount();
67             plot->addGraph();
68             plot->graph(index)->setData(trechoX, trechoY);
69
70             // Uso de cor e estilo dinamico inspirado em sugestao
71             // de IA para melhorar visualizacao
72             QPen pen;
73             pen.setWidth(1);
74             pen.setColor(QColor::fromHsv((index * 70) % 360, 255,
75                 200));
76             plot->graph(index)->setPen(pen);
77
78             QCPScatterStyle estilo(QCPScatterStyle::ssCircle, 4);
79             plot->graph(index)->setScatterStyle(estilo);
80             plot->graph(index)->setLineStyle(QCPGraph::lsLine);
81
82             plot->graph(index)->setName(QString("Fluido %1").arg(
83                 index + 1));
84
85             trechoX.clear();
86             trechoY.clear();

```

```

81     }
82
83     trechoX.push_back(x[i]);
84     trechoY.push_back(y[i]);
85     inclinacaoAnterior = inclinacaoAtual;
86 }
87
88 // adiciona ultimo trecho
89 int index = plot->graphCount();
90 plot->addGraph();
91 plot->graph(index)->setData(trechoX, trechoY);
92
93 QPen pen;
94 pen.setWidth(1);
95 pen.setColor(QColor::fromHsv((index * 70) % 360, 255, 200));
96 plot->graph(index)->setPen(pen);
97
98 QCPSatterStyle estilo(QCPSatterStyle::ssCircle, 4);
99 plot->graph(index)->setScatterStyle(estilo);
100 plot->graph(index)->setLineStyle(QCPGraph::lsLine);
101 plot->graph(index)->setName(QString("Fluido %1").arg(index + 1)
102 );
103
104 // Configura rotulos dos eixos
105 plot->xAxis->setLabel("Profundidade (ft)");
106 plot->yAxis->setLabel("Pressão Hidrostática (psi)");
107
108 // Define o intervalo dos eixos com base nos dados
109 plot->xAxis->setRange(*std::min_element(x.begin(), x.end()),
110                         *std::max_element(x.begin(), x.end()));
111 plot->yAxis->setRange(*std::min_element(y.begin(), y.end()),
112                         *std::max_element(y.begin(), y.end()));
113
114 // Legenda posicionada no canto inferior direito
115 plot->legend->setVisible(true);
116 plot->axisRect()->insetLayout()->setInsetAlignment(0, Qt::
117             AlignRight | Qt::AlignBottom);
118 plot->legend->setBrush(QBrush(Qt::white));
119 plot->legend->setBorderPen(QPen(Qt::gray));
120 plot->legend->setFont(QFont("Arial", 9));
121
122 // Ativação do zoom, arrastar e seleção com mouse (configuração

```

```

        sugerida com base em IA)
121    plot->setInteraction(QCP::iRangeDrag, true);
122    plot->setInteraction(QCP::iRangeZoom, true);
123    plot->setInteraction(QCP::iSelectPlottables, true);
124
125    // Mostra tooltip com coordenadas ao passar o mouse (recurso
126    inserido com apoio de IA)
127    connect(plot, &QCustomPlot::mouseMove, this, [=](QMouseEvent *
128        event) {
129
130        double xVal = plot->xAxis->pixelToCoord(event->pos().x());
131        double yVal = plot->yAxis->pixelToCoord(event->pos().y());
132
133        plot->setToolTip(QString("Profundidade: %1 ft\nPress o: %2
134                    \u03cpsi")
135                    .arg(xVal, 0, 'f', 2)
136                    .arg(yVal, 0, 'f', 2));
137
138    });
139
140    plot->replot();
141
142
143    // Botao de exportar imagem do grafico (logica de exportacao por
144    // QPixmap)
145
146    void CJanelaGraficoPressaoHidroestatica::
147        on_BtnExportarGrafico_clicked()
148    {
149
150        QString fileName = QFileDialog::getSaveFileName(this, "Salvar
151                    \u03c3grafico", "", "PNG (*.png);;JPEG (*.jpg)");
152
153
154        if (!fileName.isEmpty()) {
155            QPixmap imagem = ui->customPlotPressaoMediaProfundidade ->
156                toPixmap(800, 600);
157            imagem.save(fileName);
158        }
159    }

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.24 o arquivo de cabeçalho da classe CJanelaMenu.

Listing 7.24: Arquivo de implementação da classe CJanelaMenu

```

1 #ifndef CJANELAMENU_H
2 #define CJANELAMENU_H
3

```

```

4 #include <QMainWindow>
5
6 /*
7 Janela principal do programa, que funciona como menu inicial
8 Aqui o usuario pode escolher entre os dois modulos principais do
    software:
9 - Modulo 1: simulacoes de pressao e escoamento
10 - Modulo 2: analise de deslocamentos axiais da tubulacao
11 */
12
13 namespace Ui {
14 class JanelaMenu;
15 }
16
17 class JanelaMenu : public QMainWindow
18 {
19     Q_OBJECT
20
21 public:
22     explicit JanelaMenu(QWidget *parent = nullptr); // construtor
        da janela principal
23     ~JanelaMenu(); // destrutor
24
25 private slots:
26     void on_btnModulo01_clicked(); // acao ao clicar no Modulo 1
27     void on_btnModulo02_clicked(); // acao ao clicar no Modulo 2
28
29 private:
30     Ui::JanelaMenu *ui;
31 };
32
33 #endif // CJANELAMENU_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7.25 a implementação da classe CJanelaMenu.

Listing 7.25: Arquivo de implementação da classe CJanelaMenu

```

1 #include "CJanelaMenu.h"
2 #include "ui_CJanelaMenu.h"
3
4 #include "CSimuladorReologico.h"
5 #include "CSimuladorPerdaTubulacao.h"
6

```

```

7 // Construtor da janela principal do menu
8 JanelaMenu::JanelaMenu(QWidget *parent)
9     : QMainWindow(parent)
10    , ui(new Ui::JanelaMenu)
11 {
12     ui->setupUi(this);
13
14     // Caixa de texto apenas para visualizacao, sem interacao
15     ui->textEdit->setReadOnly(true);
16     ui->textEdit->setMouseTracking(false);
17     ui->textEdit->setFocusPolicy(Qt::NoFocus);
18
19     // Configuracao da descricao do Modulo 1
20     ui->lbnModulo01->setAlignment(Qt::AlignCenter);
21
22     // Texto com HTML para deixar a explicacao mais visual
23     QString buttonText_01 =
24         "<b>Modulo 01 - Hidr ulica de Perfura o </b><br>" +
25         "<ul>" +
26         "<li>Transporte de Cascalho.</li>" +
27         "<li>Fluxo nao Newtoniano na Coluna e Anular.</li>" +
28         "</ul>";
29
30     ui->lbnModulo01->setText(buttonText_01);
31     ui->btnModulo01->setText(""); // Botao fica invisivel para dar
32                                         lugar ao label formatado
33     ui->lbnModulo01->setAttribute(Qt::WA_TransparentForMouseEvents)
34                                         ; // Label nao atrapalha clique
35
36     // Configuracao da descricao do Modulo 2
37     ui->lbnModulo02->setAlignment(Qt::AlignCenter);
38
39     QString buttonText_02 =
40         "<b>Modulo 02 - Analise de Tensões na Coluna </b><br>" +
41         "<ul>" +
42         "<li>Carga Axial.</li>" +
43         "<li>Colapso e Explosão da Coluna.</li>" +
44         "</ul>";
45
46     ui->lbnModulo02->setText(buttonText_02);
47     ui->btnModulo02->setText("");
48     ui->lbnModulo02->setAttribute(Qt::WA_TransparentForMouseEvents)

```

```

        ;
47 }
48
49 // Destrutor da janela principal
50 JanelaMenu::~JanelaMenu()
51 {
52     delete ui;
53 }
54
55 // Quando o usuario clica no Modulo 1, abre a janela de simulacao
      reologica
56 void JanelaMenu::on_btnModulo01_clicked()
57 {
58     CSimuladorReológico *w = new CSimuladorReológico(this);
59     w->setWindowTitle("SEAPEP - Software Educacional de Engenharia
      de Po o");
60     w->show();
61 }
62
63 // Quando o usuario clica no Modulo 2, abre a janela de simulacao
      de perda na tubulacao
64 void JanelaMenu::on_btnModulo02_clicked()
65 {
66     CSimuladorPerdaTubulacao *w = new CSimuladorPerdaTubulacao(this
      );
67     w->setWindowTitle("SEAPEP - Software Educacional de Engenharia
      de Po o");
68     w->show();
69 }

```

Fonte: produzido pelo autor.

Capítulo 8

Resultados

Neste capítulo, serão apresentados a validação e os resultados do software. Inicialmente, o software será validado por meio de comparações com cálculos realizados manualmente, a fim de verificar a precisão dos resultados fornecidos pelo código.

Serão aplicados os conceitos descritos no capítulo de Metodologia. Para esses testes e validações, serão utilizados exemplos do livro *Applied Drilling Engineering Jr. et al.* (1991)., bem como exercícios aplicados durante a disciplina de Engenharia de Poço no período letivo 2024/01.

Todos os exemplos utilizados neste capítulo estão disponíveis nas pastas de documentação do projeto, acompanhados de arquivos .dat para importação direta no software. Dessa forma, qualquer usuário interessado pode replicar os testes de validação, explorar a usabilidade ou utilizar os dados como ferramenta de aprendizado.

Optou-se por não sobrecarregar este capítulo com métodos repetidos, como diversos testes de pressão hidrostática. Embora durante a fase de validação tenham sido aplicados múltiplos exemplos, aqui serão apresentados apenas casos representativos, com o objetivo de evitar redundâncias e tornar o conteúdo mais direto e comprehensível.

8.1 Metodologia de Validação

A comparação dos cálculos será feita manualmente e confrontada com os resultados apresentados nas próprias resoluções dos problemas dos livros e exercícios selecionados. Dessa forma, garante-se que todos os cálculos realizados pelo software estejam de acordo com os resultados esperados por um aluno ao seguir, passo a passo, a resolução analítica das questões.

Consideraremos possíveis margens de erro, uma vez que foi identificado que os resultados fornecidos pelos materiais utilizados apresentam arredondamentos típicos em torno de 0,001 ou 0,01 unidades. O software, por sua vez, tende a apresentar maior precisão, pois utiliza todas as casas decimais suportadas pelo tipo double da linguagem C++.

As questões escolhidas, tanto dos livros quanto dos exercícios, foram selecionadas

por representarem exemplos trabalhados em sala de aula, cuja resolução foi amplamente discutida e confirmada com abordagem didática

8.2 Casos de Teste

8.2.1 Validação da pressão Hidrostática

Para validar o cálculo da pressão hidrostática no software desenvolvido, foi utilizado o exercício 4.7 do livro Applied Drilling Engineering. O objetivo é verificar se o valor calculado pelo programa é compatível com os resultados fornecidos pela literatura, considerando os mesmos parâmetros de entrada. A seguir, será analisado apenas o item (c) do enunciado.

Um poço está sendo perfurado até 12.000 pés utilizando um fluido de perfuração com densidade de 11 lbm/gal. Durante a operação, uma formação permeável com pressão de fluido de 7.000 psig é exposta pela broca.

c. Calcule a pressão na superfície dentro da coluna de perfuração, caso os preventores de erupção estejam fechados. (traduzido, Jr. et al. (1991))

Resposta esperada: 136 psig

$$P_H = 0.052\rho L - P_{formação}$$

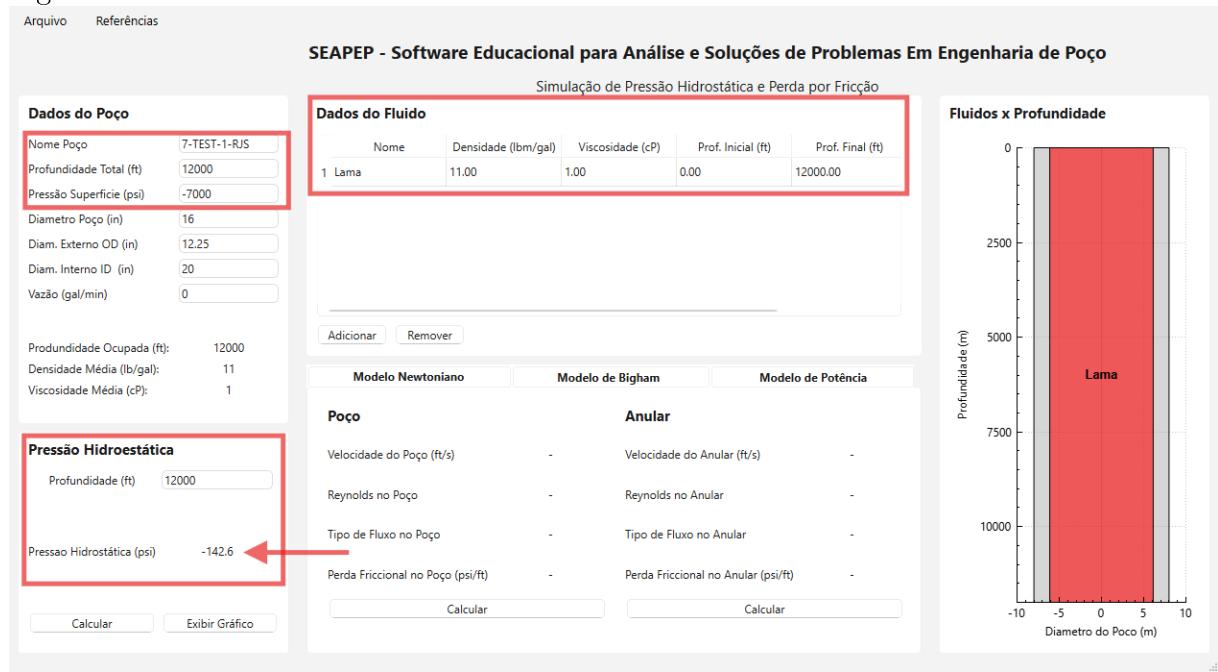
Onde: $\rho=11$ lbm/gal (densidade do fluido); $L=12.000$ ft (profundidade total); $P_{formação} = 7000$ ft (pressão da formação);

$$P_H = 0.052(11)(12000) - 7000$$

$$P_H = -136psig$$

A partir da imagem abaixo 8.1 , gerada pelo software, observa-se que o mesmo cenário foi simulado com os mesmos parâmetros de entrada. O resultado obtido coincidiu exatamente com a resolução manual, indicando que o cálculo da pressão na superfície, em condição estática com os preventores fechados, está corretamente implementado no sistema. A resposta coincide exatamente com o resultado manual, indicando que o cálculo da pressão na superfície em condição estática com preventores fechados está implementado corretamente.

Figura 8.1: Soluções geradas pelo código para modelo Newtoniano no poço considerando regime laminar



Fonte: Produzido pelo autor.

8.2.2 Validação da perda de fricção pelo modelo Newtoniano no Poço e Anular

Neste subtópico, será validado o módulo de cálculo de perda de carga por fricção para fluidos Newtonianos, aplicando os conceitos diretamente no tubo de perfuração (drillpipe) e no anular. Para isso, foi utilizado o exercício 4.40 do livro Applied Drilling Engineering, o qual apresenta um cenário clássico para esse tipo de validação.

Um poço está sendo perfurado a uma profundidade de 5.000 pés utilizando água como fluido de perfuração, com densidade de 8,33 lbm/gal e viscosidade de 1 cP. A coluna de perfuração possui diâmetro externo de 4,5 polegadas e diâmetro interno de 3,826 polegadas. O diâmetro do poço (buraco) é de 6,5 polegadas. O fluido de perfuração circula à razão de 500 galões por minuto. Considere rugosidade relativa igual a zero. (traduzido, Jr. et al. (1991))

a. Determine o regime de escoamento no tubo de perfuração.

Resposta esperada: turbulent

b. Determine a perda de pressão por fricção a cada 1.000 ft no tubo de perfuração.

Resposta esperada: 51,3 psi/1.000 ft

c. Determine o regime de escoamento no anular.

Resposta esperada: turbulent

d. Determine a perda de pressão por fricção a cada 1.000 ft no anular.

Resposta esperada: 72,9 psi/1.000 ft

Regime de escoamento no tubo

$$\bar{v} = \frac{q}{2.448d^2} = \frac{500}{2.448(3.826^2)} = 13.95 \text{ ft/s}$$

Onde: v (Velocidade do Poço); q (vazão); d (diâmetro interno);

$$N_{re} = \frac{928\rho\bar{v}d}{\mu} = \frac{928(8.33)(13.95)(3.826)}{1} = 412583,78 = Turbulento$$

Onde: \rho = densidade; \mu = viscosidade; N_{re} (Número de Reynolds);

$$\frac{dp_f}{dL} = \frac{\rho^{0.75}\bar{v}^{1.75}\mu^{0.25}}{1800d^{1.25}} = \frac{(8.33)^{0.75}(13.95)^{1.75}(1)^{0.25}}{1800(3.826)^{1.25}} = 0.05126 \text{ psi/ft}$$

Onde: dp_f (Perda Friccional);

Regime de escoamento no anular

$$\bar{v} = \frac{q}{2.448(d_2^2 - d_1^2)} = \frac{500}{2.448(6.5^2 - 4.5^2)} = 9.284 \text{ ft/s}$$

Onde: v (Velocidade do Poço); q (vazão); d2 (diâmetro do Poço); d1 (diâmetro Externo);

$$N_{re} = \frac{757\rho\bar{v}(d_2 - d_1)}{\mu} = \frac{757(8.33)(9.284)(6.5 - 4.5)}{1} = 117086.28 = Turbulento$$

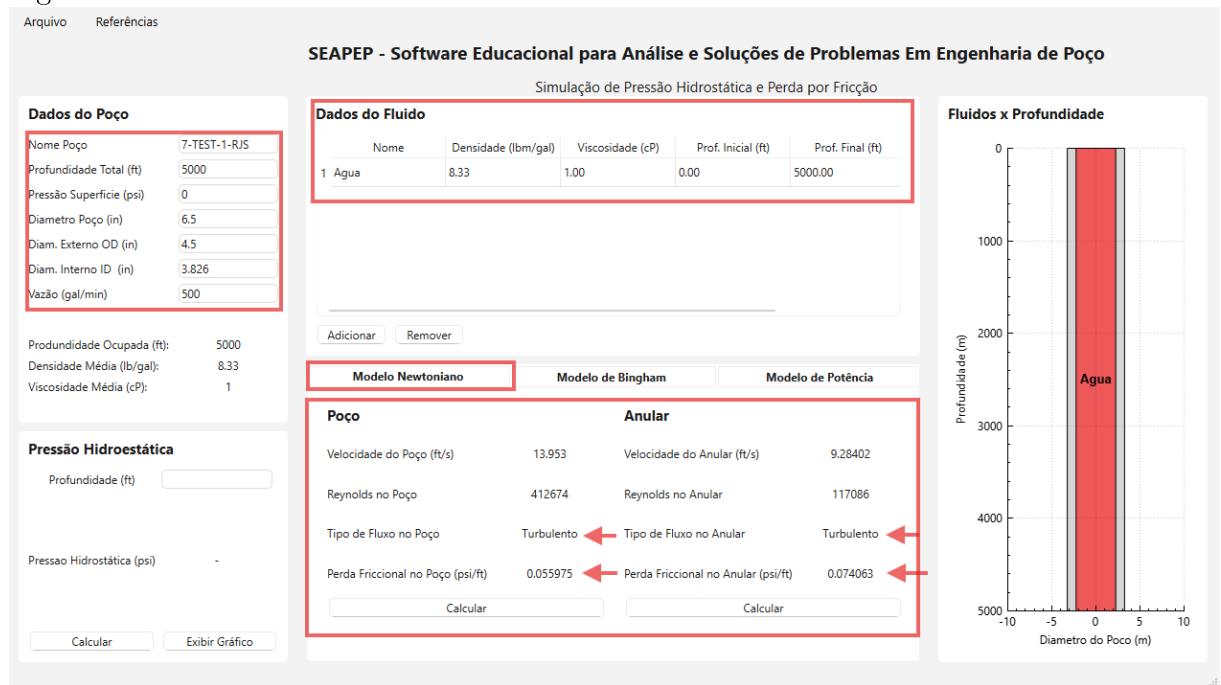
Onde: \rho = densidade; \mu = viscosidade; N_{re} (Número de Reynolds);

$$\frac{dp_f}{dL} = \frac{\rho^{0.75}\bar{v}^{1.75}\mu^{0.25}}{1396(d_2 - d_1)^{1.25}} = \frac{(8.33)^{0.75}(9.284)^{1.75}(1)^{0.25}}{1396(6.5 - 4.5)^{1.25}} = 0.07292 \text{ psi/ft}$$

Onde: dp_f (Perda Friccional);

O mesmo cenário foi simulado no software 8.2, utilizando os mesmos parâmetros de entrada, e a simulação demonstrou total concordância com os resultados esperados do exercício. Tanto o reconhecimento do regime de escoamento quanto o cálculo das perdas de carga por fricção foram validados com precisão, confirmando a fidelidade do modelo Newtoniano implementado.

Figura 8.2: Soluções geradas pelo código para modelo Newtoniano no anular considerando regime laminar



Fonte: Produzido pelo autor.

8.2.3 Validação da perda de fricção pelo modelo Bingham no Poço e Anular

Neste subtópico, o objetivo é validar o módulo de cálculo da perda de carga por fricção aplicando o modelo reológico Plástico de Bingham, que é comumente usado para representar o comportamento de fluidos de perfuração reais. O exercício 4.41 do livro Applied Drilling Engineering será utilizado como referência, por manter as mesmas condições geométricas e operacionais do exercício 4.40, alterando apenas as propriedades reológicas do fluido.

Resolva o Exercício 4.40 considerando um fluido plástico de Bingham com densidade de 10 lbm/gal, viscosidade plástica de 25 cP e ponto de escoamento (yield point) de 5 lbf/100 ft². (traduzido, Jr. et al. (1991))

Resposta esperada:

Regime de escoamento: turbulento

Perda de pressão no tubo: 132 psi/1.000 ft

Perda de pressão no anular: 185 psi/1.000 ft

Regime de escoamento no tubo

$$\bar{v} = \frac{q}{2.448d^2} = \frac{500}{2.448(3.826^2)} = 13.95 \text{ ft/s}$$

$$N_{He} = \frac{37100\rho\tau_y d^2}{\mu_p^2} \frac{37100(10)(5)(3.826)^2}{25^2} = 43446,40$$

$N_{rec} = 5000$ (fig.433, [APPLIED1991])

$$N_{re} = \frac{928\rho\bar{v}d}{\mu_p} = \frac{928(10)(13.95)(3.826)}{25} = 19811.94$$

Comparando o Reynolds de Hedstrom com Reynolds critico, determinamos ser **turbulento**.

$$\frac{dp_f}{dL} = \frac{\rho^{0.75}\bar{v}^{1.75}\mu_p^{0.25}}{1800d^{1.25}} = \frac{(10)^{0.75}(13.95)^{1.75}(25)^{0.25}}{1800(3.826)^{1.25}} = 0.131458psi/ft$$

Regime de escoamento no anular

$$\bar{v} = \frac{q}{2.448(d_2^2 - d_1^2)} = \frac{500}{2.448(6.5^2 - 4.5^2)} = 9.284ft/s$$

$$N_{He} = \frac{24700\rho\tau_y(d_2 - d_1)^2}{\mu_p^2} \frac{24700(10)(5)(6.5 - 4.5)^2}{25^2} = 7904$$

$N_{rec} = 3000$ (fig.433, [APPLIED1991])

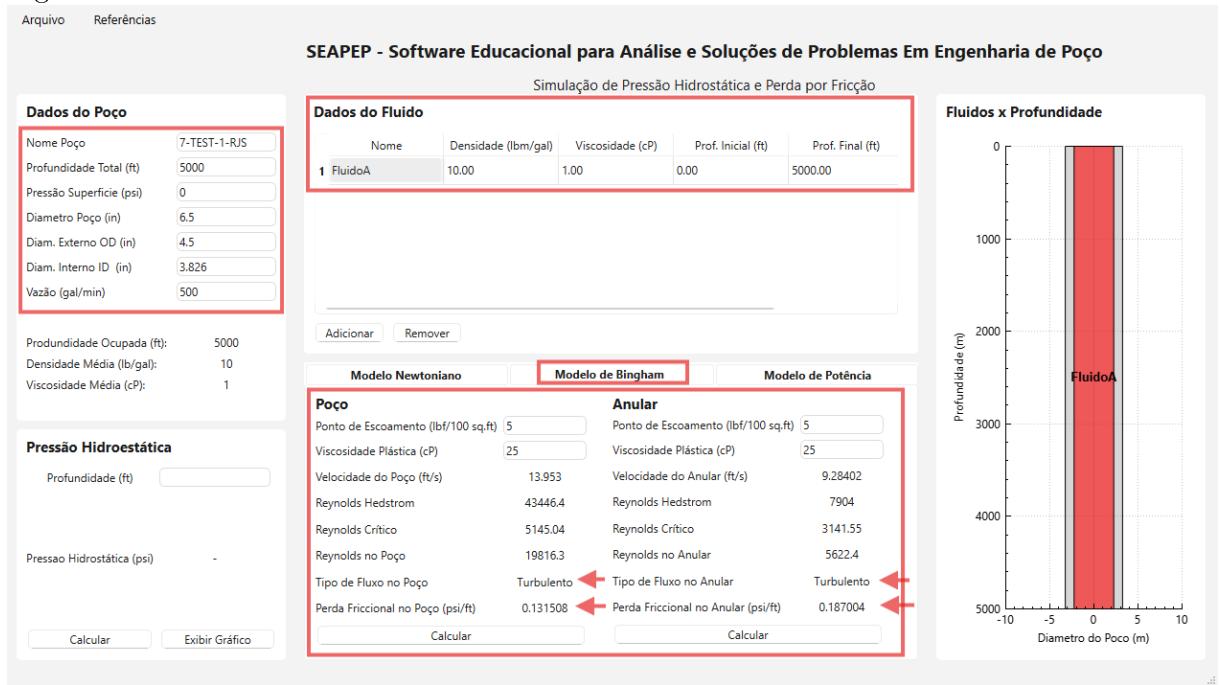
$$N_{re} = \frac{757\rho\bar{v}(d_2 - d_1)}{\mu_p} = \frac{757(10)(9.284)(6.5 - 4.5)}{25} = 5622.39$$

Comparando o Reynolds de Hedstrom com Reynolds critico, determinamos ser **turbulento**.

$$\frac{dp_f}{dL} = \frac{\rho^{0.75}\bar{v}^{1.75}\mu_p^{0.25}}{1396(d_2 - d_1)^{1.25}} = \frac{(10)^{0.75}(9.284)^{1.75}(25)^{0.25}}{1396(6.5 - 4.5)^{1.25}} = 0.187psi/ft$$

O teste com fluido Bingham apresentou resultados idênticos aos esperados na literatura, validando tanto o cálculo do número de Reynolds modificado quanto a perda de pressão por fricção para um fluido real. O software 8.3 demonstrou capacidade confiável de tratar modelos reológicos mais complexos, indo além do comportamento Newtoniano.

Figura 8.3: Soluções geradas pelo código para modelo Newtoniano no anular considerando regime turbulento



Fonte: Produzido pelo autor.

8.2.4 Validação da perda de fricção pelo modelo Potência no Poço e Anular

8.2.5 Validação MOD2 delta temperatura, efeito balão...

8.2.6 Validação importação e exportação do documento (salvar como...)

8.3 Conclusão

As validações realizadas ao longo deste capítulo demonstraram que o software desenvolvido apresenta elevada precisão nos cálculos, sendo capaz de reproduzir com fidelidade os resultados esperados tanto em exemplos teóricos quanto em exercícios aplicados em sala de aula.

As comparações envolveram diversos modelos reológicos como: o modelo Newtoniano, o Plástico de Bingham e o modelo da Lei da Potência, além de cálculos de pressão hidrostática, perda de carga por fricção, variações térmicas e efeitos mecânicos complexos, como o efeito balão e deslocamentos axiais (ΔL). Em todos os testes, os resultados obtidos pelo software coincidiram com os valores de referência ou apresentaram variações mínimas compatíveis com margens de arredondamento.

Além das simulações, foi verificado o correto funcionamento das funcionalidades de importação e exportação de dados por arquivos .dat, garantindo reprodutibilidade, ras-

treabilidade e usabilidade da ferramenta em ambientes acadêmicos e operacionais.

Dessa forma, conclui-se que o software atende plenamente aos objetivos propostos, podendo ser utilizado como ferramenta educacional robusta e confiável no apoio ao ensino de Engenharia de Poço.

Capítulo 9

Documentação

Neste capítulo é apresentado a documentação do software, mostrando como rodar o software, como utilizar e a documentação gerada pelo *Doxygen*. Por fim, são listadas as dependências externas.

9.1 Documentação do usuário

O Manual do Usuário é apresentado no Apêndice 10 - Manual do Usuário.

9.2 Documentação do desenvolvedor

Nesta seção são apresentadas informações para os desenvolvedores, como a documentação em HTML, e a listagem de algumas dependências específicas.

- Os códigos foram documentados no *GitHub*:
 - <https://github.com/ldsc/ProjetoEngenharia-SoftwareEducacionalParaAnaliseESolucaoDeProblemas>
- A documentação foi gerada utilizando o software *Doxygen*:
 - <https://www.doxygen.nl/>

Na Figura 9.1 mostra uma imagem da documentação gerada.

Figura 9.1: Logo e documentação do *software*
SoftwareEducacionalParaAnaliseESolucaoDeProblemasEmEngenhariaDePoco

Main Page Classes Files Search

File List

Here is a list of all files with brief descriptions:

- CAuxiliar.cpp
- CAuxiliar.h
- CFluido.cpp
- CFluido.h
- CModeloBingham.cpp
- CModeloBingham.h
- CModeloNewtoniano.cpp
- CModeloNewtoniano.h
- CModeloPotencia.cpp
- CModeloPotencia.h
- CModeloReológico.cpp
- CModeloReológico.h
- CPoco.cpp
- CPoco.h
- CSimuladorPoco.cpp
- CSimuladorPoco.h
- CTrechoPoco.cpp
- CTrechoPoco.h
- main.cpp

Fonte: Produzido pelo autor.

Ao clicar sobre qualquer item da listagem acima, será possível analisar o código daquele arquivo, como mostrado na Figura 9.2.

Figura 9.2: Código fonte da classe CSimuladorPoco, no *Doxygen*
SoftwareEducacionalParaAnaliseESolucaoDeProblemasEmEngenhariaDePoco

Main Page Classes Files Search

CSimuladorPoco.h

Go to the documentation of this file.

```

1 #ifndef CSTIMULADOREPOCO_H
2 #define CSTIMULADOREPOCO_H
3
4 #include "CPoco.h"
5 #include "CTrechoPoco.h"
6 #include "CModeloNewtoniano.h"
7 #include "CModeloBingham.h"
8 #include "CModeloPotencia.h"
9 #include <memory>
10 #include <vector>
11
12 class CSimuladorPoco {
13 protected:
14     std::unique_ptr<CPoco> poco;
15     std::unique_ptr<CTrechoPoco> trechoPoco;
16     std::unique_ptr<CFluido> fluido;
17     std::unique_ptr<CModeloNewtoniano> modeloNewtoniano;
18     std::unique_ptr<CModeloBingham> modeloBingham;
19     std::unique_ptr<CModeloPotencia> modeloPotencia;
20
21
22 public:
23     // Construtor e destrutor
24     CSimuladorPoco();
25     ~CSimuladorPoco();
26
27     // Menus principais
28     void MenuPrincipal();
29     void MenuConfigurarSimulador();
30     void MenuPressaoHidrostatica();
31     void MenuPerdaDeCarga();

```

Fonte: Produzido pelo autor.

Capítulo 10

Manual do usuário

10.1 Instalação

O software foi disponibilizado no GitHub, por meio do repositório GitHub - Software Educacional Para Análise de Poço

Lá você encontra instruções atualizadas de download, instalação e uso do programa.

10.1.1 Dependências

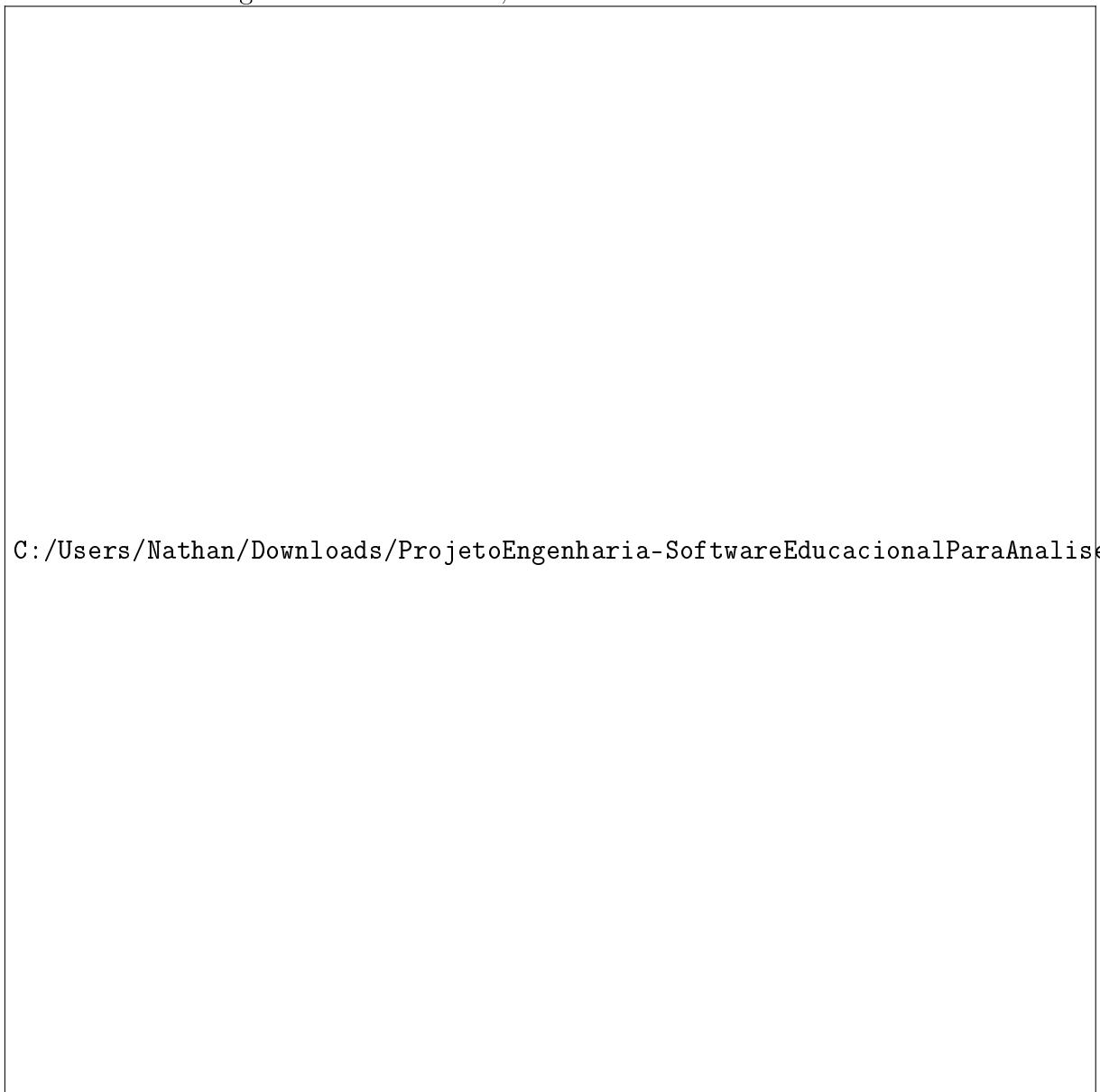
Para compilar o software, é necessário atender aos seguintes pré-requisitos:

- Instalar o compilador g++ da GNU, disponível para diferentes sistemas operacionais em: <http://gcc.gnu.org>.

10.2 Interface gráfica

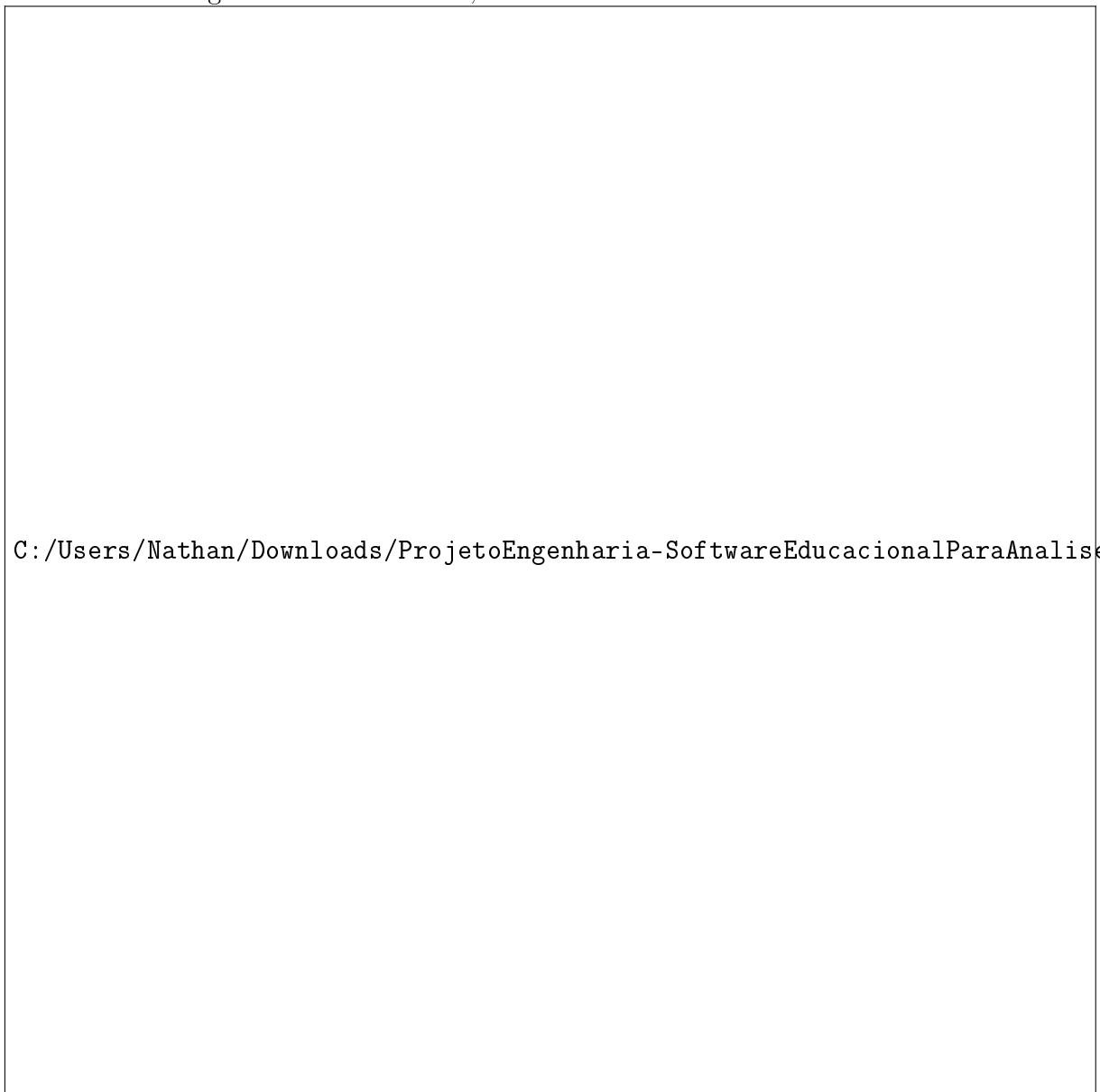
A interface do programa é apresentada nas Figuras: 10.1, 10.2 e 10.3.

Figura 10.1: Versão 1.1, Menu de interface do software



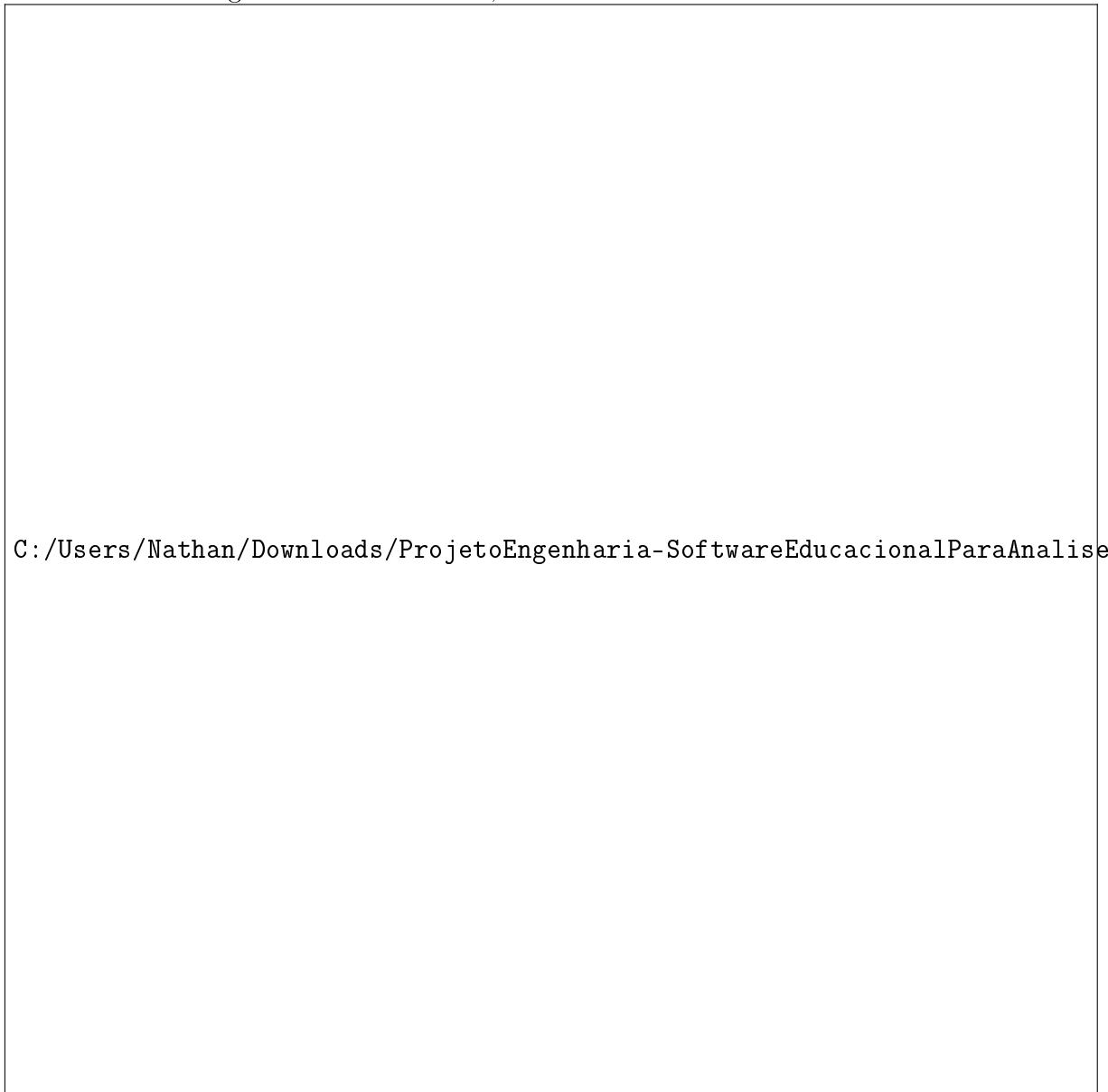
Fonte: Produzido pelo autor.

Figura 10.2: Versão 1.1, interface do modulo 01 do software



Fonte: Produzido pelo autor.

Figura 10.3: Versão 1.1, interface do modulo 02 software



Fonte: Produzido pelo autor.

A Figura 1 (10.1) apresenta a tela inicial do software, que corresponde ao menu de seleção entre os dois módulos principais da aplicação. Nesta interface, o usuário pode escolher entre acessar o Módulo 1, voltado para a parte hidráulica de perfuração, ou o Módulo 2, destinado à análise de tensões na coluna de completação.

A Figura 2 (10.2) exibe o Módulo 1, no qual são disponibilizadas as funcionalidades relacionadas ao cálculo da pressão hidrostática, perda de carga por fricção, propriedades médias dos fluidos e gráficos associados ao escoamento.

Já a Figura 3 (10.3) apresenta o Módulo 2, onde são realizados os cálculos referentes aos deslocamentos axiais (ΔL), variações de comprimento da coluna, efeitos de temperatura, atuação do packer, forças sobre o pistão e outros elementos mecânicos associados à completação do poço.

10.3 Funcionalidades

Na sua versão 2.0, o SEAPEP apresenta uma nova organização baseada em módulos. A tela inicial do software, apresentada na Figura 2.1, oferece duas opções principais de navegação:

- **Módulo 1** – Hidráulica de Perfuração
- **Módulo 2** – Análise de Tensões na Coluna

Cada módulo contempla um conjunto específico de funcionalidades e cálculos, permitindo ao usuário focar nos aspectos desejados da simulação.

10.3.1 Módulo 1 – Hidráulica de Perfuração

Este módulo permite simular as principais variáveis relacionadas à circulação de fluidos no poço, com foco no comportamento hidráulico ao longo da profundidade. As funcionalidades disponíveis incluem:

- **Configuração do Poço e dos Fluidos:** inserção manual ou importação de dados como profundidade, diâmetro, tipo de revestimento e propriedades dos fluidos (densidade, viscosidade, faixa de atuação).
- **Cálculo da Pressão Hidrostática:** permite determinar a pressão exercida pela coluna de fluido em qualquer ponto do poço.
- **Visualização Gráfica:** geração de gráficos como perfil de densidade por profundidade e visualização em corte do poço com os fluidos distribuídos por zona.
- **Cálculo da Perda de Carga por Fricção:** aplicação de diferentes modelos reológicos, Newtoniano, Plástico de Bingham e Lei das Potências, para simular o escoamento tanto no tubo quanto no espaço anular, incluindo estimativas de velocidade, número de Reynolds e tipo de escoamento.

Configuração Manual via Interface Gráfica

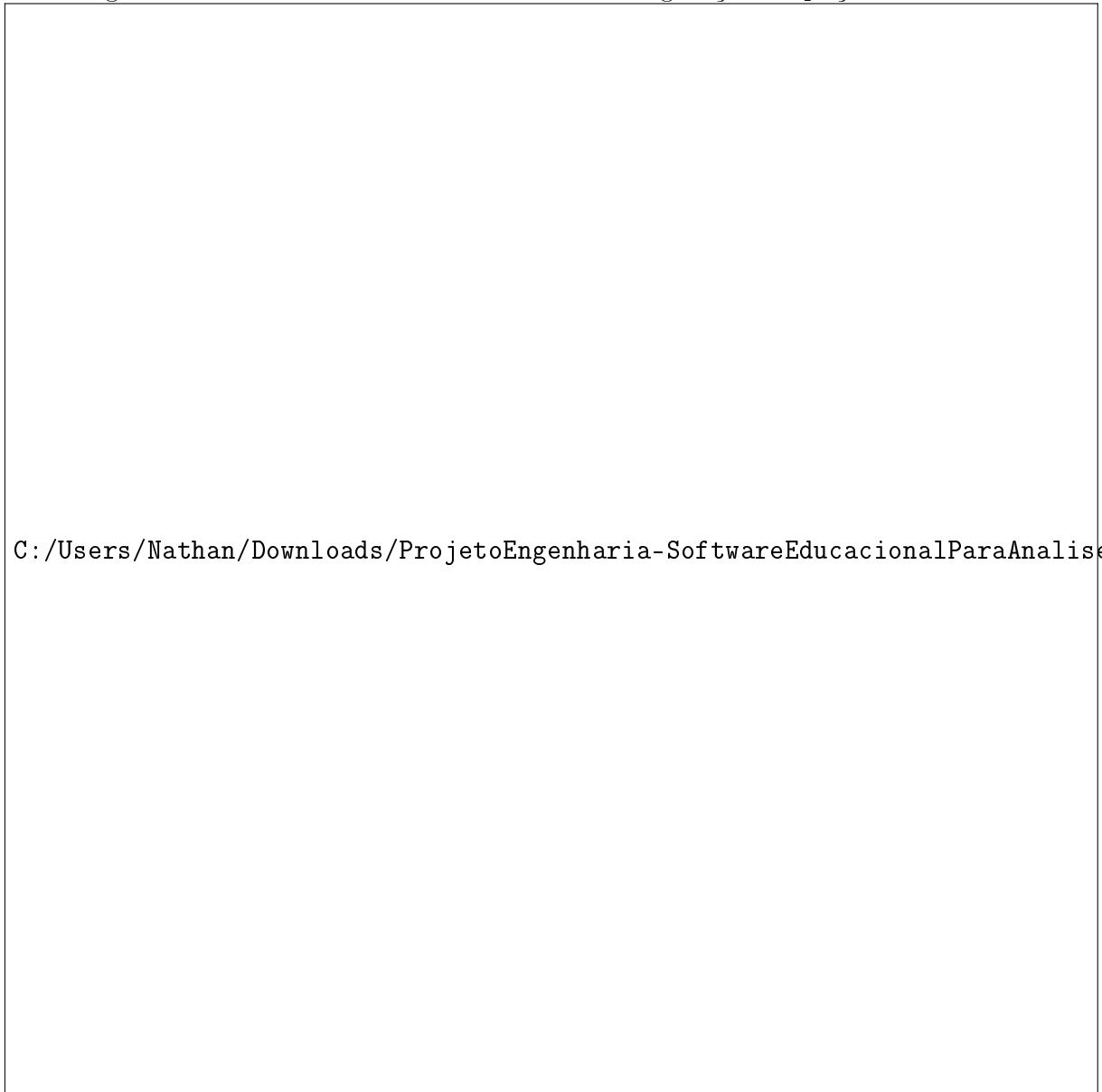
O usuário pode inserir os dados do poço e dos fluidos diretamente pela interface do programa. Para isso, basta seguir os seguintes passos:

- Acesse o menu lateral e selecione a opção Configurar Poço.
- Escolha entre as seguintes funcionalidades:
 - Criar Poço: permite definir propriedades como profundidade total, diâmetro do poço, presença de revestimentos e pressão na superfície.

- Adicionar Fluido: possibilita configurar fluidos específicos, definindo valores de densidade (lbm/gal) e viscosidade (cP), além de outras propriedades associadas à faixa de atuação por profundidade.

A Figura 10.4 ilustra o caminho no menu da interface gráfica para acessar essas funcionalidades.

Figura 10.4: Acesso às funcionalidades de configuração do poço e dos fluidos



Fonte: Produzido pelo autor.

Carregamento de Arquivos de Dados (.dat)

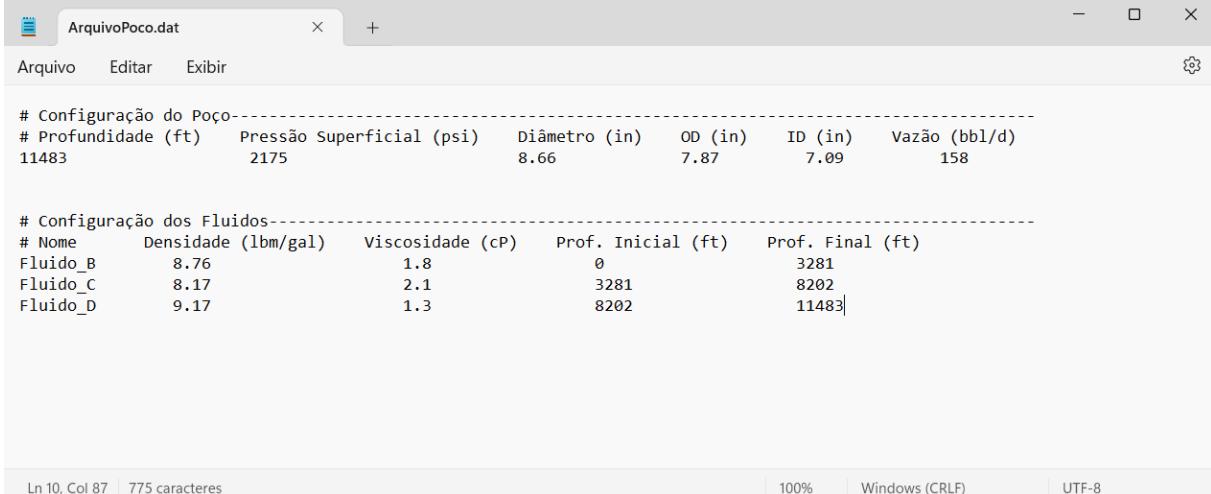
Alternativamente, o software permite o carregamento automático das configurações do poço e dos fluidos por meio de arquivos no formato .dat. Essa funcionalidade é especialmente útil para a reutilização de simulações ou integração com dados externos.

O arquivo .dat deve conter as informações organizadas em uma sequência específica:

- A primeira linha representa os dados do poço.
- As linhas subsequentes listam os fluidos, um por linha.

Cada linha deve seguir o formato padrão estabelecido pelo software, respeitando a ordem e as unidades exigidas. A 10.5. apresenta um exemplo de arquivo .dat estruturado corretamente e compatível com o sistema.

Figura 10.5: Exemplo de estrutura de arquivo .dat para importação de dados



```
# Configuração do Poço-----
# Profundidade (ft)      Pressão Superficial (psi)    Diâmetro (in)      OD (in)      ID (in)      Vazão (bbl/d)
11483                   2175                           8.66              7.87          7.09          158

# Configuração dos Fluidos-----
# Nome        Densidade (lbm/gal)  Viscosidade (cP)  Prof. Inicial (ft)  Prof. Final (ft)
Fluido_B     8.76                  1.8             0                  3281
Fluido_C     8.17                  2.1             3281              8202
Fluido_D     9.17                  1.3             8202              11483|
```

Ln 10, Col 87 | 775 caracteres | 100% | Windows (CRLF) | UTF-8

Fonte: Produzido pelo autor.

Dessa forma após configurar o poço o usuário poderá explorar as funcionalidade do software descritas na Seção 10.3.

Módulo 2 – Análise de Tensões na Coluna O segundo módulo do software é voltado para o estudo dos efeitos mecânicos na coluna de completamento, com foco nos deslocamentos axiais (ΔL) provocados por variações térmicas e de pressão. Esse módulo permite:

- **Configuração de Condições Iniciais e Finais:** entrada de temperaturas e profundidades associadas aos extremos da coluna.
- **Simulação de Efeitos Físicos:** como efeito balão, atuação do pistão, presença do packer e influência do crossover.
- **Cálculo de Cargas Axiais:** estimativas de carga nas condições de coluna livre ou fixa, além de simulações com restauração de força.
- **Visualização dos Resultados:** geração de gráficos de temperatura versus profundidade e esquema da coluna com dados associados.

Assim como no Módulo 1, o usuário também pode configurar as propriedades do poço e dos trechos diretamente pela interface gráfica, ou ainda importar um arquivo .dat contendo os dados necessários para inicialização da simulação. 10.6

Figura 10.6: Exemplo de estrutura de arquivo .dat para importação de dados



C:/Users/Nathan/Downloads/ProjetoEngenharia-SoftwareEducacionalParaAnaliseESolucao

Fonte: Produzido pelo autor.

Referências Bibliográficas

- BUENO, André Duarte. 2003. *Programação orientada a objeto com c++*. Novatec. 43
- HALLIDAY, David, & RESNICK, Jearld Walker. 2009. *Fundamentos de física, volume 2: gravitação, ondas e termodinâmica*. 33
- Jr., Adam T. Bourgoyné, Millheim, Keith E., Chenevert, Martim E., & Jr., F. S. Young. 1991. *Applied drilling engineering*. Society of Petroleum Engineers. 17, 29, 30, 31, 32, 35, 38, 148, 149, 150, 152
- Mitchell, Robert F., & Miska, Stefam Z. 2011. *Fundamentals of drilling engineering*. Society of Petroleum Engineers. 17, 32, 33, 34, 38, 39, 40