

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	56
6.5	Versão 1.1, Menu de interface do software	58
6.6	Versão 1.1, interface do modulo 01 do software	58
6.7	Versão 1.1, interface do modulo 02 software	59
8.1	Soluções geradas pelo código para modelo Newtoniano no poço considerando regime laminar	135
8.2	Soluções geradas pelo código para modelo Newtoniano no poço considerando regime turbulento	136

8.3	Soluções geradas pelo código para modelo Newtoniano no anular considerando regime laminar	137
8.4	Soluções geradas pelo código para modelo Newtoniano no anular considerando regime turbulento	138
8.5	Soluções geradas pelo código para modelo plástico de Bingham no poço considerando regime laminar	139
8.6	Soluções geradas pelo código para modelo plástico de Bingham no poço considerando regime turbulento	140
8.7	Soluções geradas pelo código para modelo plástico de Bingham no anular considerando regime laminar	141
8.8	Soluções geradas pelo código para modelo plástico de Bingham no anular considerando regime turbulento	142
8.9	Soluções geradas pelo código para modelo lei de potência no poço considerando regime laminar	143
8.10	Soluções geradas pelo código para modelo lei de potência no poço considerando regime turbulento	144
8.11	Soluções geradas pelo código para modelo lei de potência no anular considerando regime laminar	145
8.12	Soluções geradas pelo código para modelo lei de potência no anular considerando regime turbulento	146
8.13	Soluções geradas pelo código para pressão hidrostática no fundo do poço .	147
8.14	Gráfico pressão hidrostática por profundidade	148
8.15	Terminal mostrando a funcionalidade armazenar os resultados	149
8.16	Terminal mostrando a funcionalidade armazenar os resultados	149
9.1	Logo e documentação do <i>software</i>	151
9.2	Código fonte da classe CSimuladorPoco, no <i>Doxxygen</i>	151
10.1	Versão 1.1, Menu de interface do software	152
10.2	Versão 1.1, interface do modulo 01 do software	153
10.3	Versão 1.1, interface do modulo 02 software	153
10.4	Acesso às funcionalidades de configuração do poço e dos fluidos	155
10.5	Exemplo de estrutura de arquivo .dat para importação de dados	156
10.6	Exemplo de estrutura de arquivo .dat para importação de dados	157

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
8.1	Configuração do poço turbulento	133
8.2	Configuração do poço turbulento	134
8.3	Configuração do poço laminar	134
8.4	Configuração do poço turbulento	134

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	60
7.1 Código-Fonte (modelo)	60
7.1.1 Código Qt (interface)	117
8 Resultados	133
8.1 Propriedades da simulação	133
8.1.1 Configurações para o regime turbulento	133
8.1.2 Configurações para o regime laminar	134
8.2 Testes	134
8.2.1 Validação da perda de fricção pelo modelo Newtoniano no poço	134
8.2.2 Validação da perda de fricção pelo modelo Newtoniano no anular	136
8.2.3 Validação da perda de fricção pelo modelo Bingham no pôco	138
8.2.4 Validação da perda de fricção pelo modelo Bingham no anular	140
8.2.5 Validação da perda de fricção pelo modelo de potência no pôco	142
8.2.6 Validação da perda de fricção pelo modelo de potência no anular	144
8.2.7 Validação da pressão hidrostática no fundo do poço	146
8.2.8 Validação da funcionalidade impressão	148

9 Documentação	150
9.1 Documentação do usuário	150
9.2 Documentação do desenvolvedor	150
10 Manual do usuário	152
10.1 Instalação	152
10.1.1 Dependências	152
10.2 Interface gráfica	152
10.3 Funcionalidades	154
10.3.1 Módulo 1 – Hidráulica de Perfuração	154
Referências Bibliográficas	158

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:

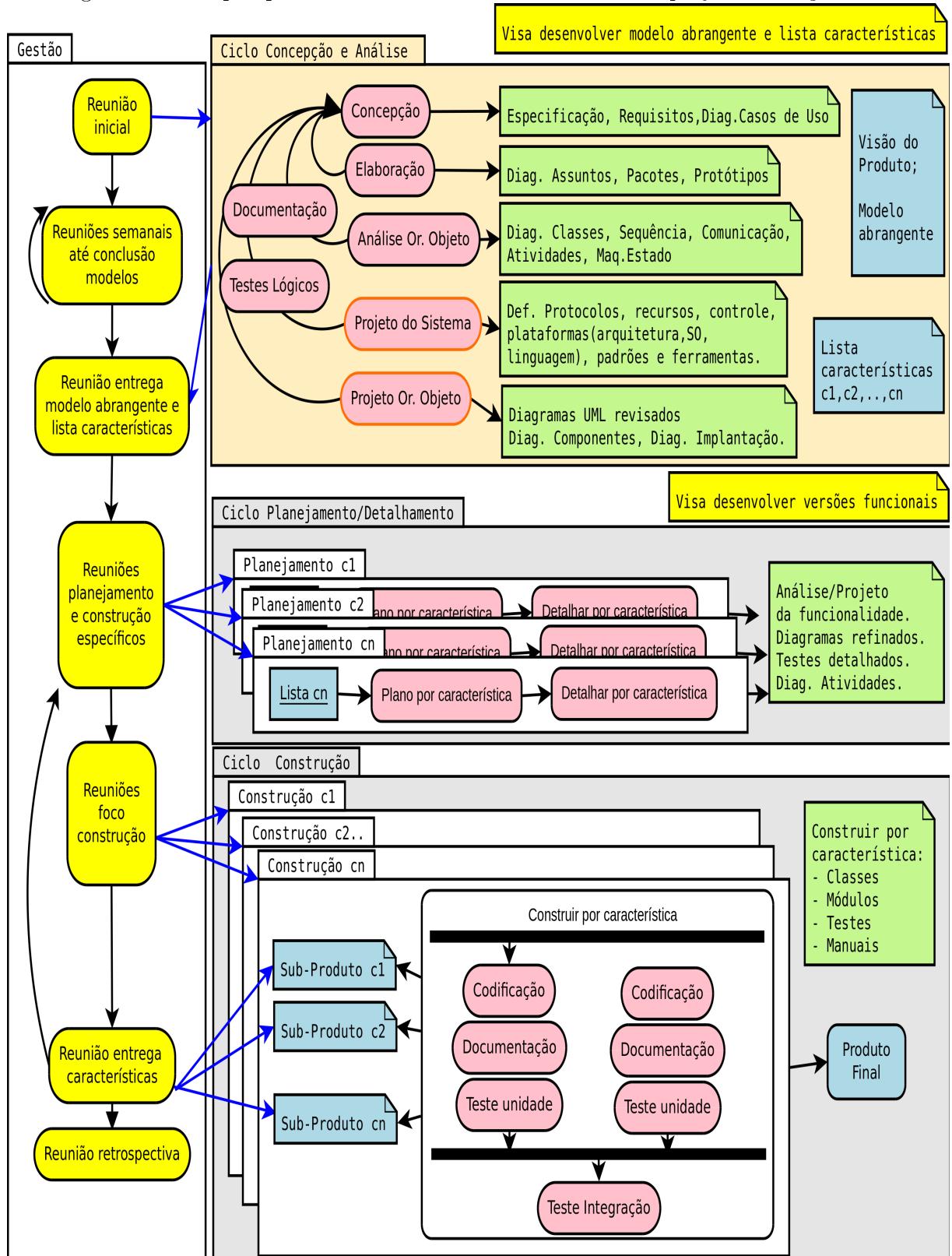
- Desenvolver um software capaz de analisar e calcular as principais equações que sustentam os fundamentos da disciplina de Engenharia de Poço, favorecendo a consolidação do conhecimento teórico adquirido em sala de aula.
- Objetivos específicos:
 - Modelar física e matematicamente os problemas abordados, 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

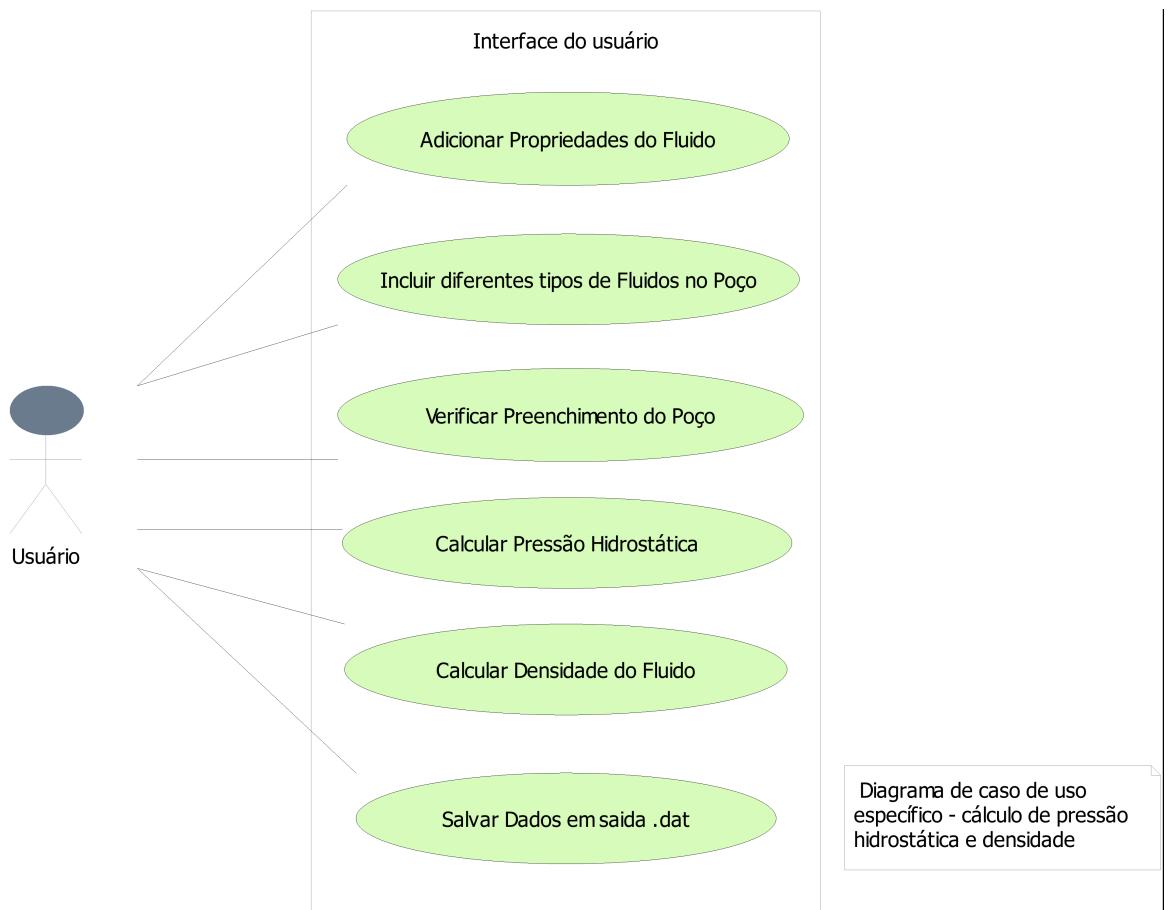
O caso de uso específico da Figura 2.2 mostra o cenário onde o usurário deseja calcular a pressão hidrostática e a densidade dos fluidos configurados no poço.

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



Fonte: Produzido pelo autor.

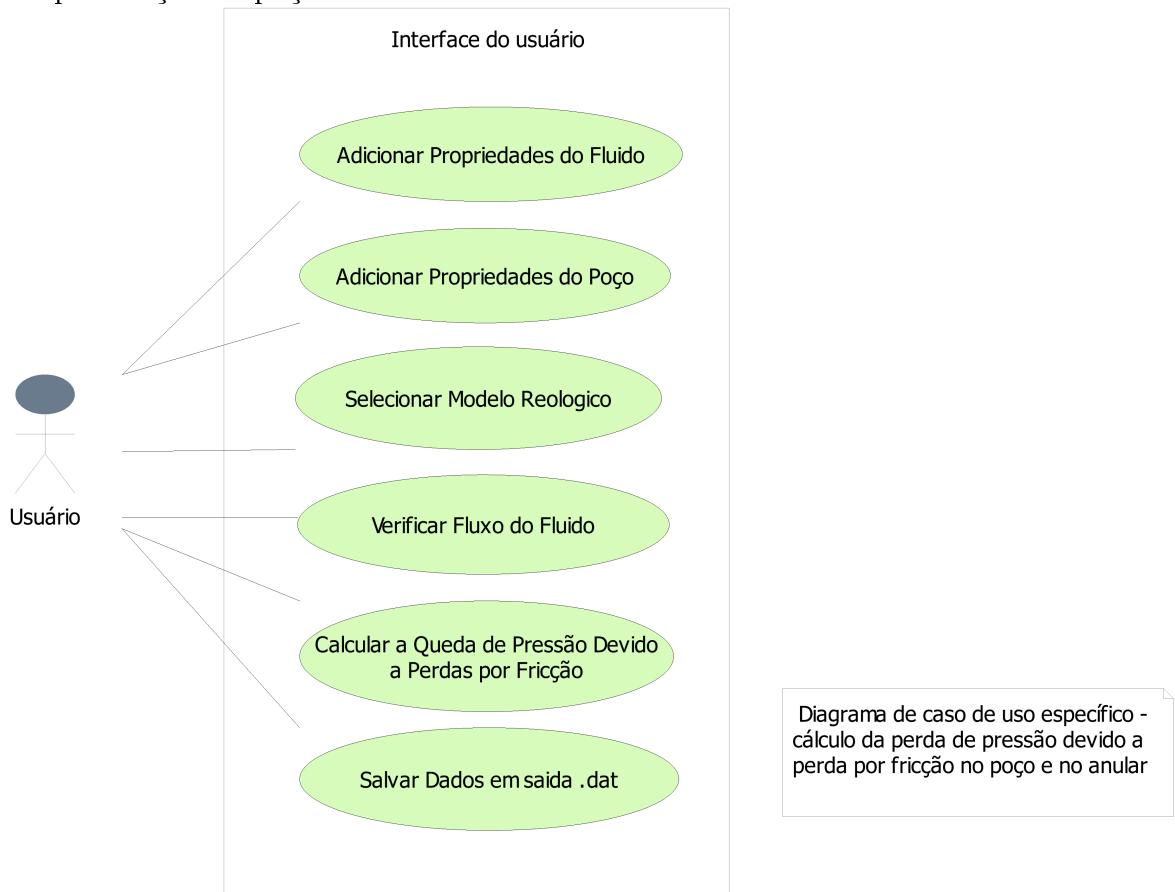
Figura 2.2: Diagrama de caso de uso específico: cálculo de pressão hidrostática e densidade



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

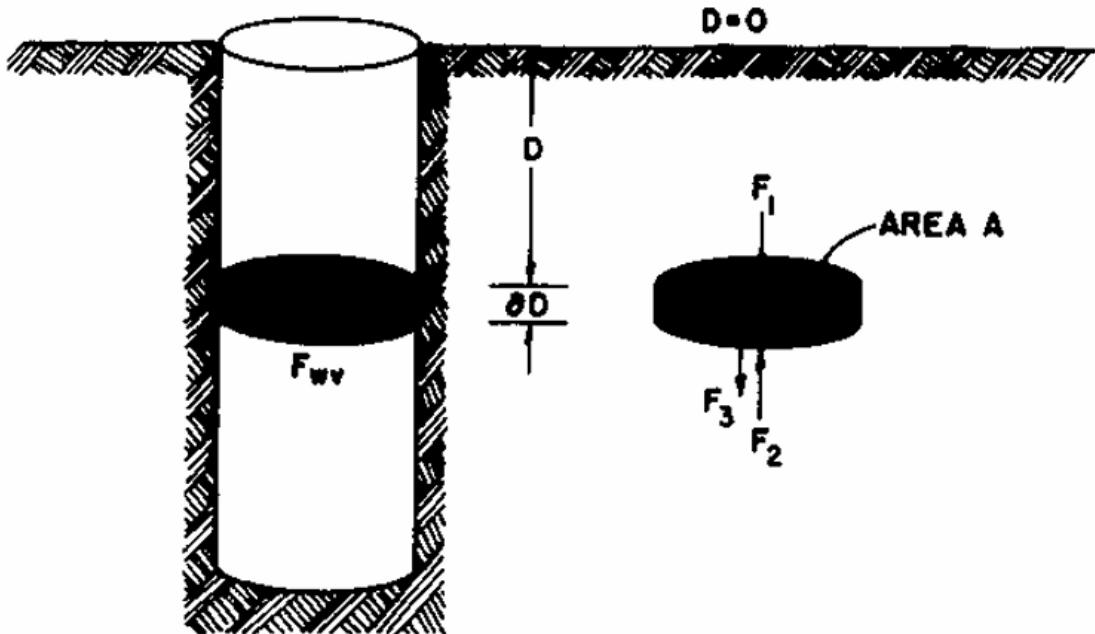
Na engenharia de perfuração, um fluido de perfuração possui três funções principais: transportar cascalho, prevenir o influxo de fluidos e manter a estabilidade do poço. Para cumprir essas funções, o fluido depende do seu escoamento na tubulação e das pressões associadas. Para que o engenheiro possa 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:

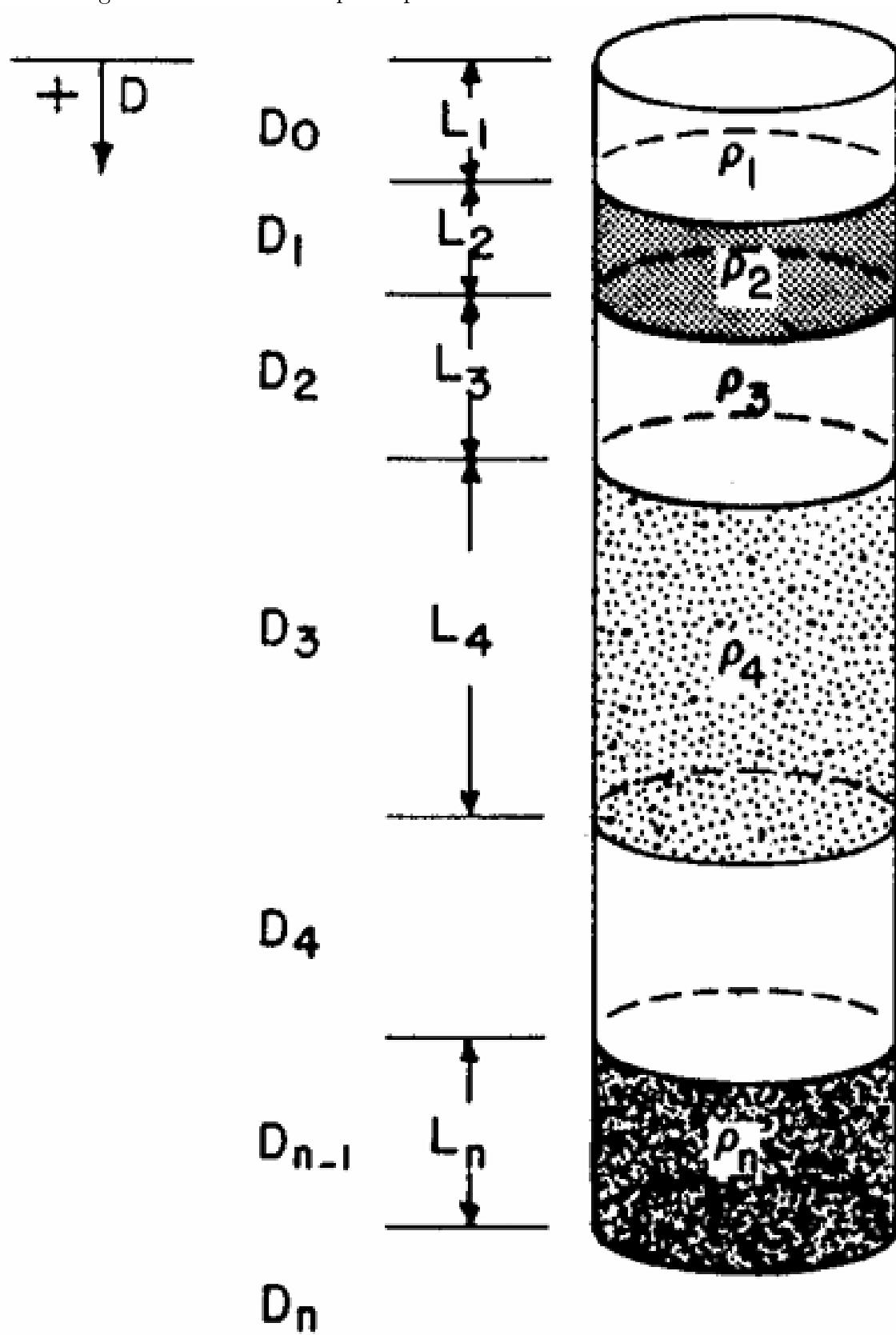
$$p = p_0 \exp \left(\frac{M \Delta Z}{1544 z T} \right) \quad (3.4)$$

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.

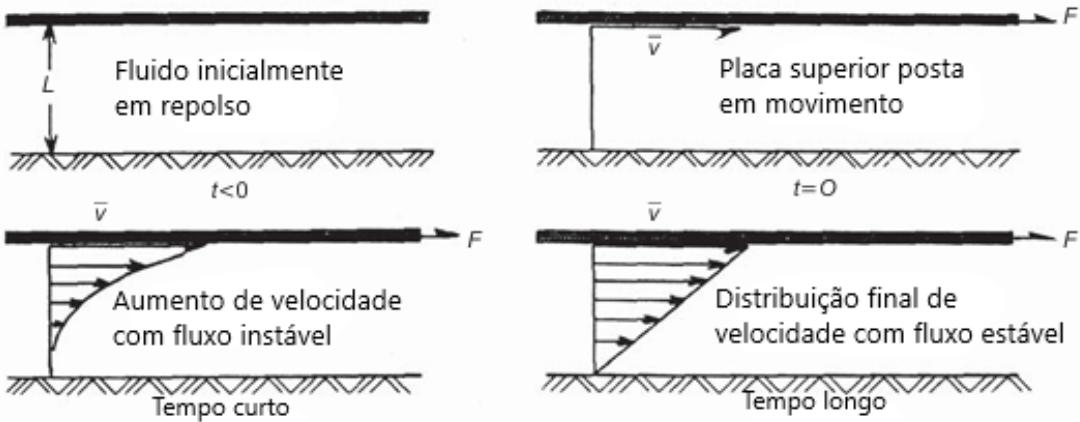
Na maioria dos casos, as propriedades elásticas dos fluidos de perfuração e seus efeitos durante o escoamento no poço são desprezíveis, sendo consideradas apenas as forças viscosas nos cálculos. Entretanto, com o avanço tecnológico, lamas cada vez mais complexas vêm sendo desenvolvidas, e, portanto,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

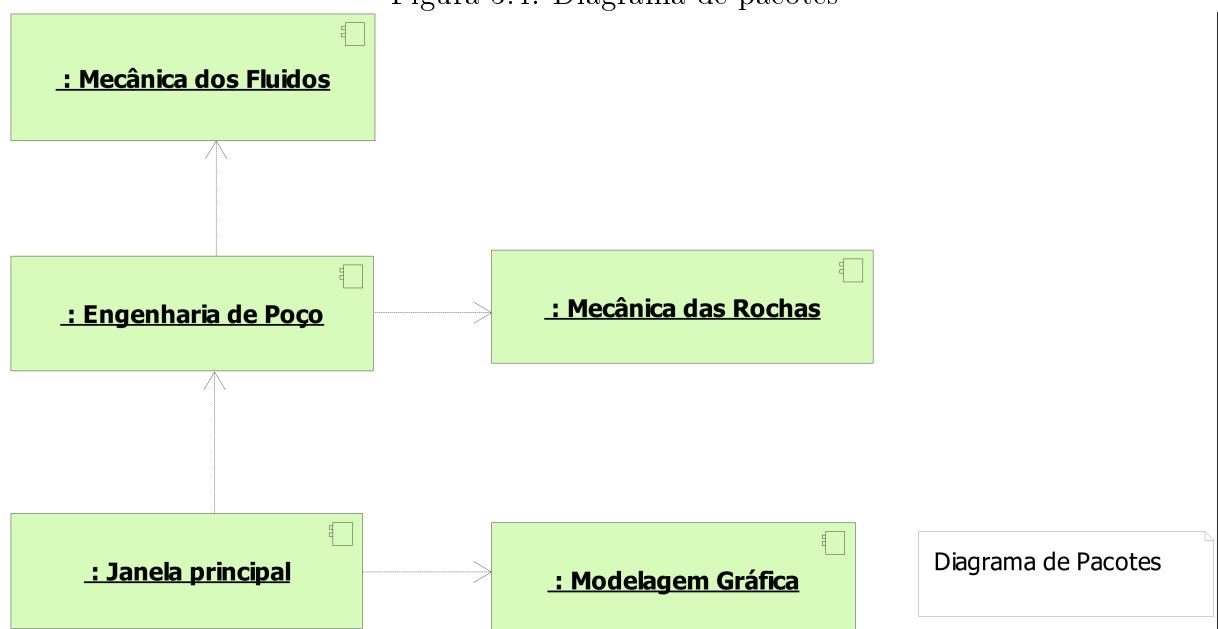


Diagrama de Pacotes

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

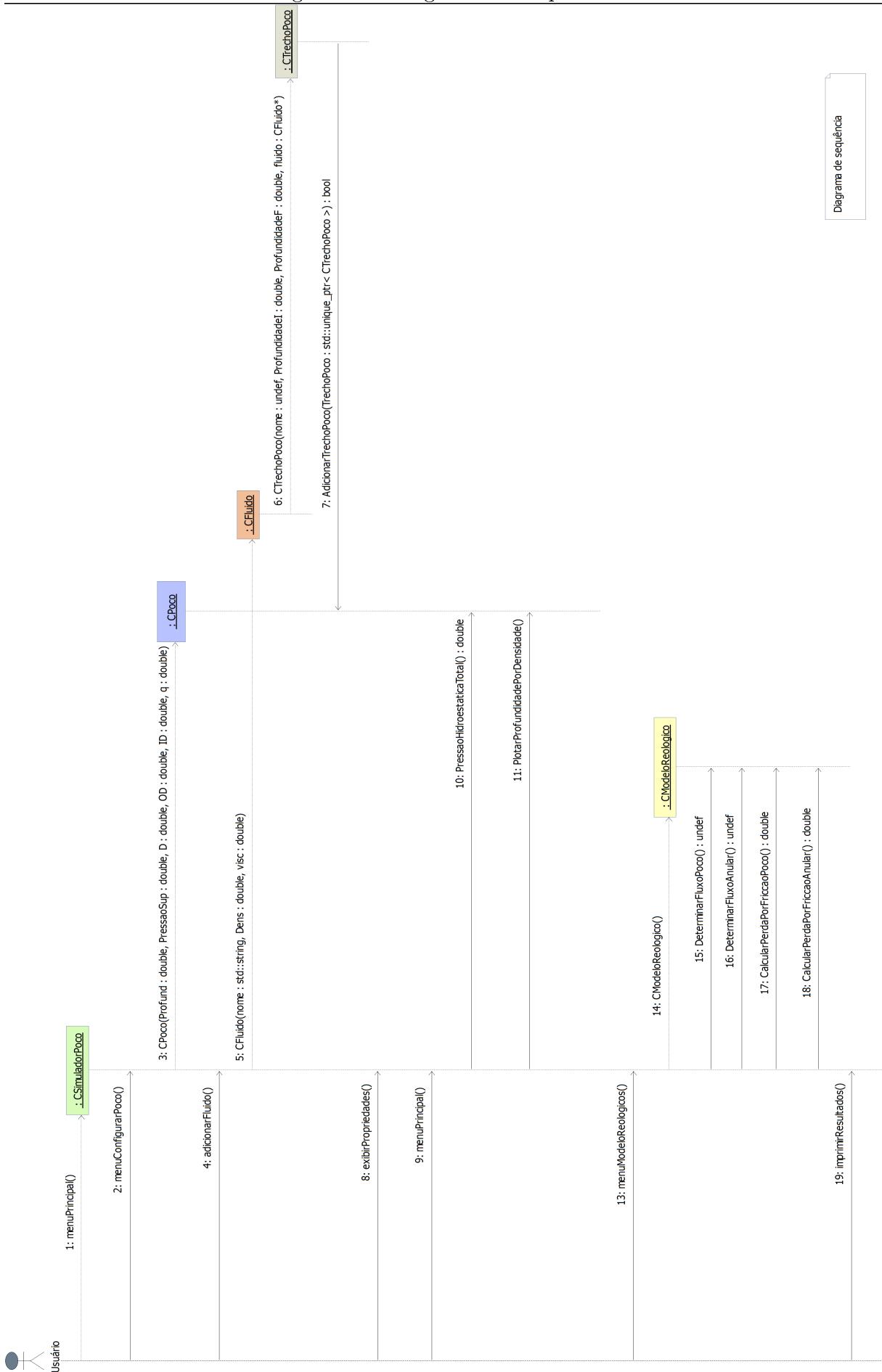
O diagrama de sequência descreve a interação entre os objetos do sistema e os elementos externos, evidenciando a ordem temporal das mensagens trocadas durante a execução de uma funcionalidade. Ele apresenta o fluxo de controle do sistema de forma cronológica, detalhando as chamadas de métodos e as respostas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

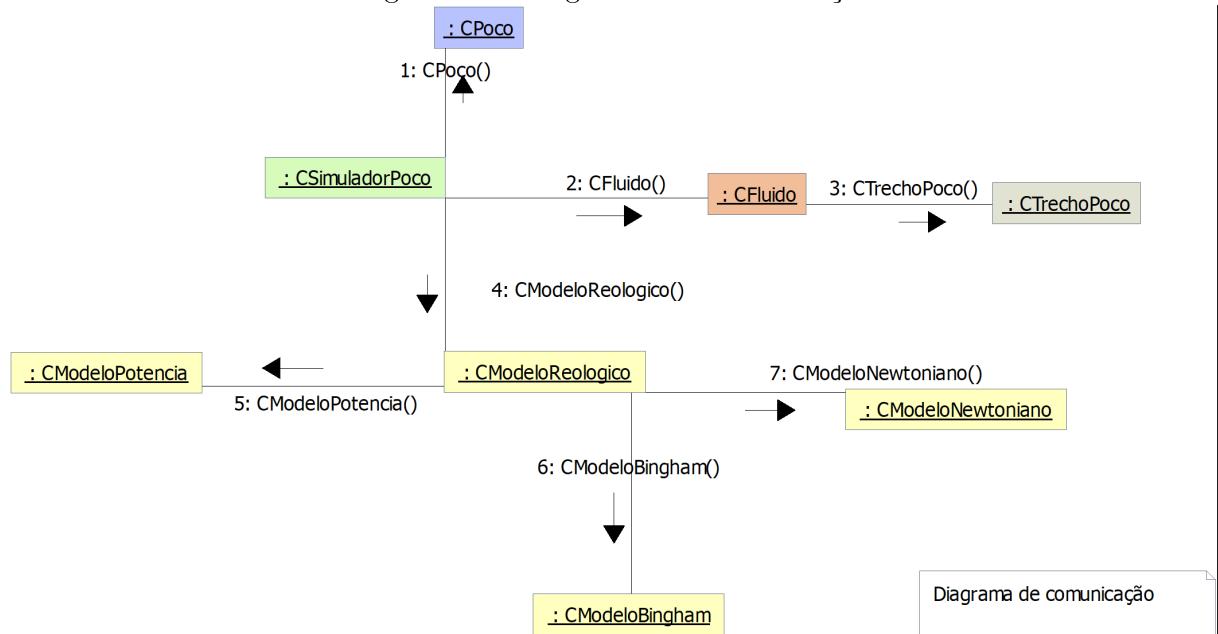


Fonte: Produzido pelo autor.

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.

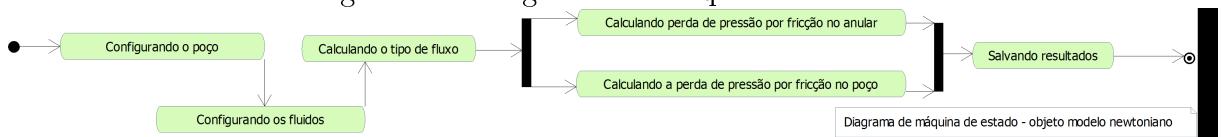
4.4 Diagrama de Máquina de Estado

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

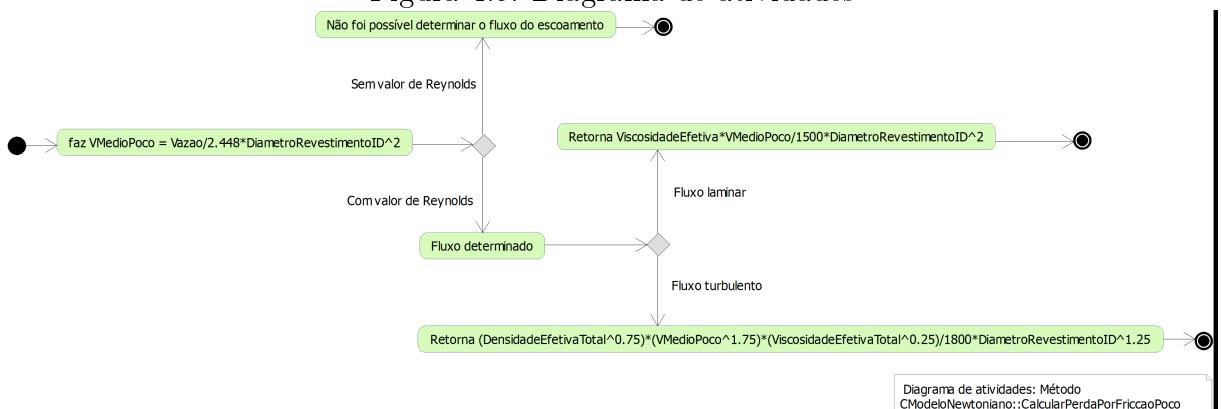
O diagrama de atividades apresentado descreve, em detalhe, a execução de uma atividade específica do sistema. No caso em questão, é representado o método **CalcularPerdaPorFriccaoPoco**, pertencente à classe **CModeloNewtoniano**.

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.

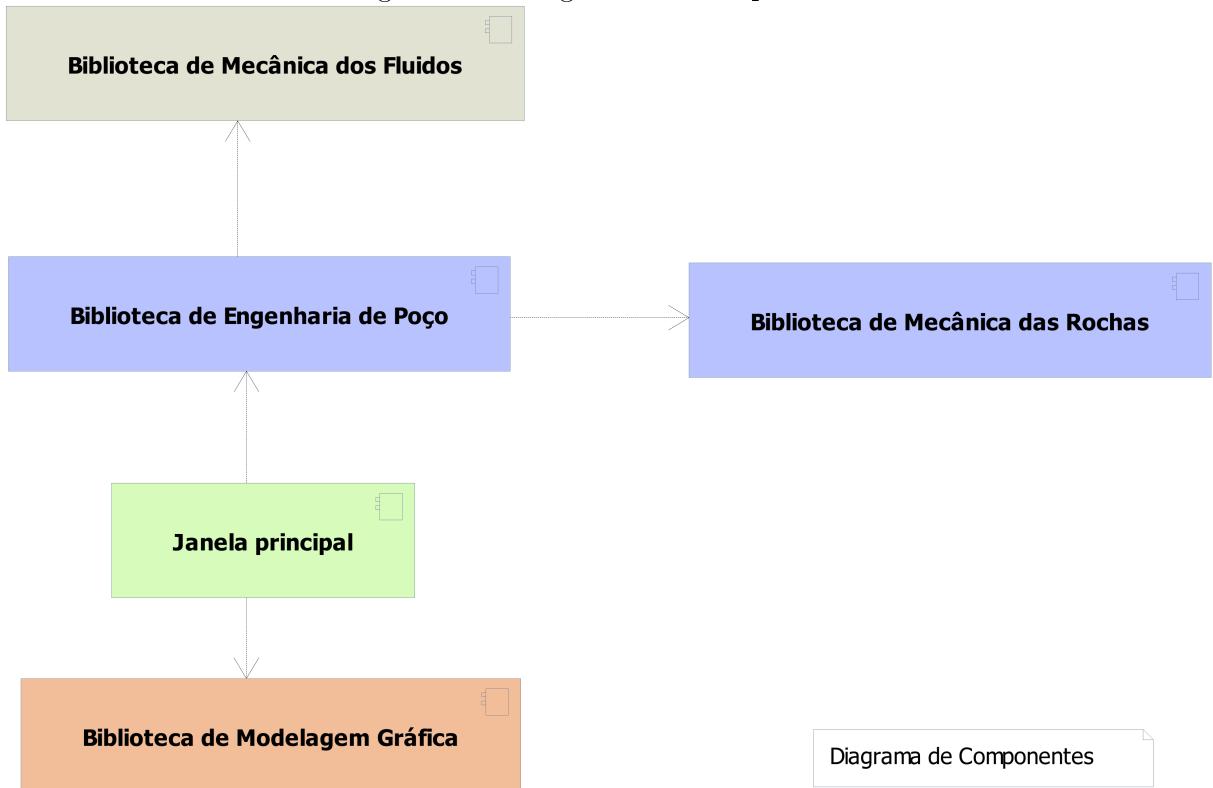
A linguagem escolhida foi o C++, em virtude de suas características que a tornam especialmente adequada para o desenvolvimento de aplicações técnicas e científicas. Os principais fatores que motivaram essa escolha incluem:

- 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

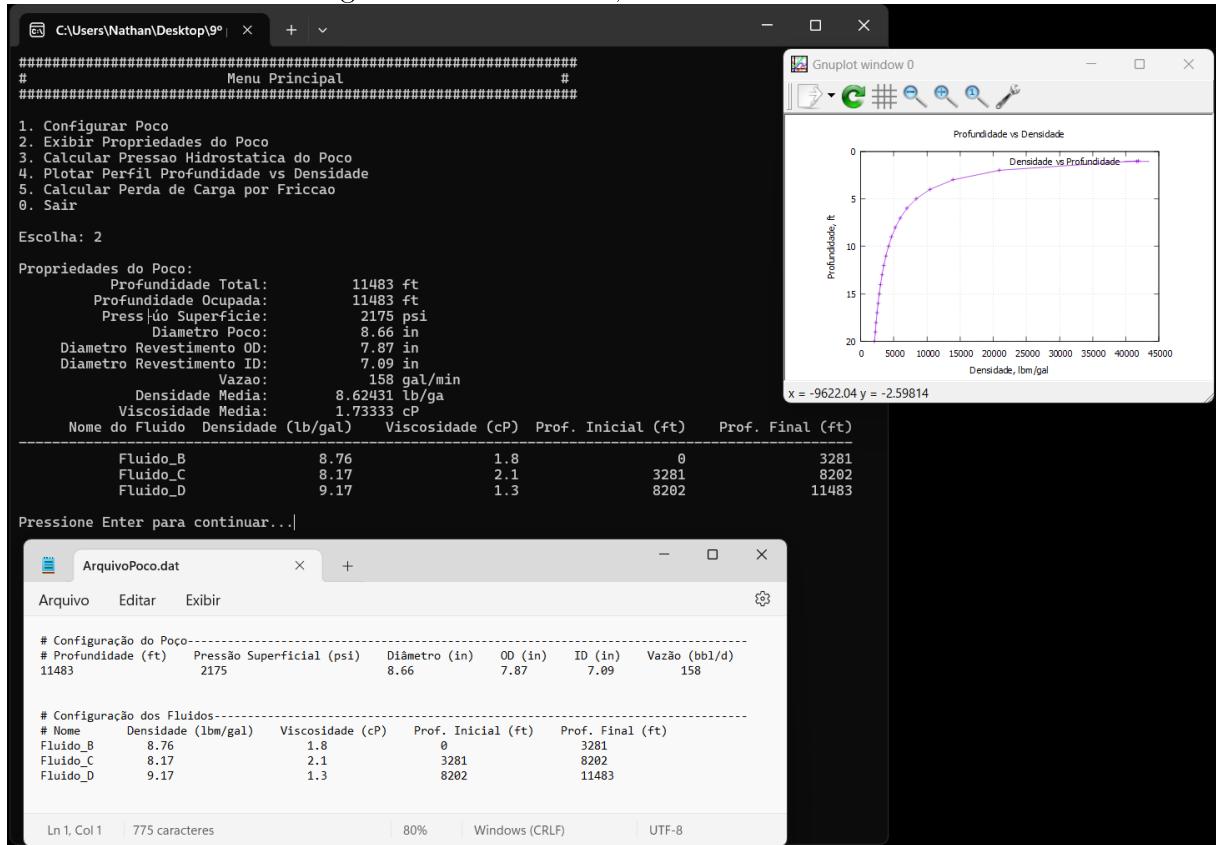
Apresenta-se neste capítulo as versões do software desenvolvido.

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.

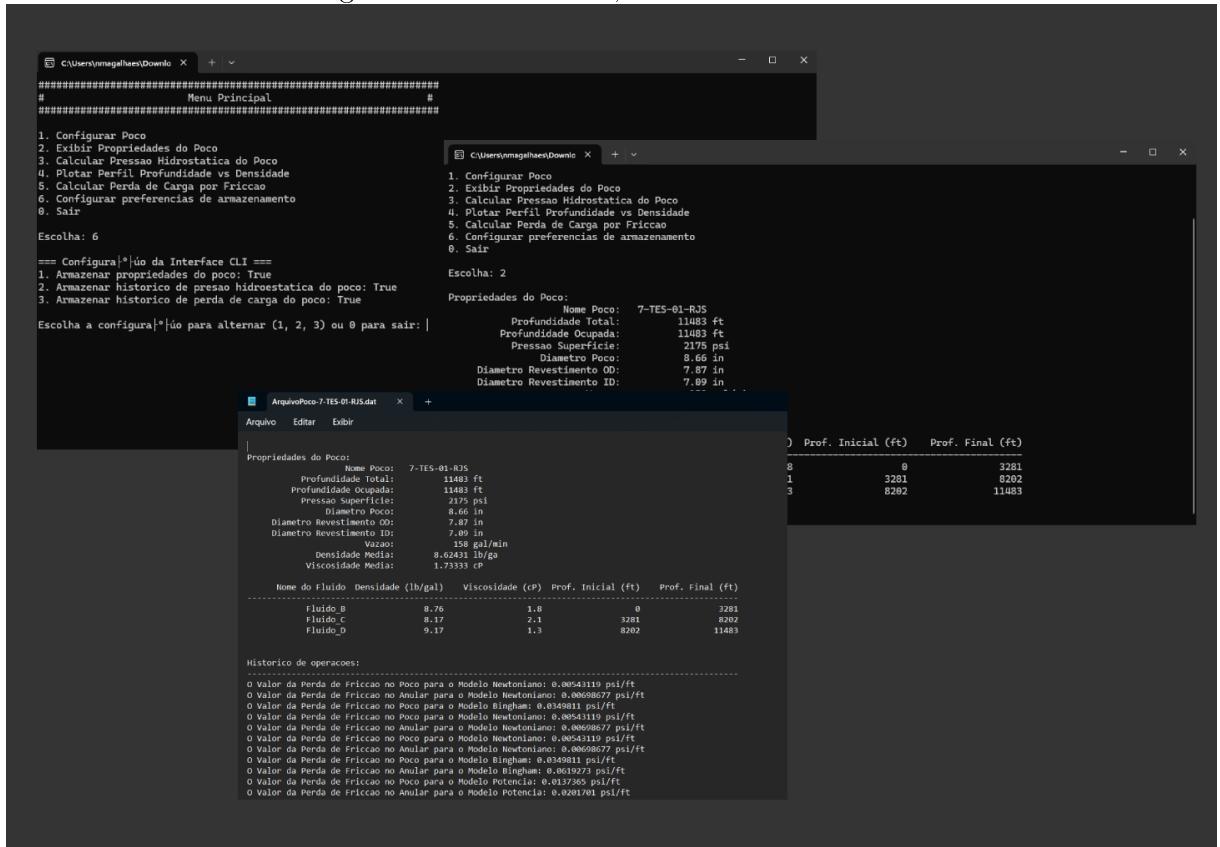
Principais aprimoramentos implementados:

- 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

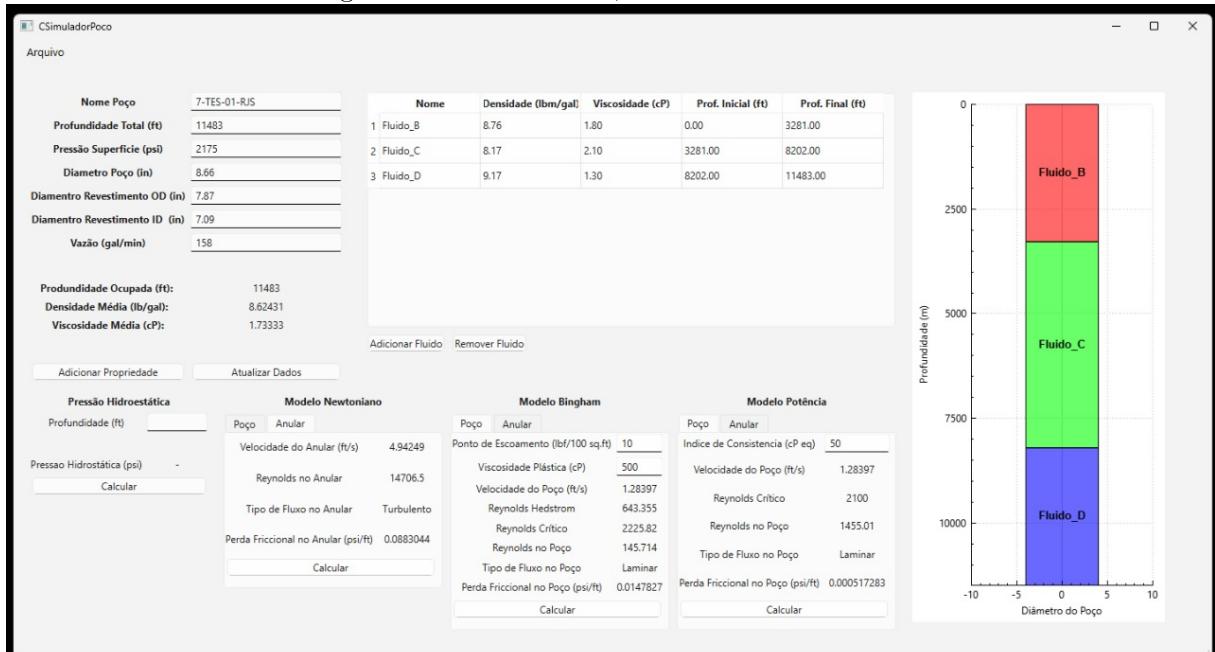
intuitiva.

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.

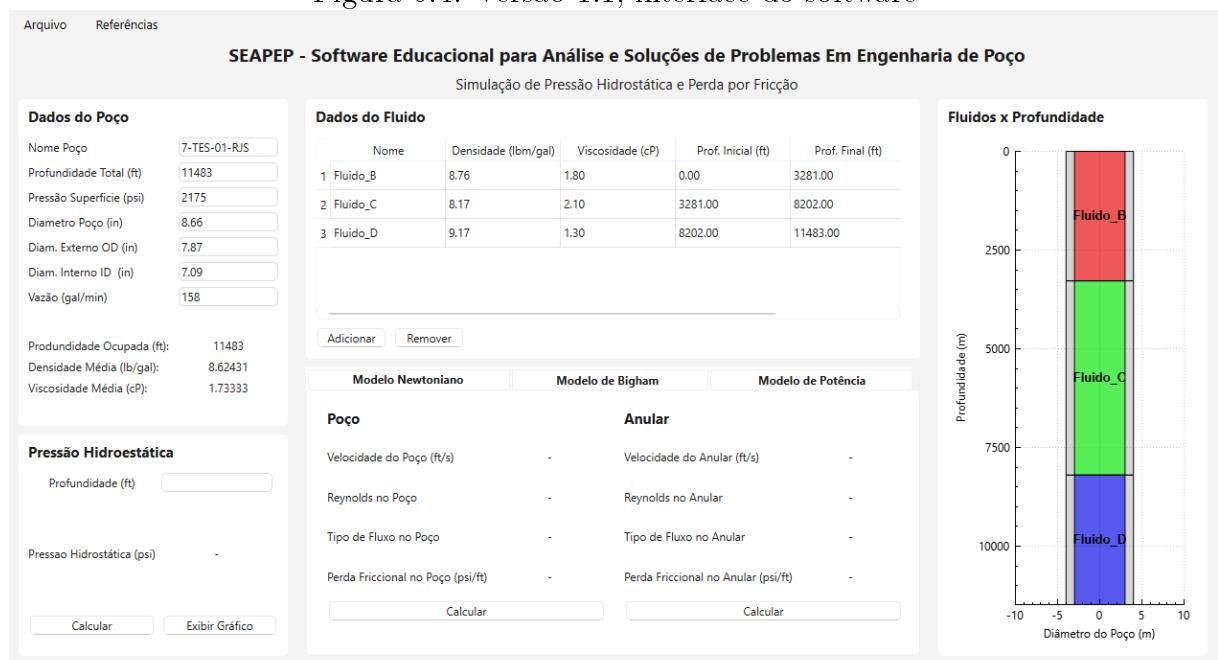
Além disso, foi implementada uma barra de tarefas com novas funcionalidades organizadas em menus acessíveis, incluindo:

- **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.

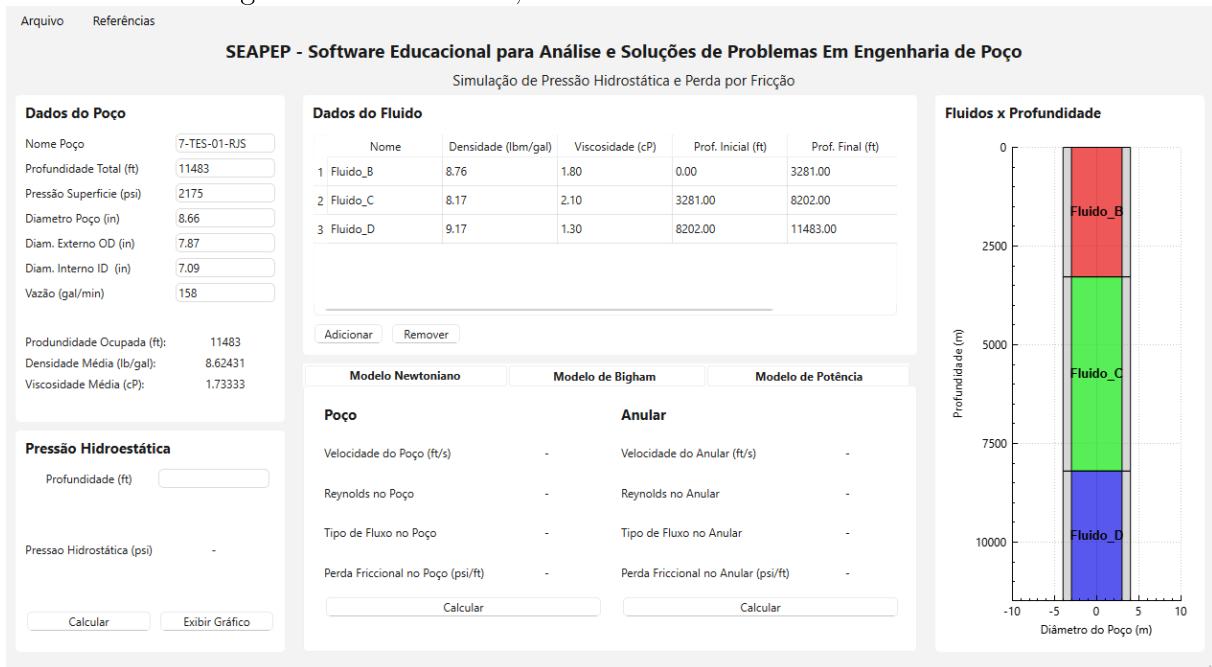
Com essa versão, o sistema passa a incorporar, além da análise hidráulica, uma abordagem mecânica de simulação, ampliando significativamente seu escopo didático e técnico na área de Engenharia de Poço.

Figura 6.5: Versão 1.1, Menu de interface do software



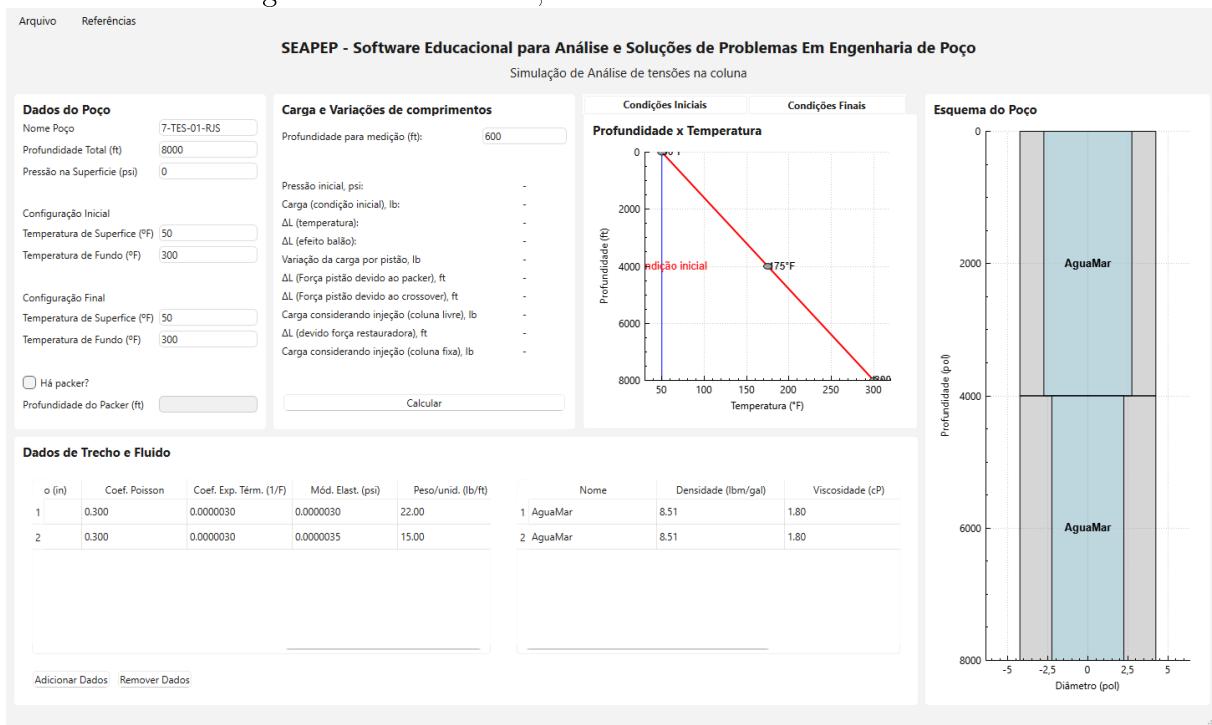
Fonte: Produzido pelo autor.

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



Fonte: Produzido pelo autor.

Figura 6.7: Versão 1.1, interface do modulo 02 software



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);
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>

```

```

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
14    return reynoldsPoco;
15}
16
17 // Mesmo calculo, mas permitindo passar a viscosidade como
18 // parametro
19 double CModeloReologico::DeterminarReynoldsPoco(double viscosidade)
20 {
21    reynoldsPoco = (928 * poco->DensidadeEfetivaTotal() *
22                    vMediaPoco * poco->DiametroRevestimentoID()) /
23                    viscosidade;
24
25    return reynoldsPoco;
26}
27
28 // Calcula o numero de Reynolds no espaco anular
29 double CModeloReologico::DeterminarReynoldsAnular() {
30    reynoldsAnular = (757 * poco->DensidadeEfetivaTotal() *
31                       vMediaAnular *
32                           (poco->DiametroPoco() - poco->
33                               DiametroRevestimentoOD())) /
34                               poco->ViscosidadeEfetivaTotal();
35
36    return reynoldsAnular;
37}
38
39 // Mesmo calculo para o anular, com viscosidade recebida
40 // externamente
41 double CModeloReologico::DeterminarReynoldsAnular(double
42           viscosidade) {
43    reynoldsAnular = (757 * poco->DensidadeEfetivaTotal() *
44                       vMediaAnular *
45                           (poco->DiametroPoco() - poco->
46                               DiametroRevestimentoOD())) /
47                               viscosidade;

```

```

34     return reynoldsAnular;
35 }
36
37 // Calcula a velocidade m dia do fluido no interior da coluna (poco)
38 double CModeloReologico::DeterminarVelocidadeMediaPoco() {
39     vMediaPoco = poco->Vazao() / (2.448 * std::pow(poco->
        DiametroRevestimentoID(), 2));
40     return vMediaPoco;
41 }
42
43 // Calcula a velocidade m dia no espaço anular entre a coluna e o revestimento
44 double CModeloReologico::DeterminarVelocidadeMediaAnular() {
45     vMediaAnular = poco->Vazao() /
        (2.448 * (std::pow(poco->DiametroPoco(), 2) -
                  std::pow(poco->DiametroRevestimentoOD(), 2)));
46     ;
47     return vMediaAnular;
48 }
49
50 // Calcula o fator de fricção usando a equação implícita de Fanning com Newton-Raphson
51 double CModeloReologico::DeterminarFatorFriccao(double re) {
52     // Estimativa inicial baseada em escoamento laminar (não é usada diretamente, mas serve como chute)
53     auto laminar_fator = [] (double re) {
54         return 0.0791 / std::pow(re, 0.25);
55     };
56
57     // Equação de Fanning: 1/sqrt(f) - 4log10(Re*sqrt(f)) + 0.395 = 0
58     auto f = [re] (double x) {
59         return 1 / std::sqrt(x) - 4 * std::log10(re * std::sqrt(x))
60         + 0.395;
61     };
62
63     // Derivada da equação de Fanning
64     auto df = [] (double x) {
65         return -0.5 / (x * std::sqrt(x)) - (2 / (x * std::log(10)))
66         ;
67     };

```

```

66
67     // Metodo de Newton-Raphson para resolver a equacao
68     auto newtonRaphson = [&](double x0, double tol = 1e-6, int
69     max_iter = 1000000) {
70         double x = x0;
71         for (int i = 0; i < max_iter; i++) {
72             double fx = f(x);
73             double dfx = df(x);
74             if (std::abs(fx) < tol) {
75                 return x;
76             }
77             x -= fx / dfx;
78         }
79         return x;
80     };
81
82     // Chute inicial com base no fator para fluxo laminar
83     double x0 = laminar_fator(re);
84
85     // Retorna o valor resolvido
86     return newtonRaphson(x0);
87 }
```

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 }
```

```

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
27 private slots:
28     // esses s o os slots que reagem aos botoes da interface
29
30     void on_btnAdicionarPropriedades_clicked();           // adiciona as
   propriedades termicas e mecanicas do fluido
31     void on_btnAtualizarDados_clicked();                  // atualiza os
   dados na tela com base no objeto do po o
32     void on_btnAdicionarTrecho_clicked();                // adiciona um
   novo trecho de tubulacao ao po o
33     void makePlotTemperatura(double TempInicial, double TempFinal,
   double profundidade, QCustomPlot* plot); // gera grafico de
   temperatura com profundidade
34     void on_btnRemoverFluido_clicked();                  // remove um
   fluido da tabela e do po o
35     void on_btnRemoverTrecho_clicked();                  // remove um
   trecho da tubulacao
36     void makePlotPoco();                                // desenha o
   perfil visual do po o
37     void on_btnCalcularVariacoes_clicked();            // calcula L ,
   efeito balao, forca etc
38
39     void on_actionArquivo_Dat_triggered();              // importa
   dados do arquivo .dat
40     void EditarDadosPoco();                            // edita os
   dados gerais do po o (nome, pressao etc)
41
42     // opcoes do menu da interface
43     void on_actionNova_Simula_o_triggered();
44     void on_actionExportar_Como_Imagen_triggered();

```

```

45     void on_actionSobre_o_SEEP_triggered();
46
47 private:
48     Ui::CSimuladorPerdaTubulacao *ui; // ponteiro pra interface
49         gerada pelo Qt Designer
50
51     // ponteiros para os objetos principais que compoem o modelo do
52         po o
53     std::shared_ptr<CObjetoPoco> poco = nullptr; // representa o po o como um todo
54
55     std::shared_ptr<CTrechoPoco> trechoPoco = nullptr; // trecho individual de tubulacao
56     std::shared_ptr<CFluido> fluido = nullptr; // fluido associado aos trechos
57
58     // modelos reologicos usados pra calcular propriedades de
59         escoamento
60     std::shared_ptr<CModeloNewtoniano> modeloNewtoniano = nullptr;
61     std::shared_ptr<CModeloBingham> modeloBingham = nullptr;
62     std::shared_ptr<CModeloPotencia> modeloPotencia = nullptr;
63 };
64
65 #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
6 #include <iostream> // para std::cerr e std::endl
7 #include <fstream> // para std::ifstream
8
9 CSimuladorPerdaTubulacao::CSimuladorPerdaTubulacao(QWidget *parent)
10    : QMainWindow(parent)
11    , ui(new Ui::CSimuladorPerdaTubulacao)
12 {
13     ui->setupUi(this);
14
15     // Inicialmente, desativa o lineEdit

```

```

16 ui->editProfundidadePacker->setEnabled(false);

17
18 // Conecta o sinal stateChanged do checkBox fun o lambda
19 connect(ui->checkBoxPacker, &QCheckBox::checkStateChanged, this
20     , [=](int state){
21         ui->editProfundidadePacker->setEnabled(state == Qt::Checked
22             );
23     });
24
25 // Sinal para alterar os das caixas
26 connect(ui->editNomePoco, &QLineEdit::textChanged, this, &
27     CSimuladorPerdaTubulacao::EditarDadosPoco);
28 connect(ui->editProfundidadeTotal, &QLineEdit::textChanged,
29     this, &CSimuladorPerdaTubulacao::EditarDadosPoco);
30 connect(ui->editPressaoSup, &QLineEdit::textChanged, this, &
31     CSimuladorPerdaTubulacao::EditarDadosPoco);
32 connect(ui->editTemperaturaSuperiorInicial, &QLineEdit::
33     textChanged, this, &CSimuladorPerdaTubulacao:::
34     EditarDadosPoco);
35 connect(ui->editTemperaturaFundoInicial, &QLineEdit::
36     textChanged, this, &CSimuladorPerdaTubulacao:::
37     EditarDadosPoco);
38 connect(ui->editTemperaturaSuperiorFinal, &QLineEdit::
39     textChanged, this, &CSimuladorPerdaTubulacao:::
40     EditarDadosPoco);
41 connect(ui->editTemperaturaFundoFinal, &QLineEdit::textChanged,
42     this, &CSimuladorPerdaTubulacao::EditarDadosPoco);
43 connect(ui->editProfundidadePacker, &QLineEdit::textChanged,
44     this, &CSimuladorPerdaTubulacao::EditarDadosPoco);

45
46 // iniciar com botões desativado
47 ui->btnAdicionarTrecho->setEnabled(false);
48 ui->btnRemoverTrecho->setEnabled(false);
49 ui->btnCalcularVariacoes->setEnabled(false);

50
51 // abrir janela no meio do monitor
52 QScreen *screen = QGuiApplication::primaryScreen();
53 QRect screenGeometry = screen->geometry();

54 int x = (screenGeometry.width() - this->width()) / 2;
55 int y = (screenGeometry.height() - this->height()) / 2;

```

```

45     this->move(x, y);
46
47     makePlotPoco();
48 }
49
50 CSimuladorPerdaTubulacao::~CSimuladorPerdaTubulacao()
51 {
52     delete ui;
53 }
54
55 void CSimuladorPerdaTubulacao::EditarDadosPoco() {
56     QString nome = ui->editNomePoco->text();
57     bool ok1, ok2, ok3, ok4, ok5, ok6, ok7;
58     double profund = ui->editProfundidadeTotal->text().toDouble(&
59                     ok1);
60     double pressao = ui->editPressaoSup->text().toDouble(&ok2);
61     double temperaturaSuperiorInicial = ui->
62         editTemperaturaSuperiorInicial->text().toDouble(&ok3);
63     double temperaturaFundoInicial = ui->
64         editTemperaturaFundoInicial->text().toDouble(&ok4);
65     double temperaturaSuperiorFinal = ui->
66         editTemperaturaSuperiorFinal->text().toDouble(&ok5);
67     double temperaturaFundoFinal = ui->editTemperaturaFundoFinal->
68         text().toDouble(&ok6);
69     double profundPacker = ui->editProfundidadePacker->text().
70         toDouble(&ok7);
71
72     if (!nome.isEmpty() && ok1 && ok2 && ok3 && ok4 && ok5 && ok6
73         && ok7) {
74         if (!poco) {
75             // Cria o p o o
76             poco = std::make_unique<CObjetoPoco>(
77                 CObjetoPoco::CriarParaModulo02(nome.toStdString(),
78                     profund, pressao, temperaturaSuperiorInicial,
79                     temperaturaFundoInicial,
80                     temperaturaSuperiorFinal, temperaturaFundoFinal,
81                     profundPacker)
82             );
83
84             ui->btnAdicionarTrecho->setEnabled(true);
85             ui->btnRemoverTrecho->setEnabled(true);
86             ui->btnCalcularVariacoes->setEnabled(true);

```

```

76
77         ui->statusbar->showMessage("Poco criado com Sucesso!")
78         ;
79     } else {
80         // Atualiza dados do poco j existente
81         poco->NomePoco(nome.toStdString());
82         poco->ProfundidadeTotal(profund);
83         poco->PressaoSuperficie(pressao);
84         poco->TemperaturaTopoInicial(temperaturaSuperiorInicial
85             );
86         poco->TemperaturaFundoInicial(temperaturaFundoInicial);
87         poco->TemperaturaTopoFinal(temperaturaSuperiorFinal);
88         poco->TemperaturaFundoFinal(temperaturaFundoFinal);
89         poco->ProfundidadePacker(profundPacker);
90         ui->statusbar->showMessage("Dados de Poco Atualizado
91             com Sucesso!");
92     }
93 }
94
95 void CSimuladorPerdaTubulacao::on_btnAdicionarPropriedades_clicked()
96 {
97     std::string nome;
98     double profundidade, pressaoSup, temperaturaSuperiorInicial,
99         temperaturaFundoInicial, temperaturaSuperiorFinal,
100        temperaturaFundoFinal, ProfundidadePacker;
101
102    QString text;
103
104    text = ui->editNomePoco->text();
105    nome = text.toStdString();
106    text = ui->editPressaoSup->text();
107    pressaoSup = text.toDouble();
108    text = ui->editProfundidadeTotal->text();
109    profundidade = text.toDouble();
110    text = ui->editTemperaturaSuperiorInicial->text();
111    temperaturaSuperiorInicial = text.toDouble();
112    text = ui->editTemperaturaFundoInicial->text();

```

```

111     temperaturaFundoInicial = text.toDouble();
112     text = ui->editTemperaturaSuperiorFinal->text();
113     temperaturaSuperiorFinal = text.toDouble();
114     text = ui->editTemperaturaFundoFinal->text();
115     temperaturaFundoFinal = text.toDouble();
116     text = ui->editProfundidadePacker->text();
117     ProfundidadePacker = text.toDouble();
118
119     if (poco) {
120         QMessageBox::StandardButton resposta = QMessageBox::
121             question(
122                 this,
123                 "",
124                 "Ao confirmar, todos os fluidos ser\u00e3o deletados! Tem
125                 certeza?",
126                 QMessageBox::Yes | QMessageBox::No
127                 );
128
129         if (resposta == QMessageBox::Yes) {
130             poco = std::make_unique<CObjetoPoco>(
131                 CObjetoPoco::CriarParaModulo02(nome, profundidade,
132                     pressaoSup, temperaturaSuperiorInicial,
133                     temperaturaFundoInicial,
134                     temperaturaSuperiorFinal, temperaturaFundoFinal,
135                     ProfundidadePacker)
136             );
137         }
138         on_btnAtualizarDados_clicked();
139     } else {
140
141         poco = std::make_unique<CObjetoPoco>(
142             CObjetoPoco::CriarParaModulo02(nome, profundidade,
143                 pressaoSup, temperaturaSuperiorInicial,
144                 temperaturaFundoInicial, temperaturaSuperiorFinal,
145                 temperaturaFundoFinal, ProfundidadePacker)
146             );
147
148     }
149
150     makePlotTemperatura(temperaturaSuperiorInicial,
151         temperaturaFundoInicial, profundidade, ui->
152         customPlotTemperaturaInicial);
153     makePlotTemperatura(temperaturaSuperiorFinal,

```

```

    temperaturaFundoFinal, profundidade, ui->
    customPlotTemperaturaFinal);
142     makePlotPoco();
143 }
144
145 void CSimuladorPerdaTubulacao::on_btnAtualizarDados_clicked()
146 {
147
148     if (poco){
149         // Atualiza os valores dos QLineEdits com os dados do
150         // objeto poco
151         ui->editNomePoco->setText(QString::fromStdString(poco->
152             NomePoco()));           // Profundidade total do po o
153         ui->editProfundidadeTotal->setText(QString::number(poco->
154             ProfundidadeTotal()));           // Profundidade total do
155             po o
156         ui->editPressaoSup->setText(QString::number(poco->
157             PressaoSuperficie())); // Profundidade ocupada
158         ui->editTemperaturaSuperiorInicial->setText(QString::number(
159             (poco->TemperaturaTopoInicial())));
160             // Di metro do po o
161         ui->editTemperaturaFundoInicial->setText(QString::number(
162             poco->TemperaturaFundoInicial()));
163             // Di metro externo do revestimento (OD)
164         ui->editTemperaturaSuperiorFinal->setText(QString::number(
165             poco->TemperaturaTopoFinal()));
166             // Di metro interno do revestimento (ID)
167         ui->editTemperaturaFundoFinal->setText(QString::number(poco
168             ->TemperaturaFundoFinal()));
169             // Vaz o do fluido no po o
170
171         if (poco->ProfundidadePacker() != 0){
172             ui->editProfundidadePacker->setEnabled(true);
173             ui->editProfundidadePacker->setText(QString::number(
174                 poco->ProfundidadePacker()));
175         }
176
177         // Atualizar QTableWidget com os dados dos trechos
178         ui->tblTreichos->setRowCount(static_cast<int>(poco->Treichos
179             ().size()));
180         ui->tblFluidos->setRowCount(static_cast<int>(poco->Treichos
181             ().size()));

```

```

166     qDebug() << " N mero de trechos:" << poco->Trechos().size()
167     ;
168     int row = 0;
169     for (const auto& trecho : poco->Trechos()) {
170
170         ui->tblTrechos->setItem(row, 0, new QTableWidgetItem(
171             QString::fromStdString(trecho->Nome())));
171         ui->tblTrechos->setItem(row, 1, new QTableWidgetItem(
172             QString::number(trecho->ProfundidadeInicial(), 'f',
173             2)));
172         ui->tblTrechos->setItem(row, 2, new QTableWidgetItem(
173             QString::number(trecho->ProfundidadeFinal(), 'f',
174             2)));
173         ui->tblTrechos->setItem(row, 3, new QTableWidgetItem(
174             QString::number(trecho->DiametroExterno(), 'f',
175             2)));
175         ui->tblTrechos->setItem(row, 4, new QTableWidgetItem(
176             QString::number(trecho->DiametroInterno(), 'f',
177             2)));
176         ui->tblTrechos->setItem(row, 5, new QTableWidgetItem(
177             QString::number(trecho->CoeficientePoisson(), 'f',
178             3)));
178         ui->tblTrechos->setItem(row, 6, new QTableWidgetItem(
179             QString::number(trecho->CoeficienteExpancaoTermica(),
180             'f', 7)));
180         ui->tblTrechos->setItem(row, 7, new QTableWidgetItem(
181             QString::number(trecho->ModuloElasticidade(), 'f',
182             7)));
182         ui->tblTrechos->setItem(row, 8, new QTableWidgetItem(
183             QString::number(trecho->PesoUnidade(), 'f',
184             2)));
184
185         ui->tblFluidos->setItem(row, 0, new QTableWidgetItem(
186             QString::fromStdString(trecho->Fluido()->Nome())));
186         ui->tblFluidos->setItem(row, 1, new QTableWidgetItem(
187             QString::number(trecho->Fluido()->Densidade(), 'f',
188             2)));
188         ui->tblFluidos->setItem(row, 2, new QTableWidgetItem(
189             QString::number(trecho->Fluido()->Viscosidade(), 'f',
190             2)));
190
191         ++row;
192     }

```

```

186     }
187     makePlotTemperatura(poco->TemperaturaTopoInicial(), poco->
188         TemperaturaFundoInicial(), poco->ProfundidadeTotal(), ui->
189         customPlotTemperaturaInicial);
190     makePlotTemperatura(poco->TemperaturaTopoFinal(), poco->
191         TemperaturaFundoFinal(), poco->ProfundidadeTotal(), ui->
192         customPlotTemperaturaFinal);
193     makePlotPoco();
194 }
195
196 void CSimuladorPerdaTubulacao::on_btnAdicionarTrecho_clicked()
197 {
198     ui->tblTrechos->setEditTriggers(QAbstractItemView::NoEditTriggers);
199
200     if (!poco) {
201         QMessageBox::warning(this, "Erro", "As propriedades do ponto precisam estar preenchida!");
202     }
203
204     else{
205         CJanelaAdicionarTrechoTubulacao JanelaTrecho;
206         JanelaTrecho.exec();
207
208         if (JanelaTrecho.Trecho() != "" &&
209             JanelaTrecho.ProfundidadeInicial() != "" &&
210             JanelaTrecho.ProfundidadeFinal() != "" &&
211             JanelaTrecho.DiametroExterno() != "" &&
212             JanelaTrecho.DiametroInterno() != "" &&
213             JanelaTrecho.CoefficientePoisson() != "" &&
214             JanelaTrecho.CoefficienteExpansaoTermica() != "" &&
215             JanelaTrecho.ModuloElasticidade() != "" &&
216             JanelaTrecho.PesoUnidade() != "" &&
217             JanelaTrecho.NomeFluido() != "" &&
218             JanelaTrecho.Densidade() != "" &&
219             JanelaTrecho.Viscosidade() != ""){
220
221         int numLinhas = ui->tblTrechos->rowCount();
222
223         ui->tblTrechos->insertRow(numLinhas);
224         ui->tblTrechos->setItem(numLinhas, 0, new

```

```

    QTableWidgetItem(JanelaTrecho.Trecho()));
222 ui->tblTrechos->setItem(numLinhas, 1, new
    QTableWidgetItem(JanelaTrecho.ProfundidadeInicial()))
);
223 ui->tblTrechos->setItem(numLinhas, 2, new
    QTableWidgetItem(JanelaTrecho.ProfundidadeFinal()));
224 ui->tblTrechos->setItem(numLinhas, 3, new
    QTableWidgetItem(JanelaTrecho.DiametroExterno()));
225 ui->tblTrechos->setItem(numLinhas, 4, new
    QTableWidgetItem(JanelaTrecho.DiametroInterno()));
226 ui->tblTrechos->setItem(numLinhas, 5, new
    QTableWidgetItem(JanelaTrecho.CoefficientePoisson()))
;
227 ui->tblTrechos->setItem(numLinhas, 6, new
    QTableWidgetItem(JanelaTrecho.
CoefficienteExpansaoTermica()));
228 ui->tblTrechos->setItem(numLinhas, 7, new
    QTableWidgetItem(JanelaTrecho.ModuloElasticidade()))
;
229 ui->tblTrechos->setItem(numLinhas, 8, new
    QTableWidgetItem(JanelaTrecho.PesoUnidade()));

230
231 ui->tblFluidos->insertRow(numLinhas);
232 ui->tblFluidos->setItem(numLinhas, 0, new
    QTableWidgetItem(JanelaTrecho.NomeFluido()));
233 ui->tblFluidos->setItem(numLinhas, 1, new
    QTableWidgetItem(JanelaTrecho.Densidade()));
234 ui->tblFluidos->setItem(numLinhas, 2, new
    QTableWidgetItem(JanelaTrecho.Viscosidade()));

235
236 std::string NomeTrecho = JanelaTrecho.Trecho().
    toStdString();
237 double profundInicial = JanelaTrecho.
    ProfundidadeInicial().toDouble();
238 double profundFinal = JanelaTrecho.ProfundidadeFinal().
   toDouble();
239 double diametroExterno = JanelaTrecho.DiametroExterno()
    .toDouble();
240 double diametroInterno = JanelaTrecho.DiametroInterno()
    .toDouble();
241 double coeffientePoisson = JanelaTrecho.
    CoeffientePoisson().toDouble();

```

```

242         double coeficienteExpansaoTermica = JanelaTrecho.
243             CoeficienteExpansaoTermica().toDouble();
244         double moduloElasticidade = JanelaTrecho.
245             ModuloElasticidade().toDouble();
246         double pesoUnidade = JanelaTrecho.PesoUnidade().
247             toDouble();
248
249         std::string nome = JanelaTrecho.NomeFluido().
250             toStdString();
251         double densidade = JanelaTrecho.Densidade().toDouble();
252         double viscosidade = JanelaTrecho.Viscosidade().
253             toDouble();
254
255         auto fluido = std::make_unique<CFluido>(nome, densidade
256             , viscosidade);
257         auto trechoPoco = std::make_unique<CTrechoPoco>(
258             NomeTrecho, profundInicial, profundFinal, std::move(
259                 fluido), diametroExterno, diametroInterno,
260                 coeficientePoisson, coeficienteExpansaoTermica,
261                 moduloElasticidade, pesoUnidade);
262         poco->AdicionarTrechoPoco(std::move(trechoPoco));
263
264         on_btnAtualizarDados_clicked();
265     }
266 }
267
268 void CSimuladorPerdaTubulacao::makePlotTemperatura(double
269 TempInicial, double TempFinal, double profundidade, QCustomPlot*
270 plot)
271 {
272     // Limpar o gráfico anterior
273     plot->clearItems();
274     plot->clearPlottables();
275
276     // Configurar os eixos
277     plot->xAxis->setLabel("Temperatura ( F )");
278     plot->yAxis->setLabel("Profundidade (ft)");
279     plot->yAxis->setRangeReversed(true); // profundidade cresce
280         para baixo
281
282     // Definir os pontos do perfil de temperatura

```

```

271     QVector<double> temperaturas = {TempInicial, (TempInicial+
272                                     TempFinal)/2, TempFinal};      // Temperaturas em F
273     QVector<double> profundidades = {0, (profundidade/2),
274                                     profundidade}; // Profundidades em ft
275
276     // Ajustar ranges com base nos dados
277     double tempMin = *std::min_element(temperaturas.begin(),
278                                         temperaturas.end());
279     double tempMax = *std::max_element(temperaturas.begin(),
280                                         temperaturas.end());
280     double profMin = 0;
281     double profMax = *std::max_element(profundidades.begin(),
282                                         profundidades.end());
283
284     plot->xAxis->setRange(tempMin - 20, tempMax + 20);
285     plot->yAxis->setRange(profMin, profMax);
286
287     // Criar linha do perfil de temperatura (vermelha)
288     QCPGraph *perfilTemp = plot->addGraph();
289     perfilTemp->setData(temperaturas, profundidades);
290     perfilTemp->setPen(QPen(Qt::red, 2));
291     perfilTemp->setName("Condição inicial");
292
293     // Criar linha azul vertical (por exemplo, linha guia em 50 F )
294     double linhaAzulX = 50; // ou alguma variável
295     QCPIItemLine *linhaAzul = new QCPIItemLine(plot);
296     linhaAzul->start->setCoords(linhaAzulX, profMin);
297     linhaAzul->end->setCoords(linhaAzulX, profMax);
298     linhaAzul->setPen(QPen(Qt::blue, 1, Qt::SolidLine));
299
300     // Adicionar texto "condição inicial"
301     QCPIItemText *label = new QCPIItemText(plot);
302     label->position->setCoords(temperaturas[0] + 10, profundidades
303                                 [1]); // proximo ao meio
304     label->setText("condição inicial");
305     label->setFont(QFont("Arial", 10));
306     label->setColor(Qt::red);
307
308     // Marcar os pontos principais
309     for (int i = 0; i < temperaturas.size(); ++i) {
310         QCPIItemEllipse *ponto = new QCPIItemEllipse(plot);
311         ponto->topLeft->setCoords(temperaturas[i] - 5,

```

```

            profundidades[i] - 100);
307     ponto->bottomRight->setCoords(temperaturas[i] + 5,
308                                     profundidades[i] + 100);
309     ponto->setPen(QPen(Qt::black));
310     ponto->setBrush(Qt::gray);
311
312     QCPIItemText *rotulo = new QCPIItemText(plot);
313     rotulo->position->setCoords(temperaturas[i], profundidades[
314         i]);
315     rotulo->setText(QString::number(temperaturas[i]) + " F");
316     rotulo->setFont(QFont("Arial", 9));
317     rotulo->setPositionAlignment(Qt::AlignLeft | Qt::
318                                   AlignVCenter);
319 }
320
321
322 // Atualizar o gráfico
323 plot->replot();
324 }
325
326 void CSimuladorPerdaTubulacao::on_btnRemoverFluido_clicked()
327 {
328     int linhaSelecionada = ui->tblFluidos->currentRow();
329
330     if (linhaSelecionada >= 0) {
331
332         QTableWidgetItem* item = ui->tblFluidos->item(
333             linhaSelecionada, 0);
334
335         if (item) {
336             QString nomeFluido = item->text();
337             ui->tblFluidos->removeRow(linhaSelecionada);
338             poco->RemoverTrechoPoco(nomeFluido.toStdString());
339             on_btnAtualizarDados_clicked();
340         }
341
342     } else {
343         QMessageBox::warning(this, "Erro", "Selecione uma linha para deletar.");
344     }
345
346 void CSimuladorPerdaTubulacao::on_btnRemoverTrecho_clicked()

```

```

343 {
344     int linhaSelecionada = ui->tblTreichos->currentRow();
345
346     if (linhaSelecionada >= 0) {
347
348         QTableWidgetItem* item = ui->tblTreichos->item(
349             linhaSelecionada, 0);
350
351         if (item) {
352             QMessageBox::StandardButton resposta = QMessageBox::
353                 question(
354                     this,
355                     "Tem certeza que deseja remover o trecho?",
356                     QMessageBox::Yes | QMessageBox::No
357                 );
358
359         if (resposta == QMessageBox::Yes) {
360             QString nomeTrecho = item->text();
361             ui->tblTreichos->removeRow(linhaSelecionada);
362             poco->RemoverTrechoPoco(nomeTrecho.toStdString());
363             on_btnAtualizarDados_clicked();
364             ui->statusbar->showMessage("Fluido Removido com "
365                                         "Sucesso!");
366         }
367
368     } else {
369         QMessageBox::warning(this, "Erro", "Selecione uma linha "
370                             "para deletar.");
371     }
372
373 void CSimuladorPerdaTubulacao::makePlotPoco()
374 {
375     ui->customPlotPoco->clearItems();
376     ui->customPlotPoco->xAxis->setLabel("Di metro (pol)");
377     ui->customPlotPoco->yAxis->setLabel("Profundidade (pol)");
378     ui->customPlotPoco->yAxis->setRangeReversed(true);
379
380     if (!poco || poco->Treichos().empty())

```

```

381         return;
382
383     // 1. Profundidade máxima e maior diâmetro externo
384     double profundidadeMaxima = 0.0;
385     double maiorDiametroExterno = 0.0;
386     for (const auto& trecho : poco->Trechos()) {
387         profundidadeMaxima = std::max(profundidadeMaxima, trecho->
388             ProfundidadeFinal());
389         maiorDiametroExterno = std::max(maiorDiametroExterno,
390             trecho->DiametroExterno());
391     }
392
393     // 2. Buraco = maior diâmetro externo + 3 polegadas
394     double diametroBuraco = maiorDiametroExterno + 3.0;
395
396     // 3. Ajuste visual no eixo X: 1.5 vezes o furo
397     double larguraGrafico = diametroBuraco * 1.5;
398     ui->customPlotPoco->xAxis->setRange(-larguraGrafico / 2.0,
399         larguraGrafico / 2.0);
400     ui->customPlotPoco->yAxis->setRange(0, profundidadeMaxima);
401
402     // 4. Cores dos fluidos
403     QMap<QString, QColor> mapaCores;
404     QVector<QColor> coresDisponiveis = {
405         QColor(173, 216, 230, 150), QColor(255, 0, 0, 150),
406         QColor(0, 255, 0, 150), QColor(0, 0, 255, 150),
407         QColor(255, 165, 0, 150), QColor(128, 0, 128, 150)
408     };
409     int corIndex = 0;
410
411     // 5. Desenho dos trechos
412     for (const auto& trecho : poco->Trechos()) {
413         double z1 = trecho->ProfundidadeInicial();
414         double z2 = trecho->ProfundidadeFinal();
415         double dExt = trecho->DiametroExterno();
416         QString nomeFluido = QString::fromStdString(trecho->Fluido
417             ()->Nome());
418
419         if (!mapaCores.contains(nomeFluido)) {
420             mapaCores[nomeFluido] = coresDisponiveis[corIndex++ %
421                 coresDisponiveis.size()];
422         }

```

```

418     QColor corFluido = mapaCores[nomeFluido];
419
420     // === Retângulo cinza (buraco do poço) ===
421     QCPIItemRect *rectBuraco = new QCPIItemRect(ui->
422         customPlotPoco);
423     rectBuraco->topLeft->setCoords(-diametroBuraco / 2.0, z1);
424     rectBuraco->bottomRight->setCoords(diametroBuraco / 2.0, z2
425         );
426     rectBuraco->setPen(QPen(Qt::black));
427     rectBuraco->setBrush(QBrush(QColor(150, 150, 150, 100)));
428
429     // === Retângulo da seção (fluido) ===
430     QCPIItemRect *rectSecao = new QCPIItemRect(ui->customPlotPoco
431         );
432     rectSecao->topLeft->setCoords(-dExt / 2.0, z1);
433     rectSecao->bottomRight->setCoords(dExt / 2.0, z2);
434     rectSecao->setPen(QPen(Qt::black));
435     rectSecao->setBrush(QBrush(corFluido));
436
437     // === Rótulo ===
438     QCPIItemText *label = new QCPIItemText(ui->customPlotPoco);
439     label->position->setCoords(0, (z1 + z2) / 2.0);
440     label->setText(nomeFluido);
441     label->setFont(QFont("Arial", 10, QFont::Bold));
442     label->setColor(Qt::black);
443     label->setPositionAlignment(Qt::AlignCenter);
444 }
445
446 // 6. Desenhar packer se existir
447 double profundidadePacker = poco->ProfundidadePacker();
448 if (profundidadePacker > 0.0) {
449     double alturaPacker = std::max(profundidadeMaxima * 0.01,
450         12.0);
451     double zTop = profundidadePacker - alturaPacker / 2.0;
452     double zBottom = profundidadePacker + alturaPacker / 2.0;
453
454     // Encontrar o trecho correspondente à profundidade do
455     // packer
456     double diametroNoPacker = 0.0;
457     for (const auto& trecho : poco->Trechos()) {
458         if (profundidadePacker >= trecho->ProfundidadeInicial()
459             &&
460             profundidadePacker <= trecho->ProfundidadeFinal())

```

```

        {
454         diametroNoPacker = trecho->DiametroExterno();
455         break;
456     }
457 }
458
459 // Se n o achou trecho correspondente , usa o maior
460 // conhecido como fallback
461 if (diametroNoPacker == 0.0)
462     diametroNoPacker = maiorDiametroExterno;
463
464 // Coordenadas horizontais para os quadrados laterais
465 double xEsq1 = -diametroBuraco / 2.0;
466 double xEsq2 = -diametroNoPacker / 2.0;
467 double xDir1 = diametroNoPacker / 2.0;
468 double xDir2 = diametroBuraco / 2.0;
469
470 // === Quadrado esquerdo ===
471 QCPIItemRect* rectPackerEsq = new QCPIItemRect(ui->
472     customPlotPoco);
473 rectPackerEsq->topLeft->setCoords(xEsq1, zTop);
474 rectPackerEsq->bottomRight->setCoords(xEsq2, zBottom);
475 rectPackerEsq->setPen(QPen(Qt::red, 1.5));
476 rectPackerEsq->setBrush(Qt::NoBrush);
477
478 // === Quadrado direito ===
479 QCPIItemRect* rectPackerDir = new QCPIItemRect(ui->
480     customPlotPoco);
481 rectPackerDir->topLeft->setCoords(xDir1, zTop);
482 rectPackerDir->bottomRight->setCoords(xDir2, zBottom);
483 rectPackerDir->setPen(QPen(Qt::red, 1.5));
484 rectPackerDir->setBrush(Qt::NoBrush);
485
486 // === X vermelho esquerdo ===
487 QCPIItemLine* linha1Esq = new QCPIItemLine(ui->customPlotPoco
488     );
489 linha1Esq->start->setCoords(xEsq1, zTop);
490 linha1Esq->end->setCoords(xEsq2, zBottom);
491 linha1Esq->setPen(QPen(Qt::red, 1.5));
492
493 QCPIItemLine* linha2Esq = new QCPIItemLine(ui->customPlotPoco
494     );

```

```

490     linha2Esq->start->setCoords(xEsq2, zTop);
491     linha2Esq->end->setCoords(xEsq1, zBottom);
492     linha2Esq->setPen(QPen(Qt::red, 1.5));
493
494     // === X vermelho direito ===
495     QCPItemLine* linha1Dir = new QCPItemLine(ui->customPlotPoco
496         );
496     linha1Dir->start->setCoords(xDir1, zTop);
497     linha1Dir->end->setCoords(xDir2, zBottom);
498     linha1Dir->setPen(QPen(Qt::red, 1.5));
499
500     QCPItemLine* linha2Dir = new QCPItemLine(ui->customPlotPoco
501         );
501     linha2Dir->start->setCoords(xDir2, zTop);
502     linha2Dir->end->setCoords(xDir1, zBottom);
503     linha2Dir->setPen(QPen(Qt::red, 1.5));
504 }
505 ui->customPlotPoco->replot();
506 }
507
508 void CSimuladorPerdaTubulacao::on_btnCalcularVariacoes_clicked()
509 {
510     QString profundidadeStr = ui->editProfundidadeMedicao->text();
511     double profundidade = profundidadeStr.toDouble();
512
513     ui->lbnPressaoHidroestatica->setText(QString::number( (poco->
514         PressaoHidroestaticaNoPonto(profundidade) ) ));
514     ui->lbnCargaInicial->setText(QString::number(poco->CargaInicial
515         (profundidade)));
515     ui->lbnTituloDeltaLTemperatura->setText(QString::number(poco->
516         DeltaLTemperaturaTotal())));
516
517 }
518
519 void CSimuladorPerdaTubulacao::on_actionArquivo_Dat_triggered()
520 {
521     QString caminhoDoArquivo = QFileDialog::getOpenFileName(
522         this,
523         "Selecione um arquivo",
524         "",
525         "Todos os arquivos (*.*)"
526     );

```

```
527
528     std::string caminhoDoArquivoStr = caminhoDoArquivo.toStdString
529         ();
530
531     std::ifstream file(caminhoDoArquivoStr);
532
533     if (!file.is_open()) {
534         ui->statusbar->showMessage("Falha ao abrir o arquivo!");
535         return;
536     }
537
538     std::string linha;
539     bool lendoTreichos = false;
540
541     while (std::getline(file, linha)) {
542         if (linha.find("Configura o dos Trechos") != std::string
543             ::npos) {
544             lendoTreichos = true;
545             continue;
546         }
547
548         if (!lendoTreichos) {
549             // Leitura dos dados do po o
550             std::istringstream iss(linha);
551             std::string nome;
552             double profundidade, pressaoSup;
553             double temperaturaSuperiorInicial,
554                 temperaturaFundoInicial;
555             double temperaturaSuperiorFinal, temperaturaFundoFinal;
556             double profundidadePacker;
557
558             if (iss >> nome >> profundidade >> pressaoSup
559                 >> temperaturaSuperiorInicial >>
560                     temperaturaFundoInicial
561                 >> temperaturaSuperiorFinal >>
562                     temperaturaFundoFinal >> profundidadePacker) {
563
564                 ui->btnAdicionarTrecho->setEnabled(true);
565                 ui->btnRemoverTrecho->setEnabled(true);
566             }
567         }
568     }
569 }
```

```

564         ui->btnCalcularVariacoes->setEnabled(true);
565
566         poco = std::make_unique<CObjetoPoco>(
567             CObjetoPoco::CriarParaModulo02(nome,
568                 profundidade, pressaoSup,
569                             temperaturaSuperiorInicial
570                             ,
571                             temperaturaFundoInicial
572                             ,
573                             temperaturaSuperiorFinal
574                             ,
575                             temperaturaFundoFinal
576                             ,
577                             profundidadePacker)
578         );
579     } else {
580         std::cerr << "Erro ao ler linha de po o:" <<
581             linha << std::endl;
582     }
583 } else {
584     // Leitura dos dados dos trechos
585     std::istringstream iss(linha);
586     std::string nomeTrecho, nomeFluido;
587     double profundInicial, profundFinal;
588     double diametroExterno, diametroInterno;
589     double coeficientePoisson, coeficienteExpansaoTermica;
590     double moduloElasticidade, pesoUnidade;
591     double densidade, viscosidade;
592
593     if (iss >> nomeTrecho >> profundInicial >> profundFinal
594         >> diametroExterno >> diametroInterno
595         >> coeficientePoisson >> coeficienteExpansaoTermica
596         >> moduloElasticidade >> pesoUnidade
597         >> nomeFluido >> densidade >> viscosidade) {
598
599         auto fluido = std::make_unique<CFluido>(nomeFluido,
600             densidade, viscosidade);
601         auto trechoPoco = std::make_unique<CTrechoPoco>(
602             nomeTrecho,
603             profundInicial, profundFinal, std::move(fluido)
604             ,
605             diametroExterno, diametroInterno,
606             )
607     }
608 }
```

```

594             coeficientePoisson, moduloElasticidade,
595             pesoUnidade, coeficienteExpansaoTermica
596         );
597
598         if (!poco->AdicionarTrechoPoco(std::move(trechoPoco
599             ))) {
600             std::cerr << "Falha ao adicionar trecho ao
601             p o o .\n";
602         } else {
603             std::cerr << "Erro ao ler trecho: " << linha << std
604             ::endl;
605         }
606     }
607
608     file.close();
609     on_btnAtualizarDados_clicked();
610     ui->statusbar->showMessage("Dados importados com sucesso!");
611
612 }
613
614
615 void CSimuladorPerdaTubulacao::on_actionNova_Simula_o_triggered()
616 {
617     QMessageBox::StandardButton resposta = QMessageBox::question(
618         this,
619         "",
620         "Tem certeza que deseja iniciar uma nova simula o?",
621         QMessageBox::Yes | QMessageBox::No
622     );
623
624     if (resposta == QMessageBox::Yes) {
625         CSimuladorPerdaTubulacao *newWindow = new
626             CSimuladorPerdaTubulacao();
627         newWindow->show();
628         this->close();
629     }
630
631

```

```

632 void CSimuladorPerdaTubulacao::
633     on_actionExportar_Como_Imagem_triggered()
634 {
635     QString fileName = QFileDialog::getSaveFileName(this, "Salvar"
636         " imagem", "", "PNG (*.png);;JPEG (*.jpg)");
637
638     QPixmap pixmap = this->grab();
639     pixmap.save(fileName);
640 }
641
642
643 void CSimuladorPerdaTubulacao::on_actionSobre_o_SEEP_triggered()
644 {
645     QDesktopServices::openUrl(QUrl("https://github.com/ldsc/
646         ProjetoEngenharia-
647         SoftwareEducacionalParaAnaliseESolucaoDeProblemasEmEngenhariaDePoco
648         "));
649 }

```

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 temperaturaTopoInicial = 0.0;
23     double temperaturaFundoInicial = 0.0;
24     double temperaturaTopoFinal = 0.0;
25     double temperaturaFundoFinal = 0.0;
26     double profundidadePacker = 0.0;
27
28 public:
29     // Construtor e destrutor padrão
30     CObjetoPoco() = default;
31     ~CObjetoPoco() = default;
32
33     // Evita cípia (pra evitar duplicação desnecessária dos
34     // trechos)
35     CObjetoPoco(const CObjetoPoco&) = delete;
36     CObjetoPoco& operator=(const CObjetoPoco&) = delete;
37
38     // Permite movimentação (move semantics)
39     CObjetoPoco(CObjetoPoco&&) = default;
40     CObjetoPoco& operator=(CObjetoPoco&&) = default;
41
42     // M todos de criação estéticos, separando claramente os
43     // dados exigidos por cada módulo
44     static CObjetoPoco CriarParaModulo01(std::string Nome, double
45         Profund, double PressaoSup, double D, double OD, double ID,
46         double q);
47     static CObjetoPoco CriarParaModulo02(std::string Nome, double
48         Profund, double PressaoSup, double TempTopoInicial, double
49         TempFundoInicial, double TempTopoFinal, double
50         TempFundoFinal, double profundidadePacker);
51
52     // Getters para acessar os atributos de forma segura
53     std::string NomePoco() const { return nomePoco; }
54     double ProfundidadeTotal() const { return profundidadeFinal; }
55     double ProfundidadeOcupada() const { return profundidadeOcupada;
56     }
57     double PressaoSuperficie() const { return pressaoSuperficie; }
58     double DiametroPoco() const { return diametroPoco; }

```

```
51     double DiametroRevestimentoOD() const { return
52         diametroRevestimentoOD; }
53     double DiametroRevestimentoID() const { return
54         diametroRevestimentoID; }
55     double Vazao() const { return vazao; }
56     double TemperaturaTopoInicial() const { return
57         temperaturaTopoInicial; }
58     double TemperaturaFundoInicial() const { return
59         temperaturaFundoInicial; }
60     double TemperaturaTopoFinal() const { return
61         temperaturaTopoFinal; }
62     double TemperaturaFundoFinal() const { return
63         temperaturaFundoFinal; }
64     double ProfundidadePacker() const { return profundidadePacker;
65     }
66
67     // Retorna os trechos adicionados ao po o (em forma de
68     // ponteiros brutos)
69     std::vector<CTrechoPoco*> Trechos() const;
70
71     // Setters permitem modificar os atributos do po o
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 DiametroPoco(double D) { diametroPoco = D; }
80     void DiametroRevestimentoOD(double DiametroExterno) {
81         diametroRevestimentoOD = DiametroExterno; }
82     void DiametroRevestimentoID(double DiametroInterno) {
83         diametroRevestimentoID = DiametroInterno; }
84     void Vazao(double q) { vazao = q; }
85     void TemperaturaTopoInicial(double temperatura) {
86         temperaturaTopoInicial = temperatura; }
87     void TemperaturaFundoInicial(double temperatura) {
88         temperaturaFundoInicial = temperatura; }
89     void TemperaturaTopoFinal(double temperatura) {
90         temperaturaTopoFinal = temperatura; }
91     void TemperaturaFundoFinal(double temperatura) {
92         temperaturaFundoFinal = temperatura; }
```

```

76     void ProfundidadePacker(double profundidade) {
77         profundidadePacker = profundidade; }
78
79     // M todos de c lculo
80     double PressaoHidroestaticaTotal() const;
81     double PressaoHidroestaticaNoPonto(double profundidade) const;
82     double DensidadeEfetivaTotal() const;
83     double ViscosidadeEfetivaTotal() const;
84     bool VerificarPreenchimentoColuna();
85     double CargaInicial(double profundidade) const;
86     double DeltaLTemperaturaTotal() const;
87     double DeltaLEfeitoBalao(double profundidade) const;
88     double VariacaoCargaEfeitoPistao(double profundidade) const;
89     double DeltaLPistaoPacker(double profundidade) const;
90     double DeltaLPistaoCrossover(double profundidade) const;
91     double DeltaLForcaRestauradora(double profundidade) const;
92     double CargaInjecao(double profundidade, bool colunaFixa) const
93         ;
94
95     double TemperaturaNoPonto(double profundidade) const;
96
97     bool AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco>
98         trechoParaAdicionar); // Fun o para adicionar um novo
99         trecho ao p o
100    void RemoverTrechoPoco(const std::string& nomeFluido); // Remove
101        trecho do p o com base no nome do fluido
102
103    // M todos que retornam dados para gr ficos
104    std::pair<std::vector<double>, std::vector<double>>
105        PlotarProfundidadePorPressao();
106    std::pair<std::vector<double>, std::vector<double>>
107        PlotarProfundidadePorPressaoMedia();
108};

109#endif

```

Fonte: produzido pelo autor.

Apresenta-se na listagem ?? a implementa o da classe CObjetoPoco.

Listing 7.7: Arquivo de implementa o da classe CObjetoPoco

```

1 #include "CObjetoPoco.h"
2 #include <iostream>
3 #include <vector>

```

```

4 #include <iostream>
5 #include <cstdlib>
6 #include <numbers>
7 #include <math.h>
8
9
10 CObjetoPoco CObjetoPoco::CriarParaModulo01(std::string nomeDoPoco ,
11     double profundidadeFinalDoPoco , double pressaoNaSuperficie ,
12     double diametroDoPoco ,
13     double
14     diametroRevestimentoExterno
15     , double
16     diametroRevestimentoInterno
17     , double vazaoDoPoco )
18 {
19
20     CObjetoPoco objetoPoco ;
21
22     objetoPoco . nomePoco = nomeDoPoco ;
23     objetoPoco . profundidadeFinal = profundidadeFinalDoPoco ;
24     objetoPoco . pressaoSuperficie = pressaoNaSuperficie ;
25     objetoPoco . diametroPoco = diametroDoPoco ;
26     objetoPoco . diametroRevestimento0D = diametroRevestimentoExterno
27     ;
28     objetoPoco . diametroRevestimentoID = diametroRevestimentoInterno
29     ;
30     objetoPoco . vazao = vazaoDoPoco ;
31
32     return objetoPoco ;
33 }
34
35
36 CObjetoPoco CObjetoPoco::CriarParaModulo02(std::string nomeDoPoco ,
37     double profundidadeFinalDoPoco , double pressaoNaSuperficie ,
38     double
39     temperaturaTopoInicial
40     , double
41     temperaturaFundoInicial
42     ,
43     double
44     temperaturaTopoFinal ,
45     double
46     temperaturaFundoFinal

```

```

        , double
        profundidadeDoPacker)
{
29    CObjetoPoco objetoPoco;
30
31    objetoPoco.nomePoco = nomeDoPoco;
32    objetoPoco.profundidadeFinal = profundidadeFinalDoPoco;
33    objetoPoco.pressaoSuperficie = pressaoNaSuperficie;
34    objetoPoco.temperaturaTopoInicial = temperaturaTopoInicial;
35    objetoPoco.temperaturaFundoInicial = temperaturaFundoInicial;
36    objetoPoco.temperaturaTopoFinal = temperaturaTopoFinal;
37    objetoPoco.temperaturaFundoFinal = temperaturaFundoFinal;
38    objetoPoco.profundidadePacker = profundidadeDoPacker;
39
40    return objetoPoco;
41}
42
43
44 std::vector<CTrechoPoco*> CObjetoPoco::Treichos() const {
45     std::vector<CTrechoPoco*> vetorDePonteirosParaTreichos;
46
47     // percorre todos os trechos armazenados no po o
48     for (const auto& trechoUnico : trechos) {
49         // adiciona o ponteiro cru (n o nico ) de cada trecho ao
50         // vetor de sa da
51         vetorDePonteirosParaTreichos.push_back(trechoUnico.get());
52     }
53
54     // retorna o vetor contendo os ponteiros dos trechos
55     return vetorDePonteirosParaTreichos;
56 }
57 bool CObjetoPoco::AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco>
58                                         trechoParaAdicionar) {
59     // calcula o comprimento do trecho com base nas profundidades
60     // inicial e final
61     double comprimentoDoTrecho = trechoParaAdicionar->
62         ProfundidadeFinal() - trechoParaAdicionar->
63         ProfundidadeInicial();
64
65     // move o trecho para dentro do vetor principal do po o
66     trechos.push_back(std::move(trechoParaAdicionar));

```

```

63
64     // atualiza a profundidade total ocupada no po o somando o
       novo trecho
65     profundidadeOcupada += comprimentoDoTrecho;
66
67     return true; // opera o realizada com sucesso
68 }
69
70
71 void CObjetoPoco::RemoverTrechoPoco(const std::string& nomeFluido)
{
72     for (auto it = trechos.begin(); it != trechos.end();) {
73         if ((*it)->Fluido()->Nome() == nomeFluido) {
74             double comprimento = (*it)->ProfundidadeFinal() - (*it)
                           ->ProfundidadeInicial();
75             it = trechos.erase(it);
76             profundidadeOcupada -= comprimento;
77         } else {
78             ++it;
79         }
80     }
81 }
82
83 double CObjetoPoco::PressaoHidroestaticaTotal() const {
84     double pressaoTotalHidrostatica = 0.0;
85
86     // soma a pressao hidrostatica de todos os trechos do po o
87     for (const auto& trechoAtual : trechos) {
88         pressaoTotalHidrostatica += trechoAtual->
                           PressaoHidroestatica();
89     }
90
91     // adiciona a pressao da superficie para obter a pressao total
92     return pressaoTotalHidrostatica + pressaoSuperficie;
93 }
94
95 double CObjetoPoco::PressaoHidroestaticaNoPonto(double
profundidadeDesejada) const {
96     double pressaoAcumulada = pressaoSuperficie;
97     double profundidadeJaCalculada = 0.0;
98
99     // percorre cada trecho verificando se ele contem a

```

```

    profundidade desejada
100   for (const auto& trechoAtual : trechos) {
101     double comprimentoDoTrecho = trechoAtual->ProfundidadeFinal
102       () - trechoAtual->ProfundidadeInicial();
103
104     // se a profundidade estiver dentro do trecho atual
105     if (profundidadeDesejada <= profundidadeJaCalculada +
106       comprimentoDoTrecho) {
107       double profundidadeDentroDoTrecho =
108         profundidadeDesejada - profundidadeJaCalculada;
109
110       // adiciona somente a parte proporcional da pressao no
111       // trecho
112       pressaoAcumulada += trechoAtual->PressaoHidroestatica(
113         profundidadeDentroDoTrecho);
114       break;
115     } else {
116       // adiciona a pressao total do trecho completo
117       pressaoAcumulada += trechoAtual->PressaoHidroestatica()
118       ;
119       profundidadeJaCalculada += comprimentoDoTrecho;
120     }
121   }
122
123   return pressaoAcumulada;
124 }

125 bool CObjetoPoco::VerificarPreenchimentoColuna() {
126   double profundidadeNaoPreenchida = profundidadeFinal -
127     profundidadeOcupada;
128
129   if (profundidadeNaoPreenchida > 0) {
130     std::cout << "Uma coluna de " << profundidadeNaoPreenchida
131       << " litros de fluido precisa ser adicionada!" << std::endl;
132     return false; // coluna incompleta
133   } else {
134     std::cout << "A coluna de fluidos equivale a profundidade "
135       "total do po o !" << std::endl;
136     return true; // coluna completamente preenchida
137   }
138 }

139 }
```

```

132
133 double CObjetoPoco::DensidadeEfetivaTotal() const {
134     double somaDensidadePonderada = 0.0;
135     double comprimentoTotalDaColuna = 0.0;
136
137     for (const auto& trechoAtual : trechos) {
138         double comprimentoTrecho = trechoAtual->ProfundidadeFinal()
139             - trechoAtual->ProfundidadeInicial();
140
141         // multiplica a densidade equivalente pelo comprimento do
142         // trecho para ponderar
143         somaDensidadePonderada += trechoAtual->DensidadeEquivalente
144             () * comprimentoTrecho;
145         comprimentoTotalDaColuna += comprimentoTrecho;
146     }
147
148     return somaDensidadePonderada / comprimentoTotalDaColuna;
149 }
150
151
152     double CObjetoPoco::ViscosidadeEfetivaTotal() const {
153         double somaDasViscosidades = 0.0;
154
155         // percorre todos os trechos para somar as viscosidades dos
156         // fluidos
157         for (const auto& trechoAtual : trechos) {
158             somaDasViscosidades += trechoAtual->Fluido()->Viscosidade()
159                 ;
160         }
161
162         // retorna a media simples das viscosidades
163         return somaDasViscosidades / trechos.size();
164     }
165
166     std::pair<std::vector<double>, std::vector<double>> CObjetoPoco::
167     PlotarProfundidadePorPressaoMedia() {
168         std::vector<double> vetorDeProfundidades;
169         std::vector<double> vetorDePressoes;
170
171         // percorre cada metro ao longo da profundidade total do poço
172         for (double profundidadeAtual = 0.0; profundidadeAtual <=
173             ProfundidadeTotal(); profundidadeAtual += 1.0) {
174             vetorDeProfundidades.push_back(profundidadeAtual);
175         }
176     }

```

```

167
168     // calcula a pressao no ponto atual usando a funcao
169     // apropriada
170     double pressaoNoPonto = PressaoHidroestaticaNoPonto(
171         profundidadeAtual);
172     vetorDePressoes.push_back(pressaoNoPonto);
173 }
174
175 }
176
177 double CObjetoPoco::CargaInicial(double profundidadeAlvo) const {
178     const double pi = 3.141592653589793;
179     double cargaTotal = 0.0;
180
181     // === 1. Pressao hidrostatica interna no ponto analisado ===
182     for (const auto& trechoAtual : trechos) {
183         if (profundidadeAlvo >= trechoAtual->ProfundidadeInicial()
184             &&
185             profundidadeAlvo <= trechoAtual->ProfundidadeFinal()) {
186
187             double diametroInterno = trechoAtual->DiametroInterno()
188                 ; // polegadas
189             double areaInterna = pi / 4.0 * (diametroInterno *
190                 diametroInterno); // in
191
192             double densidadeFluido = trechoAtual->Fluido()->
193                 Densidade(); // lb/gal
194             double pressaoNoPonto = 0.052 * densidadeFluido *
195                 profundidadeAlvo; // psi
196
197             double forcaInterna = pressaoNoPonto * areaInterna; //
198             // lbf
199             cargaTotal += forcaInterna;
200
201             break; // se considera o primeiro trecho que cobre a
202                 // profundidade
203         }
204     }
205 }
```

```

199 // === 2. Efeito pistão (crossovers acima do ponto) ===
200 for (size_t i = 1; i < trechos.size(); ++i) {
201     const auto& trechoAcima = trechos[i - 1];
202     const auto& trechoAbaixo = trechos[i];
203
204     double profundidadeCrossover = trechoAbaixo->
205         ProfundidadeInicial();
206
207     if (profundidadeCrossover >= profundidadeAlvo)
208         continue;
209
210     double densidadeInterna = trechoAcima->Fluido()->Densidade
211         ();
212     double densidadeExterna = trechoAbaixo->Fluido()->Densidade
213         ();
214
215     double pressaoInterna = 0.052 * densidadeInterna *
216         profundidadeCrossover;
217     double pressaoExterna = 0.052 * densidadeExterna *
218         profundidadeCrossover;
219
220     double areaInterna = pi / 4.0 * pow(trechoAcima->
221         DiametroInterno(), 2);
222     double areaExterna = pi / 4.0 * pow(trechoAbaixo->
223         DiametroExterno(), 2);
224
225     double forcaPistao = pressaoInterna * areaInterna +
226         pressaoExterna * areaExterna;
227     cargaTotal += forcaPistao;
228 }
229
230 // === 3. Peso da coluna acima da profundidade ===
231 for (const auto& trechoAtual : trechos) {
232     double profundidadeInicialTrecho = trechoAtual->
233         ProfundidadeInicial();
234     double profundidadeFinalTrecho = trechoAtual->
235         ProfundidadeFinal();
236
237     if (profundidadeFinalTrecho <= profundidadeAlvo)
238         continue;
239
240     double profundidadeUtil = std::max(profundidadeAlvo,

```

```

        profundidadeInicialTrecho);
231     double comprimentoConsiderado = profundidadeFinalTrecho -
232         profundidadeUtil;
233
234     if (comprimentoConsiderado > 0.0) {
235         double pesoPorMetro = trechoAtual->PesoUnidade(); // lb
236         / ft
237         double pesoTotal = pesoPorMetro *
238             comprimentoConsiderado;
239         cargaTotal += pesoTotal;
240     }
241 }
242
243 double CObjetoPoco::DeltaTemperaturaTotal() const {
244     double variacaoTotalComprimento = 0.0;
245     double temperaturaReferencia = TemperaturaTopoInicial(); // /
246         temperatura do topo como base
247
248     for (const auto& trechoAtual : trechos) {
249         double profundidadeInicial = trechoAtual->
250             ProfundidadeInicial();
251         double profundidadeFinal = trechoAtual->ProfundidadeFinal()
252             ;
253         double comprimentoTrecho = profundidadeFinal -
254             profundidadeInicial;
255
256         if (comprimentoTrecho <= 0.0)
257             continue; // ignora trechos nulos ou invertidos
258
259         double coefExpansaoTermica = trechoAtual->
260             CoeficienteExpancaoTermica();
261
262         // temperatura m dia do trecho considerando varia o
263         linear
264         double temperaturaInicial = TemperaturaNoPonto(
265             profundidadeInicial);
266         double temperaturaFinal = TemperaturaNoPonto(
267             profundidadeFinal);
268         double temperaturaMedia = (temperaturaInicial +

```

```

        temperaturaFinal) / 2.0;

261
262     double variacaoTemperatura = temperaturaReferencia -
263         temperaturaMedia;
264
265     // aplica a f rmula de dilata o linear: L = * L *
266     // T
267     double deltaComprimento = coefExpansaoTermica *
268         comprimentoTrecho * variacaoTemperatura;
269     variacaoTotalComprimento += deltaComprimento;
270 }
271
272
273 double CObjetoPoco::TemperaturaNoPonto(double profundidadeDesejada)
274 {
275     double temperaturaNoTopo = TemperaturaTopoInicial();
276     double temperaturaNoFundo = TemperaturaFundoInicial();
277     double profundidadeTotalDoPoco = ProfundidadeTotal();
278
279     // evita divis o por zero no caso de profundidade inv lida
280     if (profundidadeTotalDoPoco <= 0.0)
281         return temperaturaNoTopo;
282
283     // aplica interpola o linear para calcular a temperatura no
284     // ponto
285     return temperaturaNoTopo + (temperaturaNoFundo -
286         temperaturaNoTopo) * (profundidadeDesejada /
287         profundidadeTotalDoPoco);
288 }

```

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
41 #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 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 etodos para determinar fatores e velocidades
48     double DeterminarFatorFriccao(double re);
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 etodos 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 poco 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
16 // Mesmo calculo, mas permitindo passar a viscosidade como
17 // parametro
18 double CModeloReologico::DeterminarReynoldsPoco(double viscosidade)
19 {
20    reynoldsPoco = (928 * poco->DensidadeEfetivaTotal() *
21                    vMediaPoco * poco->DiametroRevestimentoID()) /
22                    viscosidade;
23    return reynoldsPoco;
24 }
25
26 // Calcula o numero de Reynolds no espaco anular
27 double CModeloReologico::DeterminarReynoldsAnular() {
28    reynoldsAnular = (757 * poco->DensidadeEfetivaTotal() *
29                      vMediaAnular *
30                      (poco->DiametroPoco() - poco->
31                         DiametroRevestimentoOD())) /
32                      poco->ViscosidadeEfetivaTotal();
33    return reynoldsAnular;
34 }
35
36 // Mesmo calculo para o anular, com viscosidade recebida
37 // externamente
38 double CModeloReologico::DeterminarReynoldsAnular(double
39           viscosidade) {
40    reynoldsAnular = (757 * poco->DensidadeEfetivaTotal() *
41                      vMediaAnular *
42                      (poco->DiametroPoco() - poco->
43                         DiametroRevestimentoOD())) /
44                      viscosidade;
45    return reynoldsAnular;

```

```

35 }
36
37 // Calcula a velocidade m dia do fluido no interior da coluna (
38 // o poco)
38 double CModeloReologico::DeterminarVelocidadeMediaPoco() {
39     vMediaPoco = poco->Vazao() / (2.448 * std::pow(poco->
40         DiametroRevestimentoID(), 2));
41     return vMediaPoco;
41 }
42
43 // Calcula a velocidade m dia no espaco anular entre a coluna e o
43 // revestimento
44 double CModeloReologico::DeterminarVelocidadeMediaAnular() {
45     vMediaAnular = poco->Vazao() /
46         (2.448 * (std::pow(poco->DiametroPoco(), 2) -
47             std::pow(poco->DiametroRevestimentoOD(), 2)));
47 ;
48     return vMediaAnular;
48 }
49
50 // Calcula o fator de friccao usando a equacao implicita de
50 // Fanning com Newton-Raphson
51 double CModeloReologico::DeterminarFatorFriccao(double re) {
52     // Estimativa inicial baseada em escoamento laminar (nao e
52     // usada diretamente, mas serve como chute)
53     auto laminar_fator = [] (double re) {
54         return 0.0791 / std::pow(re, 0.25);
55     };
56
57     // Equacao de Fanning: 1/sqrt(f) - 4log10(Re*sqrt(f)) + 0.395 =
57     // 0
58     auto f = [re](double x) {
59         return 1 / std::sqrt(x) - 4 * std::log10(re * std::sqrt(x))
60             + 0.395;
61     };
62
62     // Derivada da equacao de Fanning
63     auto df = [] (double x) {
64         return -0.5 / (x * std::sqrt(x)) - (2 / (x * std::log(10)))
64         ;
65     };
66

```

```

67     // Metodo de Newton-Raphson para resolver a equacao
68     auto newtonRaphson = [&](double x0, double tol = 1e-6, int
69         max_iter = 1000000) {
70         double x = x0;
71         for (int i = 0; i < max_iter; i++) {
72             double fx = f(x);
73             double dfx = df(x);
74             if (std::abs(fx) < tol) {
75                 return x;
76             }
77             x -= fx / dfx;
78         }
79         return x;
80     };
81
82     // Chute inicial com base no fator para fluxo laminar
83     double x0 = laminar_fator(re);
84
85     // Retorna o valor resolvido
86     return newtonRaphson(x0);
87 }
```

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 */
13
14 class CModeloNewtoniano : public CModeloReologico {
15     public:
16         // 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" : "

```

```

        Turbulento";
20     return fluxoAnular;
21 }
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     fatorFriccaoPoco = DeterminarFatorFriccao(reynoldsPoco);
30
31     if (fluxoPoco == "Laminar") {
32         return (poco->ViscosidadeEfetivaTotal() * vMediaPoco) /
33                 (1500 * std::pow(poco->DiametroRevestimentoID(), 2))
34                 ;
35 } else {
36     return (fatorFriccaoPoco * poco->DensidadeEfetivaTotal() *
37             std::pow(vMediaPoco, 2)) /
38             (25.8 * poco->DiametroRevestimentoID());
39 }
40
41 // Calcula a perda de carga por friccao no espaco anular
41 double CModeloNewtoniano::CalcularPerdaPorFriccaoAnular() {
42     if (fluxoAnular.empty()) {
43         DeterminarFluxoAnular();
44     }
45
46     double diametroAnular = poco->DiametroPoco() - poco->
47             DiametroRevestimentoOD();
48     fatorFriccaoAnular = DeterminarFatorFriccao(reynoldsAnular);
49
50     if (fluxoAnular == "Laminar") {
51         return (poco->ViscosidadeEfetivaTotal() * vMediaAnular) /
52                 (1000 * std::pow(diametroAnular, 2));
53 } else {
54     return (fatorFriccaoAnular * poco->DensidadeEfetivaTotal()
55             * std::pow(vMediaAnular, 2)) /
56             (21.1 * diametroAnular);
57 }
```

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 parâmetros do modelo
36     CModeloBingham(CObjetoPoco* poco, double viscosidadePlastica,
```

```

        double pontoDeEscoamento)
37      : CModeloReologico(poco),
38      viscosidadePlastica(viscosidadePlastica),
39      pontoDeEscoamento(pontoDeEscoamento)
40    {
41      DeterminarFluxoPoco();
42      DeterminarFluxoAnular();
43    }
44
45 // Getters
46 double ViscosidadePlastica() const { return viscosidadePlastica;
47   ; }
48 double PontoDeEscoamento() const { return pontoDeEscoamento; }
49 double ReynoldsCriticoPoco() const { return reynoldsCriticoPoco;
50   ; }
51 double ReynoldsCriticoAnular() const { return
52   reynoldsCriticoAnular; }
53 double ReynoldsHedstronPoco() const { return
54   reynoldsHedstronPoco; }
55 double ReynoldsHedstronAnular() const { return
56   reynoldsHedstronAnular; }

57 // Setters
58 void ViscosidadePlastica(double valor) { viscosidadePlastica =
59   valor; }
60 void PontoDeEscoamento(double valor) { pontoDeEscoamento =
61   valor; }

62 // Metodos especificos do modelo de Bingham
63 double DeterminarReynoldsCritico(double hedstron);
64 double DeterminarReynoldsHedstronPoco();
65 double DeterminarReynoldsHedstronAnular();
66 std::string DeterminarFluxoPoco() override;
67 std::string DeterminarFluxoAnular() override;
68 double CalcularPerdaPorFriccaoPoco() override;
69 double CalcularPerdaPorFriccaoAnular() override;
70 };
71
72 #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
9 // Hedstron
10 // Utiliza metodo numerico de Newton-Raphson para resolver a
11 // equacao implicita
12 double CModeloBingham::DeterminarReynoldsCritico(double hedstron) {
13     // f(x) = ((x / (1 - x)^3) * 16800) - Hedstron
14     auto func = [hedstron](double x) {
15         return ((x / std::pow(1 - x, 3)) * 16800) - hedstron;
16     };
17
18     // Derivada de f(x)
19     auto derivadaFunc = [] (double x) {
20         return (16800 * (1 + x) / std::pow(1 - x, 4));
21     };
22
23     // Metodo de Newton-Raphson para aproximar a raiz
24     auto newtonRaphson = [&] (double x) {
25         for (int i = 0; i < 10000; ++i) {
26             x = x - func(x) / derivadaFunc(x);
27             if (fabs(func(x)) < 1e-2) break;
28         }
29         return x;
30     };
31
32     // Chute inicial
33     double y = newtonRaphson(0.99);
34
35     // Retorna o Reynolds critico com base em y encontrado
36     return ((1 - ((4.0 / 3.0) * y) + ((1.0 / 3.0) * std::pow(y, 4)))
37             * hedstron) / (8 * y);
38 }
39
40 // Calcula o numero de Hedstron para o escoamento no poco
41 double CModeloBingham::DeterminarReynoldsHedstronPoco() {
```

```

39     reynoldsHedstronPoco =
40         (37100 * poco->DensidadeEfetivaTotal() * pontoDeEscoamento
41             *
42             std::pow(poco->DiametroRevestimentoID(), 2)) /
43             std::pow(viscosidadePlastica, 2);
44
45     return reynoldsHedstronPoco;
46 }
47
48 // Calcula o numero de Hedstron para o escoamento no espaco anular
49 double CModeloBingham::DeterminarReynoldsHedstronAnular() {
50     double diametroAnular = poco->DiametroPoco() - poco->
51             DiametroRevestimentoOD();
52
53     reynoldsHedstronAnular =
54         (24700 * poco->DensidadeEfetivaTotal() * pontoDeEscoamento
55             *
56             std::pow(diametroAnular, 2)) /
57             std::pow(viscosidadePlastica, 2);
58
59     return reynoldsHedstronAnular;
60 }
61
62
63 // Determina o tipo de fluxo no poco (laminar ou turbulento)
64 std::string CModeloBingham::DeterminarFluxoPoco() {
65     DeterminarVelocidadeMediaPoco();
66     DeterminarReynoldsPoco(viscosidadePlastica);
67     DeterminarReynoldsHedstronPoco();
68
69     reynoldsCriticoPoco = DeterminarReynoldsCritico(
70         reynoldsHedstronPoco);
71
72     fluxoPoco = (reynoldsPoco <= reynoldsCriticoPoco) ? "Laminar" :
73         "Turbulento";
74
75     return fluxoPoco;
76 }
77
78
79 // Determina o tipo de fluxo no espaco anular
80 std::string CModeloBingham::DeterminarFluxoAnular() {
81     DeterminarVelocidadeMediaAnular();
82     DeterminarReynoldsAnular(viscosidadePlastica);
83     DeterminarReynoldsHedstronAnular();
84
85 }
```

```

76
77     reynoldsCriticoAnular = DeterminarReynoldsCritico(
78         reynoldsHedstromAnular);
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 (fatorFriccaoPoco * poco->DensidadeEfetivaTotal() *
96                 std::pow(vMediaPoco, 2)) /
97                 (25.8 * poco->DiametroRevestimentoID());
98     }
99 }
100
101 // Calcula a perda de carga por friccao no espaco anular
102 double CModeloBingham::CalcularPerdaPorFriccaoAnular() {
103     double diametroAnular = poco->DiametroPoco() - poco->
104         DiametroRevestimentoOD();
105     fatorFriccaoAnular = DeterminarFatorFriccao(reynoldsAnular);
106
107     if (fluxoAnular == "Laminar") {
108         return ((viscosidadePlastica * vMediaAnular) /
109                 (1000 * std::pow(diametroAnular, 2))) +
110                     (pontoDeEscoamento / (200 * diametroAnular));
111     } else {
112         return (fatorFriccaoAnular * poco->DensidadeEfetivaTotal()
113                 *
114                     std::pow(vMediaAnular, 2)) /
115                     (21.1 * diametroAnular);
116     }
117 }

```

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 Potência
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 indiceDeConsistência = 0.0;           // K
19     double indiceDeComportamento = 1.0;          // n
20
21     // Reynolds crítico arbitrário (2100 como base de separação)
22     double reynoldsCriticoPoco = 2100.0;
23     double reynoldsCriticoAnular = 2100.0;
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 indiceDeConsistência)
33         : CModeloReologico(poco),
34             indiceDeConsistência(indiceDeConsistência)
35     {
36         DeterminarFluxoPoco();
37         DeterminarFluxoAnular();
38     }
39
40     // Getters
41
42     double getIndiceDeConsistência() const { return indiceDeConsistência; }
43     void setIndiceDeConsistência(double novoIndice) { indiceDeConsistência = novoIndice; }
44
45     double getIndiceDeComportamento() const { return indiceDeComportamento; }
46     void setIndiceDeComportamento(double novoIndice) { indiceDeComportamento = novoIndice; }
47
48     double getReynoldsCriticoPoco() const { return reynoldsCriticoPoco; }
49     void setReynoldsCriticoPoco(double novoValor) { reynoldsCriticoPoco = novoValor; }
50
51     double getReynoldsCriticoAnular() const { return reynoldsCriticoAnular; }
52     void setReynoldsCriticoAnular(double novoValor) { reynoldsCriticoAnular = novoValor; }
53 }
```

```

38     double ReynoldsCriticoPoco() const { return reynoldsCriticoPoco
39         ; }
40     double ReynoldsCriticoAnular() const { return
41         reynoldsCriticoAnular; }
42     double IndiceDeConsistencia() const { return
43         indiceDeConsistencia; }
44     double IndiceDeComportamento() const { return
45         indiceDeComportamento; }
46     std::string FluxoPoco() const { return fluxoPoco; }
47     std::string FluxoAnular() const { return fluxoAnular; }
48
49 // Setters
50
51     void IndiceDeConsistencia(double valor) { indiceDeConsistencia
52         = valor; }
53     void IndiceDeComportamento(double valor) {
54         indiceDeComportamento = valor; }
55     void FluxoPoco(const std::string& fluxo) { fluxoPoco = fluxo; }
56     void FluxoAnular(const std::string& fluxo) { fluxoAnular =
57         fluxo; }
58
59 // Metodos especificos do modelo de potencia
60
61     double DeterminarReynoldsCritico(double reynolds);
62     double DeterminarReynoldsPoco();
63     double DeterminarReynoldsAnular();
64     std::string DeterminarFluxoPoco() override;
65     std::string DeterminarFluxoAnular() override;
66     double CalcularPerdaPorFriccaoPoco() override;
67     double CalcularPerdaPorFriccaoAnular() override;
68 };
69
70 #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
4 // Funcao generica para Reynolds critico (nao esta sendo usada
5 // diretamente aqui)
5 double CModeloPotencia::DeterminarReynoldsCritico(double Reynolds)
6 {

```

```

6     (void) Reynolds; // evita warning por parametro nao usado
7     return 0.0;
8 }
9
10 // Calcula o numero de Reynolds para o escoamento no poco
11 double CModeloPotencia::DeterminarReynoldsPoco() {
12     reynoldsPoco =
13         ((89100 * poco->DensidadeEfetivaTotal() * std::pow(
14             vMediaPoco, 2 - indiceDeComportamento)) /
15             indiceDeConsistencia) *
16             (std::pow((0.0416 * poco->DiametroRevestimentoID()) / (3 +
17                 (1 / indiceDeComportamento)), indiceDeComportamento));
18
19     // Para fluidos com n muito baixo (pseudoplastico forte), pode
20     // ajustar o critico
21     if (indiceDeComportamento < 0.4) {
22         reynoldsCriticoPoco = DeterminarReynoldsCritico(
23             reynoldsPoco);
24     }
25
26     return reynoldsPoco;
27 }
28
29 // Calcula o numero de Reynolds no espaco anular
30 double CModeloPotencia::DeterminarReynoldsAnular() {
31     reynoldsAnular =
32         ((109000 * poco->DensidadeEfetivaTotal() * std::pow(
33             vMediaAnular, 2 - indiceDeComportamento)) /
34             indiceDeConsistencia) *
35             (std::pow((0.0208 * (poco->DiametroPoco() - poco->
36                 DiametroRevestimentoID())) / (2 + (1 /
37                 indiceDeComportamento)), indiceDeComportamento));
38
39     if (indiceDeComportamento < 0.4) {
40         reynoldsCriticoAnular = DeterminarReynoldsCritico(
41             reynoldsAnular);
42     }
43
44     return reynoldsAnular;
45 }
46
47 // Determina o tipo de fluxo no poco com base no valor de Reynolds

```

```

    calculado

38 std::string CModeloPotencia::DeterminarFluxoPoco() {
39     vMediaPoco = DeterminarVelocidadeMediaPoco();
40     reynoldsPoco = DeterminarReynoldsPoco();
41     fluxoPoco = (reynoldsPoco <= reynoldsCriticoPoco) ? "Laminar" :
42         "Turbulento";
43     return fluxoPoco;
44 }

45 // Determina o tipo de fluxo no espaço anular
46 std::string CModeloPotencia::DeterminarFluxoAnular() {
47     vMediaAnular = DeterminarVelocidadeMediaAnular();
48     reynoldsAnular = DeterminarReynoldsAnular();
49     fluxoAnular = (reynoldsAnular <= reynoldsCriticoAnular) ? "
50         Laminar" : "Turbulento";
51     return fluxoAnular;
52 }

53 // Calcula a perda de carga por fricção no pôco, separando para
54 // fluxo laminar e turbulento
55 double CModeloPotencia::CalcularPerdaPorFriccaoPoco() {
56     fatorFriccaoPoco = DeterminarFatorFriccao(reynoldsPoco);
57
58     if (fluxoPoco == "Laminar") {
59         return ((indiceDeConsistencia * std::pow(vMediaPoco, -
60             indiceDeComportamento)) *
61                 std::pow(( (3 + (1 / indiceDeComportamento)) /
62                     0.0416), indiceDeComportamento)) /
63                 (144000 * std::pow(poco->DiametroRevestimentoID(), 1
64                     + indiceDeComportamento));
65     } else {
66         return (fatorFriccaoPoco * poco->DensidadeEfetivaTotal() *
67                 std::pow(vMediaPoco, 2)) /
68                 (25.8 * poco->DiametroRevestimentoID());
69     }
70 }

71 // Calcula a perda de carga por fricção no espaço anular
72 double CModeloPotencia::CalcularPerdaPorFriccaoAnular() {
73     double diametroAnular = poco->DiametroPoco() - poco->
74         DiametroRevestimentoOD();
75     fatorFriccaoAnular = DeterminarFatorFriccao(reynoldsAnular);

```

```

71
72     if (fluxoAnular == "Laminar") {
73         return ((indiceDeConsistencia * std::pow(vMediaAnular, -
74             indiceDeComportamento)) *
75                 std::pow(((2 + (1 / indiceDeComportamento)) /
76                     0.0208), indiceDeComportamento)) /
77                 (144000 * std::pow(diametroAnular, 1 +
78                     indiceDeComportamento));
79     } else {
80         return (fatorFriccaoAnular * poco->DensidadeEfetivaTotal()
81             * std::pow(vMediaAnular, 2)) /
82             (21.1 * diametroAnular);
83     }
84 }
```

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 class CJanelaAdicionarFluido : public QDialog
```

```

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     bool modoEdicao = true;
48     Ui::CJanelaAdicionarFluido *ui = nullptr;
49
50     QString nomeFluido;
51     QString densidade;
52     QString viscosidade;
53     QString profundidadeInicial;
54     QString profundidadeFinal;
55 };
56
57 #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 // Destrutor - 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         } else {
40             // Modo sem edicao da faixa de profundidade

```

```

41     NomeFluido(ui->LnValorNome->text());
42     Densidade(ui->LnValorDensidade->text());
43     Viscosidade(ui->LnValorViscosidade->text());
44
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 namespace Ui {
16 class CJanelaAdicionarTrechoTubulacao;
```

```

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) {

```

```

        coeficienteExpansaoTermica = valor; }

46 void ModuloElasticidade(const QString& valor) {
    moduloElasticidade = valor; }

47 void PesoUnidade(const QString& valor) { pesoUnidade = valor; }
48 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     // dados do fluido no trecho
72     QString nomeFluido;
73     QString densidade;
74     QString viscosidade;
75
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(
7     QWidget *parent)
8     : QDialog(parent)
9     , ui(new Ui::CJanelaAdicionarTrechoTubulacao)
10 {
11     ui->setupUi(this);
12 }
13 // Destrutor: 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, \u00e1preencha\u00e1 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 hidroestatica x profundidade
9 Recebe os dados (profundidade e pressao) e plota o grafico usando
10 QCustomPlot
11 Permite exportar o grafico em imagem
11 */
12
13 namespace Ui {
14 class CJanelaGraficoPressaoHidroestatica;
15 }
16
```

```

17 class CJanelaGraficoPressaoHidroestatica : public QDialog
18 {
19     Q_OBJECT
20
21 public:
22     explicit CJanelaGraficoPressaoHidroestatica(
23         const std::pair<std::vector<double>, std::vector<double>>&
24             dados,
25             QWidget *parent = nullptr);
26
27     // metodo para atualizar os dados do grafico
28     void PerfilHidrostatico(const std::pair<std::vector<double>,
29                             std::vector<double>>& novoPerfil);
30
31 private slots:
32     // acao ao clicar no botao de exportar imagem
33     void on_BtnExportarGrafico_clicked();
34
35 private:
36     Ui::CJanelaGraficoPressaoHidroestatica *ui;
37
38     // vetores com dados de profundidade e pressao
39     std::vector<double> profundidades;
40     std::vector<double> pressoes;
41
42     // funcao que monta o grafico
43     void PlotarGraficoPressaoxProfundidade();
44 };
45 #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::
10 CJanelaGraficoPressaoHidroestatica(
11     const std::pair<std::vector<double>, std::vector<double>>&
12     dados,
13     QWidget *parent)
14     : QDialog(parent)
15     , ui(new Ui::CJanelaGraficoPressaoHidroestatica)
16     , profundidades(dados.first)
17     , pressoes(dados.second)
18 {
19     ui->setupUi(this);
20     PlotarGraficoPressaoxProfundidade();
21 }
22
23 // Destrutor: libera memoria
24 CJanelaGraficoPressaoHidroestatica::~
25 CJanelaGraficoPressaoHidroestatica()
26 {
27     delete ui;
28 }
29
30 // Permite atualizar os dados apos a criacao da janela
31 void CJanelaGraficoPressaoHidroestatica::PerfilHidrostatico(
32     const std::pair<std::vector<double>, std::vector<double>>&
33     novoPerfil)
34 {
35     profundidades = novoPerfil.first;
36     pressoes = novoPerfil.second;
37     PlotarGraficoPressaoxProfundidade();
38 }
39
40 // Funcao que plota o grafico com base na relacao profundidade x
41 // pressao
42 // Parte da logica para segmentacao por inclinacao foi adaptada com
43 // auxilio de IA (ChatGPT)
44 void CJanelaGraficoPressaoHidroestatica::
45     PlotarGraficoPressaoxProfundidade()
46 {
47     QVector<double> x(profundidades.begin(), profundidades.end());
48     QVector<double> y(pressoes.begin(), pressoes.end());

```

```

41
42     auto *plot = ui->customPlotPressaoMediaProfundidade;
43     plot->clearGraphs();
44
45     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 -
61             1]);
62
63         // Se mudou muito a inclinacao, cria um novo trecho no
64         // grafico
65         if (!isMesmaInclinacao(inclinacaoAtual, inclinacaoAnterior,
66             tolerancia)) {
67             int index = plot->graphCount();
68             plot->addGraph();
69             plot->graph(index)->setData(trechoX, trechoY);
70
71             // Uso de cor e estilo dinamico inspirado em sugestao
72             // de IA para melhorar visualizacao
73             QPen pen;
74             pen.setWidth(1);
75             pen.setColor(QColor::fromHsv((index * 70) % 360, 255,
76                 200));
77             plot->graph(index)->setPen(pen);
78
79             QCPScatterStyle estilo(QCPScatterStyle::ssCircle, 4);
80             plot->graph(index)->setScatterStyle(estilo);
81             plot->graph(index)->setLineStyle(QCPGraph::lsLine);
82
83             plot->graph(index)->rescaleAxes();
84         }
85     }
86
87     plot->replot();
88 }
```

```

77         plot->graph(index)->setName(QString("Fluido %1").arg(
78             index + 1));
79
80         trechoX.clear();
81         trechoY.clear();
82     }
83
84     trechoX.push_back(x[i]);
85     trechoY.push_back(y[i]);
86     inclinacaoAnterior = inclinacaoAtual;
87 }
88
89 // adiciona ultimo trecho
90 int index = plot->graphCount();
91 plot->addGraph();
92 plot->graph(index)->setData(trechoX, trechoY);
93
94 QPen pen;
95 pen.setWidth(1);
96 pen.setColor(QColor::fromHsv((index * 70) % 360, 255, 200));
97 plot->graph(index)->setPen(pen);
98
99 QCPSatterStyle estiolo(QCPSatterStyle::ssCircle, 4);
100 plot->graph(index)->setScatterStyle(estiolo);
101 plot->graph(index)->setLineStyle(QCPGraph::lsLine);
102 plot->graph(index)->setName(QString("Fluido %1").arg(index + 1)
103 );
104
105 // Configura rotulos dos eixos
106 plot->xAxis->setLabel("Profundidade(ft)");
107 plot->yAxis->setLabel("Pressão Hidrostática(psi)");
108
109 // Define o intervalo dos eixos com base nos dados
110 plot->xAxis->setRange(*std::min_element(x.begin(), x.end()),
111                         *std::max_element(x.begin(), x.end()));
112 plot->yAxis->setRange(*std::min_element(y.begin(), y.end()),
113                         *std::max_element(y.begin(), y.end()));
114
115 // Legenda posicionada no canto inferior direito
116 plot->legend->setVisible(true);
117 plot->axisRect()->insetLayout()->setInsetAlignment(0, Qt::
118             AlignRight | Qt::AlignBottom);

```

```

116     plot->legend->setBrush(QBrush(Qt::white));
117     plot->legend->setBorderPen(QPen(Qt::gray));
118     plot->legend->setFont(QFont("Arial", 9));
119
120     // Ativacao do zoom, arrastar e selecao com mouse (configuracao
121     // 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)
126     connect(plot, &QCustomPlot::mouseMove, this, [=](QMouseEvent *
127             event) {
128         double xVal = plot->xAxis->pixelToCoord(event->pos().x());
129         double yVal = plot->yAxis->pixelToCoord(event->pos().y());
130
131         plot->setToolTip(QString("Profundidade: %1 ft\nPress o: %2
132                         \u03cpsi")
133                         .arg(xVal, 0, 'f', 2)
134                         .arg(yVal, 0, 'f', 2));
135     });
136 }
137
138 // Botao de exportar imagem do grafico (logica de exportacao por
139 // QPixmap)
139 void CJanelaGraficoPressaoHidroestatica::
140     on_BtnExportarGrafico_clicked()
141 {
142     QString fileName = QFileDialog::getSaveFileName(this, "Salvar
143                         grafico", "", "PNG (*.png);;JPEG (*.jpg)");
144
145     if (!fileName.isEmpty()) {
146         QPixmap imagem = ui->customPlotPressaoMediaProfundidade->
147             toPixmap(800, 600);
148         imagem.save(fileName);
149     }
150 }
```

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
9 software:
10 - Modulo 1: simulacoes de pressao e escoamento
11 - Modulo 2: analise de deslocamentos axiais da tubulacao
12 */
13
14 namespace Ui {
15     class JanelaMenu;
16 }
17
18 class JanelaMenu : public QMainWindow
19 {
20     Q_OBJECT
21
22     public:
23         explicit JanelaMenu(QWidget *parent = nullptr); // construtor
24             da janela principal
25         ~JanelaMenu(); // destrutor
26
27     private slots:
28         void on_btnModulo01_clicked(); // acao ao clicar no Modulo 1
29         void on_btnModulo02_clicked(); // acao ao clicar no Modulo 2
30
31     private:
32         Ui::JanelaMenu *ui;
33 };
34
35#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 Tensoes na Coluna </b><br>"
41         "<ul>"
42         "<li>Carga Axial.</li>"
43         "<li>Colapsos e Explosao da Coluna.</li>"
44         "</ul>";

```

```

43
44     ui ->lbnModulo02 ->setText(buttonText_02);
45     ui ->btnModulo02 ->setText("");
46     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
56 // reologica
56 void JanelaMenu::on_btnModulo01_clicked()
57 {
58     CSimuladorReologico *w = new CSimuladorReologico(this);
59     w->setWindowTitle("SEAPEP - Software Educacional de Engenharia"
60                         " de Po o");
60     w->show();
61 }
62
63 // Quando o usuario clica no Modulo 2, abre a janela de simulacao
63 // de perda na tubulacao
64 void JanelaMenu::on_btnModulo02_clicked()
65 {
66     CSimuladorPerdaTubulacao *w = new CSimuladorPerdaTubulacao(this
67 );
68     w->setWindowTitle("SEAPEP - Software Educacional de Engenharia"
69                         " de Po o");
69     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, verificando a precisão dos resultados apresentados pelo código.

Aplicando os conceitos apresentados no capítulo de Metodologia, os modelos de perda de fricção foram desenvolvidos para aplicação no poço e anular, além disso consideram tanto o fluxo laminar quanto o fluxo turbulento. Inicialmente, será apresentado o modelo para o regime laminar, detalhando suas características e cálculos específicos. Em seguida, abordaremos o regime turbulento, destacando as diferenças e particularidades de cada caso.

8.1 Propriedades da simulação

Para iniciarmos os testes, antes precisamos definir as propriedades do poço e dos fluidos que foram utilizadas na simulação. Dois cenários de propriedades foram utilizados, um para produzir um regime de fluxo laminar e outro para produzir um regime de fluxo turbulento.

8.1.1 Configurações para o regime turbulento

Abaixo na Tabela 8.1 temos as propriedades do poço.

Tabela 8.1: Configuração do poço turbulento

Profundidade (ft)	Pressão na superfície (psi)	Diâmetro (in)	OD (in)	ID (in)	Vazão (gal/min)
11483	2175	8.66	7.87	7.09	158

Abaixo na Tabela 8.2 temos as propriedades dos fluidos.

Tabela 8.2: Configuração do poço turbulento

Fluidos	Densidade (lbm/gal)	Viscosidade (cP)	Profundidade inicial (ft)	Profundidade final (ft)
1	8.76	1.8	0	3281
2	8.17	2.1	3281	8202
3	9.17	1.3	8202	11483

8.1.2 Configurações para o regime laminar

Abaixo na Tabela 8.3 temos as propriedades do poço.

Tabela 8.3: Configuração do poço laminar

Profundidade (ft)	Pressão na superfície (psi)	Diâmetro (in)	OD (in)	ID (in)	Vazão (gal/min)
3281	2175	8.66	7.87	7.09	100

Abaixo na Tabela 8.4 temos as propriedades dos fluidos.

Tabela 8.4: Configuração do poço turbulento

Fluidos	Densidade (lbm/gal)	Viscosidade (cP)	Profundidade inicial (ft)	Profundidade final (ft)
1	8.76	30	0	3281

8.2 Testes

8.2.1 Validação da perda de fricção pelo modelo Newtoniano no poço

Aqui, abordaremos o modelo de perda de fricção para um fluido com comportamento Newtoniano, aplicado diretamente no poço. Posteriormente, o regime turbulento será analisado, considerando a adaptação das equações para representar o comportamento do fluido no poço em condições de escoamento mais instáveis.

Para o fluxo laminar no poço realizamos os seguintes cálculos:

$$\bar{v} = \frac{100}{2.448 \times 7.09^2} = 0.8126$$

$$N_{re} = \frac{928 \times 8.76 \times 0.8126 \times 7.09}{30} = 1561.18$$

$$\frac{dp_f}{dL} = \frac{30 \times 0.8126}{1500 \times 7.09^2} = 3.23 \times 10^{-4}$$

Na Figura 8.1 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

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

```
#####
#               Menu de Perda de Carga - Newtoniano          #
#####

1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 1

Velocidade no Poco: 0.812636 ft/s
Reynolds no Poco: 1561.25
Tipo de Fluxo no Poco: Laminar
Perda Friccional no Poco: 0.000323321 psi/ft

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

Para o fluxo turbulento no poço realizamos os seguintes cálculos:

$$\bar{v} = \frac{158}{2.448 \times 7.09^2} = 1.2839$$

$$N_{re} = \frac{928 \times 8.62 \times 1.2839 \times 7.09}{1.73} = 42090.74$$

$$\frac{dp_f}{dL} = \frac{0.0055 \times 8.62 \times 1.2839^2}{25.8 \times 7.09} = 4.21 \times 10^{-4}$$

Na Figura 8.2 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 8.2: Soluções geradas pelo código para modelo Newtoniano no poço considerando regime turbulento

```
#####
#               Menu de Perda de Carga - Newtoniano      #
#####

1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 1

Velocidade no Poco: 1.28397 ft/s
Reynolds no Poco: 42032.9
Tipo de Fluxo no Poco: Turbulento
Perda Friccional no Poco: 0.000422143 psi/ft

Fator de Friccao no Poco: 0.00543119

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

8.2.2 Validação da perda de fricção pelo modelo Newtoniano no anular

Neste tópico, será realizada a validação do modelo de perda de fricção para um fluido Newtoniano escoando na região anular do poço. Primeiramente, será analisado o comportamento em regime laminar, considerando as equações e parâmetros que governam esse tipo de escoamento. Em seguida, será apresentado o modelo para o regime turbulento, que envolve cálculos adicionais devido à maior complexidade no comportamento do fluxo.

Para o fluxo laminar no anular realizamos os seguintes cálculos:

$$\bar{v} = \frac{100}{2.448 \times (8.66^2 - 7.87^2)} = 3.1281$$

$$N_{re} = \frac{757 \times 8.76 \times 3.1281 \times (8.66 - 7.87)}{30} = 546.24$$

$$\frac{dp_f}{dL} = \frac{30 \times 3.1281}{1000 \times (8.66 - 7.87)^2} = 0.1503$$

Na Figura 8.3 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

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

```
#####
#               Menu de Perda de Carga - Newtoniano
#####
1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 2

Velocidade no anular: 3.12816 ft/s
Reynolds no anular: 546.254
Tipo de Fluxo no anular: Laminar
Perda Friccional no Anular: 0.150368 psi/ft

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

Para o fluxo turbulento no anular realizamos os seguintes cálculos:

$$\bar{v} = \frac{158}{2.448 \times (8.66^2 - 7.87^2)} = 4.94$$

$$N_{re} = \frac{757 \times 8.62 \times 4.94 \times (8.66 - 7.87)}{1.73} = 14720$$

$$\frac{dp_f}{dL} = \frac{0.007 \times 8.62 \times 4.94^2}{21.1 \times (8.66 - 7.87)} = 0.09$$

Na Figura 8.4 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 8.4: Soluções geradas pelo código para modelo Newtoniano no anular considerando regime turbulento

```
#####
#           Menu de Perda de Carga - Newtoniano          #
#####

1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 2

Velocidade no anular: 4.94249 ft/s
Reynolds no anular: 14706.5
Tipo de Fluxo no anular: Turbulento
Perda Friccional no Anular: 0.0883044 psi/ft

Fator de Friccao no Anular: 0.00698677

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

8.2.3 Validação da perda de fricção pelo modelo Bingham no poço

Aqui, abordaremos o modelo de perda de fricção para um fluido com comportamento Bingham, aplicado diretamente no poço. O estudo inicia com o regime laminar, onde os parâmetros como limite de escoamento e viscosidade plástica são avaliados para determinar a perda de carga. Posteriormente, o regime turbulento será analisado, considerando a adaptação das equações para representar o comportamento do fluido no poço em condições de escoamento mais instáveis.

Para o fluxo laminar no poço realizamos os seguintes cálculos:

$$\bar{v} = \frac{100}{2.448 \times 7.09^2} = 0.8126$$

$$N_{He} = \frac{37100 \times 8.76 \times 40 \times 7.09^2}{8.7^2} = 8.63 \times 10^6$$

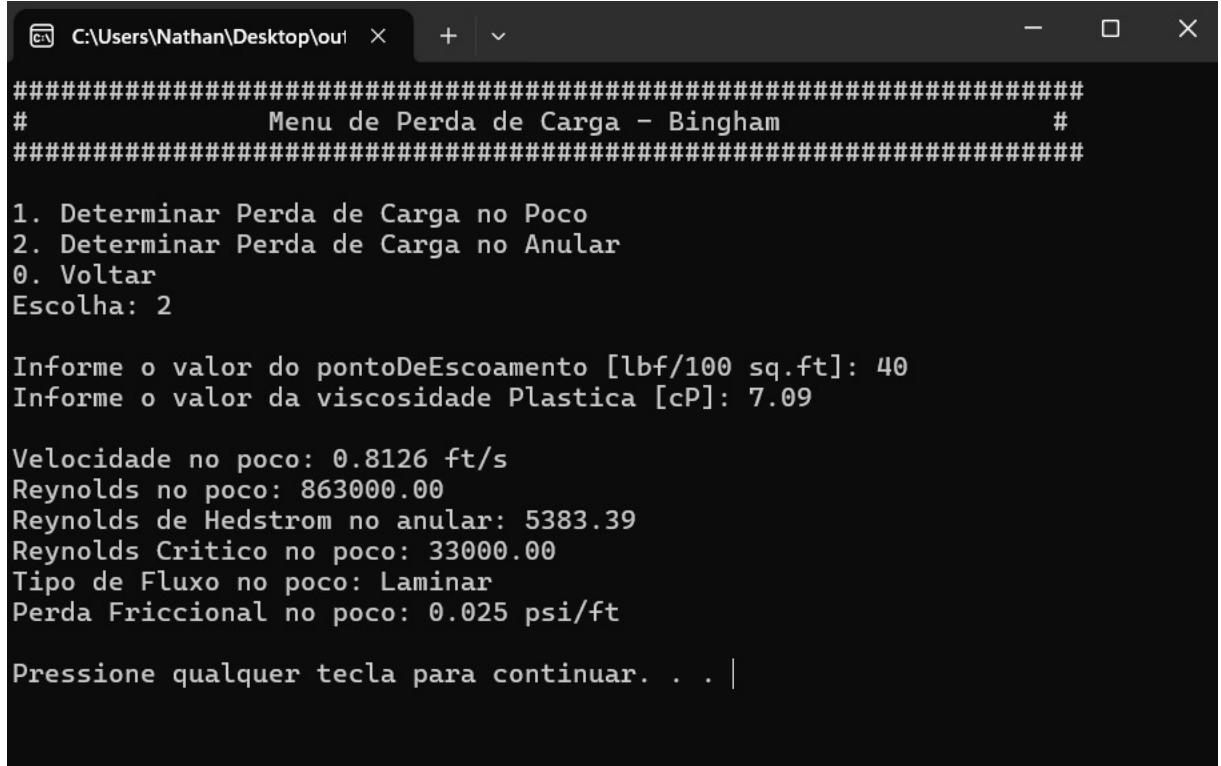
$$N_{rec} = 33000$$

$$N_{re} = \frac{928 \times 8.76 \times 0.8126 \times 7.09}{8.7} = 5383.39$$

$$\frac{dp_f}{dL} = \frac{8.7 \times 0.8126}{1500 \times 7.09^2} + \frac{40}{225 \times 7.09} = 0.025$$

Na Figura 8.5 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 8.5: Soluções geradas pelo código para modelo plástico de Bingham no poço considerando regime laminar



```

C:\Users\Nathan\Desktop\out X
+ | - X

#####
#      Menu de Perda de Carga - Bingham      #
#####

1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 2

Informe o valor do pontoDeEscoamento [lbf/100 sq.ft]: 40
Informe o valor da viscosidade Plastica [cP]: 7.09

Velocidade no poco: 0.8126 ft/s
Reynolds no poco: 863000.00
Reynolds de Hedstrom no anular: 5383.39
Reynolds Critico no poco: 33000.00
Tipo de Fluxo no poco: Laminar
Perda Friccional no poco: 0.025 psi/ft

Pressione qualquer tecla para continuar. . .

```

Fonte: Produzido pelo autor.

Para o fluxo turbulento no poço realizamos os seguintes cálculos:

$$\bar{v} = \frac{158}{2.448 \times 7.09^2} = 1.2839$$

$$N_{He} = \frac{37100 \times 8.62 \times 1 \times 7.09^2}{1.8^2} = 4.96 \times 10^6$$

$$N_{rec} = 27000$$

$$N_{re} = \frac{928 \times 8.62 \times 1.2839 \times 7.09}{1.8} = 40453.88$$

$$\frac{dp_f}{dL} = \frac{0.0057 \times 8.62 \times 1.2839^2}{25.8 \times 7.09} = 4.45 \times 10^{-4}$$

Na Figura 8.6 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 8.6: Soluções geradas pelo código para modelo plástico de Bingham no poço considerando regime turbulento

```
#####
#               Menu de Perda de Carga - Bingham      #
#####

1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 1

Informe o valor do pontoDeEscoamento [lbf/100 sq.ft]: 1
Informe o valor da viscosidade Plastica [cP]: 1.8

Velocidade no Poco: 1.28397 ft/s
Reynolds no Poco: 40476.1
Reynolds Hedstrom no Poco: 4.96416e+006
Reynolds Critico no Poco: 26746.7
Tipo de Fluxo no Poco: Turbulento
Perda Friccional no Poco: 0.000425782 psi/ft

Fator de Friccao: 0.005478

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

8.2.4 Validação da perda de fricção pelo modelo Bingham no anular

Nesta seção, validaremos a perda de fricção para o modelo Bingham na região anular. A análise começará pelo regime laminar, onde a influência do limite de escoamento será considerada na determinação das perdas. Em seguida, abordaremos o caso turbulento, adaptando o modelo para capturar as características dinâmicas do fluxo de um fluido Bingham sob essas condições.

Para o fluxo laminar no anular realizamos os seguintes cálculos:

$$\bar{v} = \frac{100}{2.448 \times (8.66^2 - 7.87^2)} = 3.1281$$

$$N_{He} = \frac{24700 \times 8.76 \times 40 \times (8.66 - 7.87)^2}{8.7^2} = 71363.59$$

$$N_{rec} = 6000$$

$$N_{re} = \frac{757 \times 8.76 \times 3.1281 \times (8.66 - 7.87)}{8.7} = 1883.59$$

$$\frac{dp_f}{dL} = \frac{8.7 \times 3.1281}{1000 \times (8.66 - 7.87)^2} + \frac{40}{200 \times (8.66 - 7.87)} = 0.29$$

Na Figura 8.7 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 8.7: Soluções geradas pelo código para modelo plástico de Bingham no anular considerando regime laminar

```

C:\Users\Nathan\Desktop\out x + v
#####
#      Menu de Perda de Carga - Bingham      #
#####

1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 2

Informe o valor do pontoDeEscoamento [lbf/100 sq.ft]: 40
Informe o valor da viscosidade Plastica [cP]: 7.09

Velocidade no poco: 0.8126 ft/s
Reynolds no poco: 863000.00
Reynolds de Hedstrom no anular: 5383.39
Reynolds Critico no poco: 33000.00
Tipo de Fluxo no poco: Laminar
Perda Friccional no poco: 0.025 psi/ft

Pressione qualquer tecla para continuar. . .

```

Fonte: Produzido pelo autor.

Para o fluxo turbulento no anular realizamos os seguintes cálculos:

$$\bar{v} = \frac{158}{2.448 \times (8.66^2 - 7.87^2)} = 4.94$$

$$N_{He} = \frac{24700 \times 8.62 \times 1 \times (8.66 - 7.87)^2}{1.8^2} = 41012.23$$

$$N_{rec} = 5000$$

$$N_{re} = \frac{757 \times 8.62 \times 4.94 \times (8.66 - 7.87)}{1.8} = 14147.66$$

$$\frac{dp_f}{dL} = \frac{0.007 \times 8.62 \times 4.94^2}{21.1 \times (8.66 - 7.87)} = 0.088$$

Na Figura 8.8 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 8.8: Soluções geradas pelo código para modelo plástico de Bingham no anular considerando regime turbulento

```
#####
#           Menu de Perda de Carga - Bingham      #
#####
1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 2

Informe o valor do pontoDeEscoamento [lbf/100 sq.ft]: 1
Informe o valor da viscosidade Plastica [cP]: 1.8

Velocidade no anular: 4.94249 ft/s
Reynolds no anular: 14161.9
Reynolds de Hedstrom no anular: 41032.7
Reynolds Critico no anular: 5049.83
Tipo de Fluxo no anular: Turbulento
Perda Friccional no Anular: 0.0891554 psi/ft

Fator de Friccao: 0.0070541

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

8.2.5 Validação da perda de fricção pelo modelo de potência no poço

Este tópico aborda a validação do modelo de perda de fricção para fluidos que seguem a lei de potência dentro do poço. Primeiramente, será feita a análise para o regime laminar, considerando o comportamento característico dos fluidos pseudoplásticos ou dilatantes. Em seguida, será apresentado o caso turbulento, com as adaptações necessárias para refletir o comportamento do modelo de potência sob condições de maior velocidade e instabilidade no fluxo.

Para o fluxo laminar no poço realizamos os seguintes cálculos:

$$\bar{v} = \frac{100}{2.448 \times 7.09^2} = 0.8126$$

$$N_{re} = \frac{89100 \times 8.76 \times 0.8126^1}{200} \left(\frac{0.0416 \times 7.09}{3 + \frac{1}{1}} \right)^1 = 233.83$$

$$\frac{dp_f}{dL} = \frac{200 \times 0.8126^1 \left(\frac{3+\frac{1}{1}}{0.0416} \right)^1}{144000 \times 7.09^{1+1}} = 2.1 \times 10^{-3}$$

Na Figura 8.9 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 8.9: Soluções geradas pelo código para modelo lei de potência no poço considerando regime laminar

```
C:\Users\Nathan\Desktop\out x + v
#####
#           Menu de Perda de Carga - Potencia #
#####
1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 1

Informe o valor do indice de consistencia [Cp eq]: 1

Velocidade no poco: 0.8126 ft/s
Reynolds no poco: 233.83
Reynolds Critico no poco: 2100
Tipo de Fluxo no poco: Laminar
Perda Friccional no poco: 0.29 psi/ft

Pressione qualquer tecla para continuar. . . |
```

Fonte: Produzido pelo autor.

Para o fluxo turbulento no poço realizamos os seguintes cálculos:

$$\bar{v} = \frac{158}{2.448 \times 7.09^2} = 1.2839$$

$$N_{re} = \frac{89100 \times 8.62 \times 0.8126^1}{25} \left(\frac{0.0416 \times 7.09}{3 + \frac{1}{1}} \right)^1 = 2908$$

$$\frac{dp_f}{dL} = \frac{0.012 \times 8.62 \times 1.2839^2}{25.8 \times 7.09} = 9.3 \times 10^{-4}$$

Na Figura 8.10 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 8.10: Soluções geradas pelo código para modelo lei de potência no poço considerando regime turbulento

```
#####
#           Menu de Perda de Carga - Potencia          #
#####

1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 1

Informe o valor do indice de consistencia [Cp eq]: 25

Velocidade no Poco: 1.28397 ft/s
Reynolds no Poco: 2910.01
Reynolds Critico no Poco: 2100
Tipo de Fluxo no Poco: Turbulento
Perda Friccional no Poco: 0.000853658 psi/ft

Fator de Friccao: 0.010983

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

8.2.6 Validação da perda de fricção pelo modelo de potência no anular

Nesta seção, será validado o modelo de perda de fricção para fluidos da lei de potência na região anular. A validação começa com o regime laminar, onde as propriedades de escoamento dos fluidos pseudoplásticos ou dilatantes são analisadas. Posteriormente, será tratado o caso turbulento, com um enfoque nas alterações nos parâmetros para capturar adequadamente as perdas de fricção em um ambiente de escoamento mais dinâmico.

Para o fluxo laminar no poço realizamos os seguintes cálculos:

$$\bar{v} = \frac{100}{2.448 \times (8.66^2 - 7.87^2)} = 3.1281$$

$$N_{re} = \frac{109000 \times 8.76 \times 3.1281^1}{200} \left(\frac{0.0208 \times (8.66 - 7.87)}{2 + \frac{1}{1}} \right)^1 = 81.79$$

$$\frac{dp_f}{dL} = \frac{200 \times 3.1281^1 \left(\frac{2+\frac{1}{1}}{0.0208} \right)^1}{144000 \times (8.66 - 7.87)^{1+1}} = 1.004$$

Na Figura 8.11 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 8.11: Soluções geradas pelo código para modelo lei de potência no anular considerando regime laminar

```

C:\Users\Nathan\Desktop\out  X + | v
#####
#      Menu de Perda de Carga - Potencia      #
#####

1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 2

Informe o valor do indice de consistencia [Cp eq]: 1

Velocidade no anular: 3.1281 ft/s
Reynolds no anular: 81.79
Reynolds Critico no anular: 2100
Tipo de Fluxo no poco: laminar
Perda Friccional no anular: 1.004 psi/ft

Pressione qualquer tecla para continuar. . .

```

Fonte: Produzido pelo autor.

Para o fluxo turbulento no poço realizamos os seguintes cálculos:

$$\bar{v} = \frac{158}{2.448 \times (8.66^2 - 7.87^2)} = 4.94$$

$$N_{re} = \frac{109000 \times 8.62 \times 4.94^1}{12} \left(\frac{0.0208 \times (8.66 - 7.87)}{2 + \frac{1}{1}} \right)^1 = 2118.59$$

$$\frac{dp_f}{dL} = \frac{0.013 \times 8.62 \times 4.94^2}{21.1 \times (8.66 - 7.87)} = 0.164$$

Na Figura 8.12 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 8.12: Soluções geradas pelo código para modelo lei de potência no anular considerando regime turbulento

```
#####
#           Menu de Perda de Carga - Potencia      #
#####

1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 2

Informe o valor do indice de consistencia [Cp eq]: 12

Velocidade no anular: 4.94249 ft/s
Reynolds no anular: 2120.72
Reynolds Critico no anular: 2100
Tipo de Fluxo no anular: Turbulento
Perda Friccional no Anular: 0.153337 psi/ft

Fator de Friccao: 0.0121322

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

8.2.7 Validação da pressão hidrostática no fundo do poço

Aqui, será realizada a validação dos cálculos de pressão hidrostática no fundo do poço. Inicialmente, consideraremos a condição de escoamento laminar, onde a distribuição da pressão ao longo do poço pode ser calculada de maneira mais direta. Em seguida, abordaremos o regime turbulento, ajustando o modelo para representar as variações de pressão devido às características de escoamento mais intensas.

Foram realizados os seguintes cálculos:

$$p_1 = 0.05195 \times 8.76 \times 3281 + 2175 = 3668.12$$

$$p_2 = 0.05195 \times 8.17 \times 4921 + 3668.12 = 5756.74$$

$$p_3 = 0.05195 \times 9.17 \times 3281 + 5756.74 = 7319.74$$

Na Figura 8.13 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

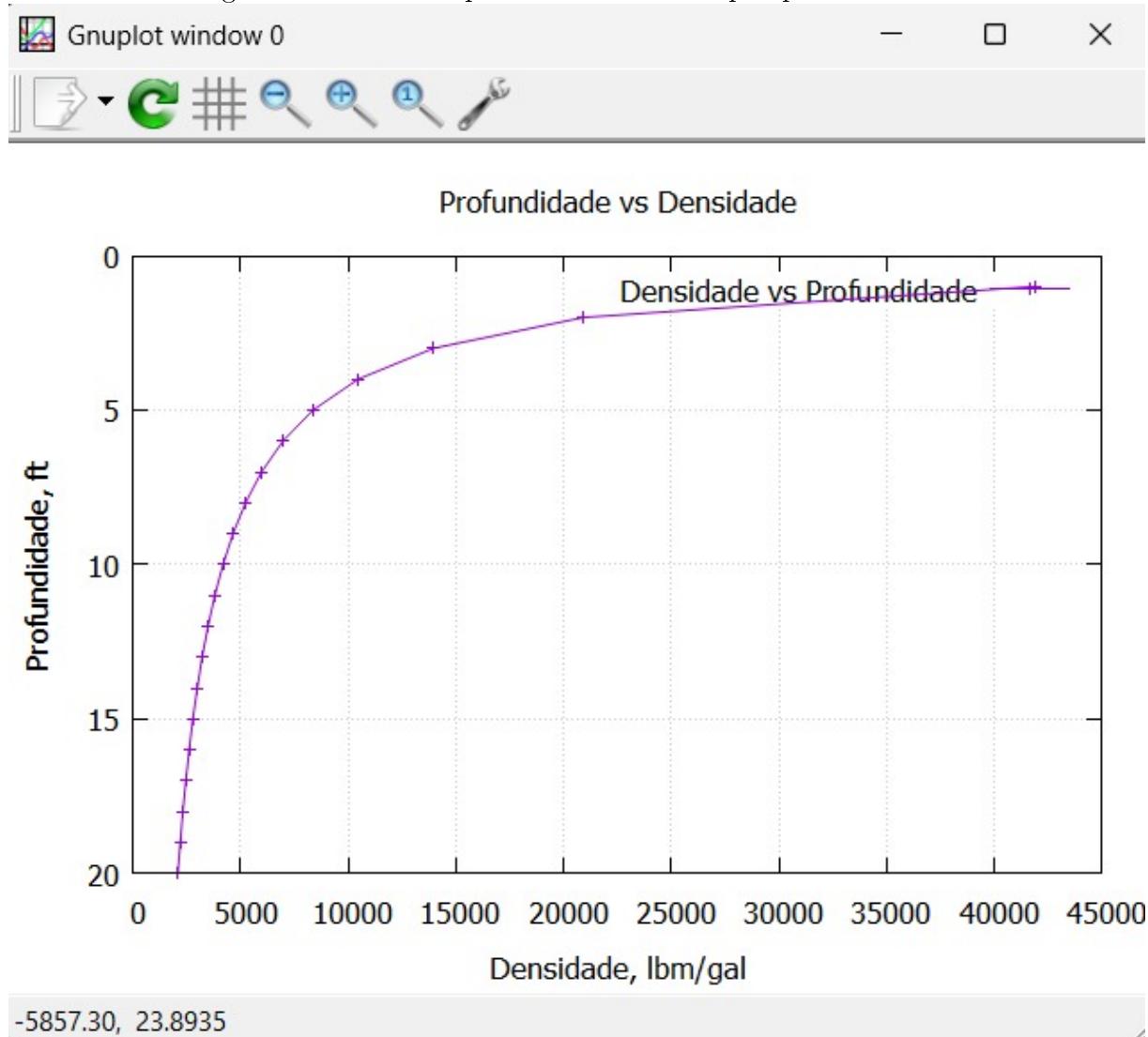
Figura 8.13: Soluções geradas pelo código para pressão hidrostática no fundo do poço

```
#####
#           Menu de Pressao Hidrostatica          #
#####  
1. Calcular Pressao Hidroestatica (Fundo de poco)
2. Calcular Pressao Hidroestatica em um Ponto do Poco
0. Voltar
Escolha: 1  
  
Pressao Hidrostatica Total: 7319.76 psi  
  
Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

Na Figura 8.14 temos um gráfico com a variação da pressão hidrostática em função da profundidade.

Figura 8.14: Gráfico pressão hidrostática por profundidade

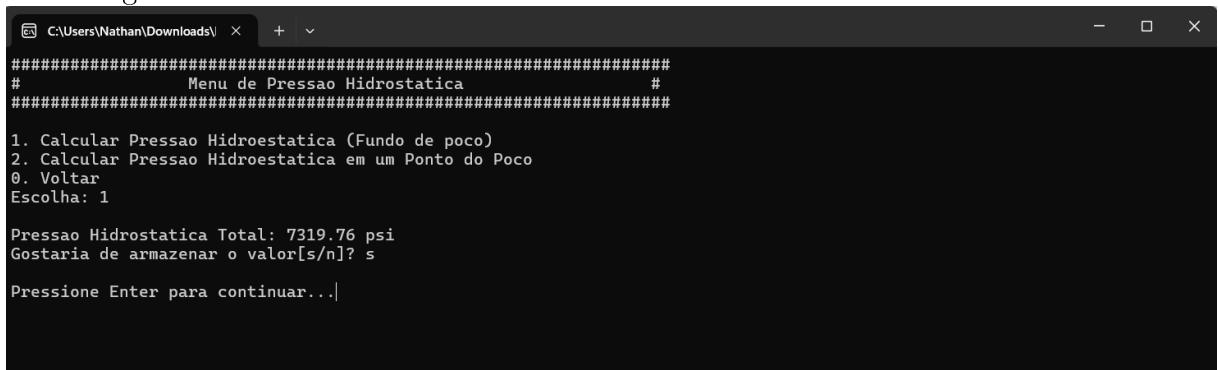


Fonte: Produzido pelo autor.

8.2.8 Validação da funcionalidade impressão

Nesta seção, será validada a classe CImpressao que tem a função de armazenar os resultados da simulação em um arquivo de saída .dat. Pode-se observar na Figura 8.15 a funcionalidade armazenar os resultados obtidos durante a simulação.

Figura 8.15: Terminal mostrando a funcionalidade armazenar os resultados



```
C:\Users\Nathan\Downloads\ + 
#####
#      Menu de Pressao Hidrostatica      #
#####
1. Calcular Pressao Hidroestatica (Fundo de poco)
2. Calcular Pressao Hidroestatica em um Ponto do Poco
0. Voltar
Escolha: 1

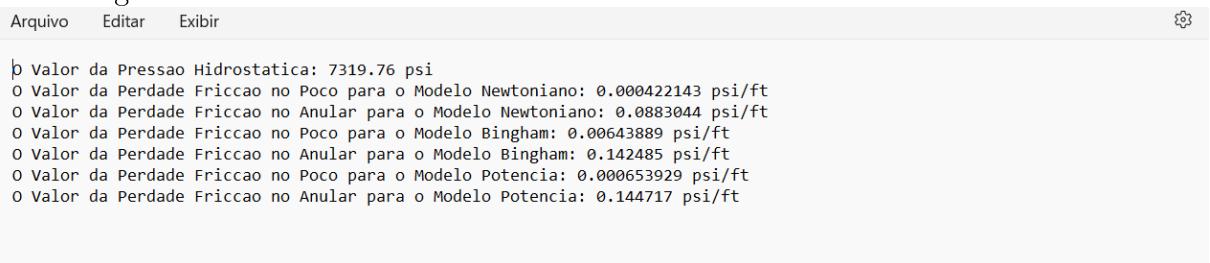
Pressao Hidrostatica Total: 7319.76 psi
Gostaria de armazenar o valor[s/n]? s

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

Na Figura 8.16 podemos ver todos os resultados gerados durante a simulação que forma armazenados no arquivo de saída .dat.

Figura 8.16: Terminal mostrando a funcionalidade armazenar os resultados



```
Arquivo   Editar   Exibir   ☰

þ Valor da Pressao Hidrostatica: 7319.76 psi
o Valor da Perdade Friccao no Poco para o Modelo Newtoniano: 0.000422143 psi/ft
o Valor da Perdade Friccao no Anular para o Modelo Newtoniano: 0.0883044 psi/ft
o Valor da Perdade Friccao no Poco para o Modelo Bingham: 0.00643889 psi/ft
o Valor da Perdade Friccao no Anular para o Modelo Bingham: 0.142485 psi/ft
o Valor da Perdade Friccao no Poco para o Modelo Potencia: 0.000653929 psi/ft
o Valor da Perdade Friccao no Anular para o Modelo Potencia: 0.144717 psi/ft
```

Fonte: Produzido pelo autor.

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

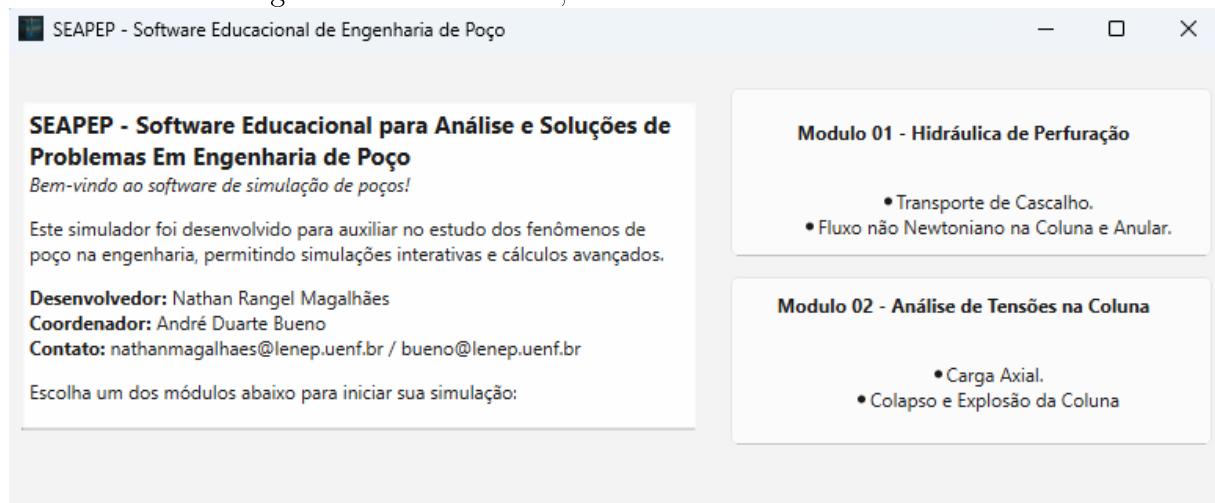
Para compilar o software, é necessário atender aos seguintes pré-requisitos:

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

10.2 Interface gráfica

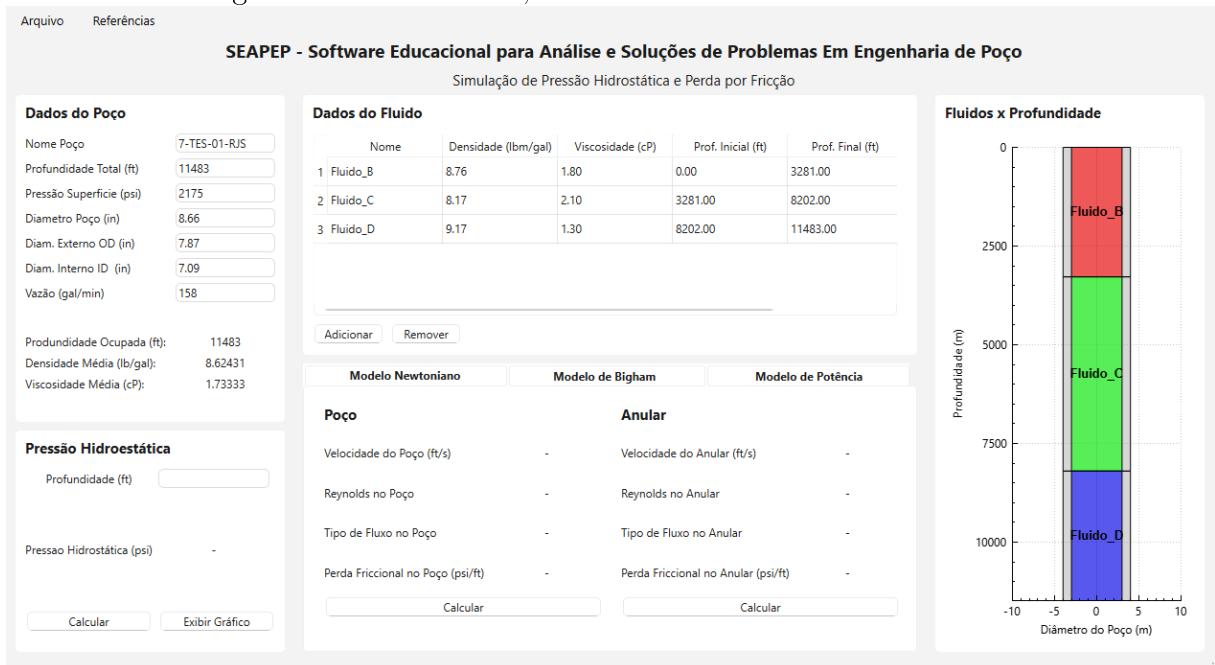
A interface do programa é apresentada nas Figuras: 10.1, 10.2 e 10.3.

Figura 10.1: Versão 1.1, Menu de interface do software



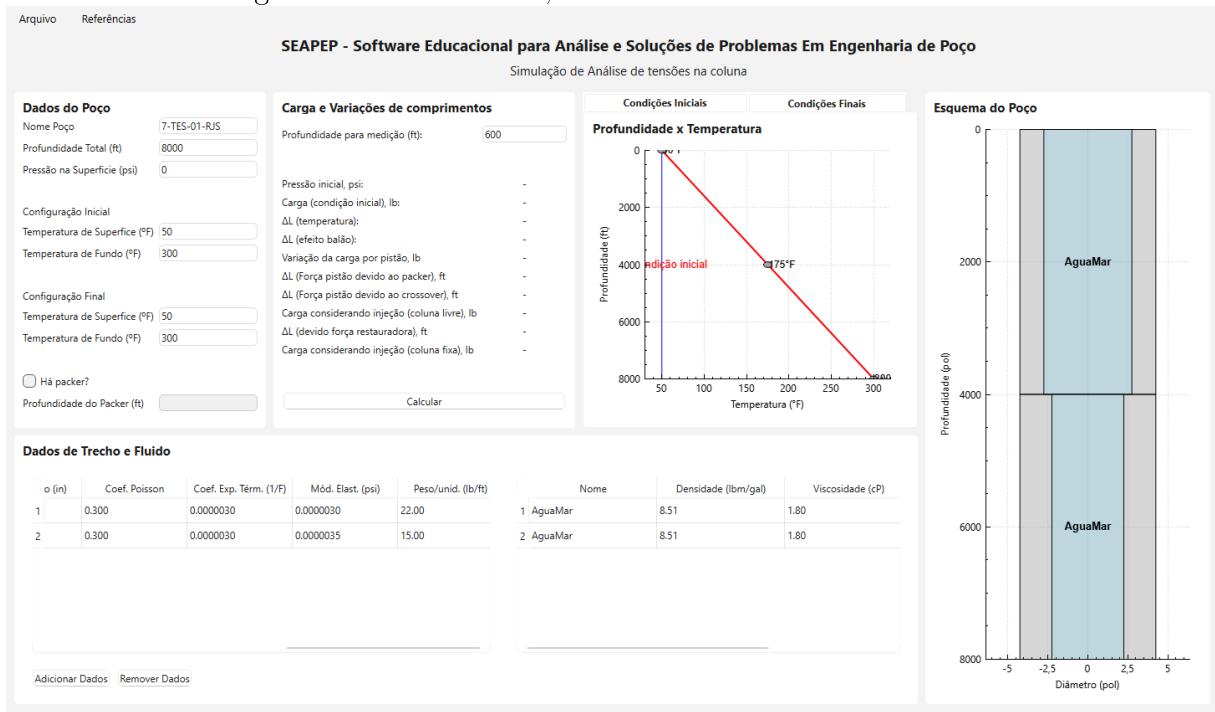
Fonte: Produzido pelo autor.

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.

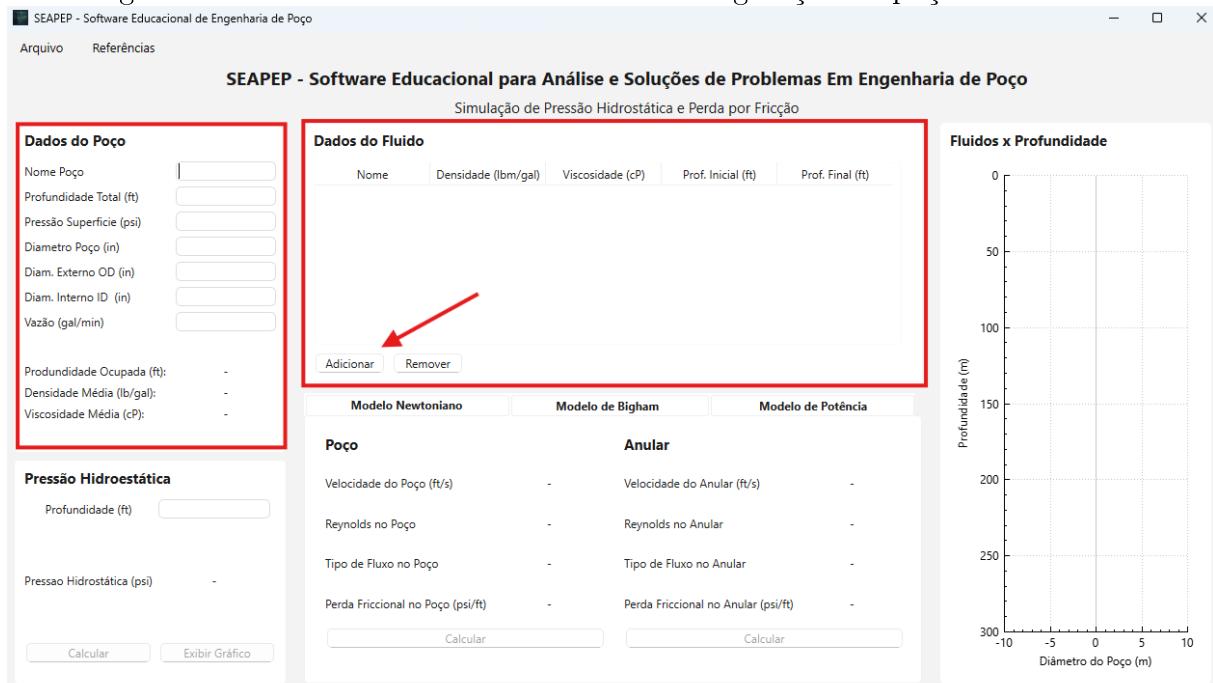
Configuração Manual via Interface Gráfica

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

```

ArquivoPoco.dat
Arquivo Editar Exibir

# 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 completação, 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

The screenshot shows a Windows Notepad window titled "ArquivoPocoMOD2.dat". The window contains two sections of configuration data:

Configuração do Poco-----

# Nome	Profundidade (ft)	Pressão Superficial (psi)	Temp. superficie, inicial (°F)	Temp. Fundo, inicial (°F)	Temp. superficie, Final (°F)	Temp. Fundo, Final (°F)	Profund. Packer (ft)
7-TES-01-RJS	8000	0	50	300	50	300	0

Configuração dos Trechos-----

# Nome Trecho	Prof. Inicial (ft)	Prof. Final (ft)	Diam. externo (in)	Diam. interno (in)	Coef. Poisson	Coef. Exp. Térn.(1/F)	Mod. Elast. (psi)	Peso/unid (lb/ft)	Nome fluido	Densidade (lbm/gal)
Trecho_A	0	4000	5.500	4.892	0.3	0.000003	0.000003	22	AguaMar	8.51
Fluido_B	4000	8000	4.500	3.958	0.3	0.000003	0.0000035	15	AquaMar	8.51

At the bottom of the window, status bar details are visible: Ln 11, Col 1 | 1.333 caracteres | 100% | Windows (CRLF) | UTF-8.

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
- Mitchell, Robert F., & Miska, Stefam Z. 2011. *Fundamentals of drilling engineering*. Society of Petroleum Engineers. 17, 32, 33, 34, 38, 39, 40