

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 ementa da disciplina LEP01353 - Engenharia de Poço, ministrada no Laboratório de Engenharia e Exploração de Petróleo (LENEP/CCT/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/ldsc/SoftwareEducacionalEngenhariaDePoco.](https://github.com/ldsc/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
2.2	Diagrama de caso de uso específico: “calcular pressão hidrostática”	20
2.1	Diagrama de caso de uso – caso de uso geral	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	43
4.2	Diagrama de sequência - Módulo reológico	46
4.3	Diagrama de comunicação para caso do cálculo usando modelo reológico . .	47
4.4	Diagrama de máquina de estado do objeto CSIMULADORPERDATUBULACAO	48
4.5	Diagrama de atividades: CObjetoPoco::Carga(double profundidade, bool inicio)	49
5.1	Diagrama de componentes	52
6.1	Versão 1.0, interface do software	54
6.2	Versão 1.1, interface do software	55
6.3	Versão 2.0, interface do software	57
6.4	Versão 2.6, interface do software	58
6.5	Versão 3.0, Menu de interface do software	60
6.6	Versão 3.0, interface do módulo 01 do software	60
6.7	Versão 3.0, interface do módulo 02 software	61
8.1	Solução gerada para calculo da pressão hidrostática	154
8.2	Soluções geradas pelo código para modelo Newtoniano no poço e anular .	156
8.3	Soluções geradas pelo código para modelo de Bingham no poço e anular .	158
8.4	Soluções geradas pelo código para modelo de lei de Potência no poço e anular	160

8.5	Solução gerada para variação do comprimento do tudo devido a força pistão	162
8.6	Solução gerada para variação do comprimento do tudo devido ao efeito balão	163
8.7	Solução gerada para variação do comprimento do tudo devido ao efeito da temperatura	164
8.8	Opção de Salvar	165
8.9	Modelo do Arquivo .dat	165
8.10	Opção de importação de arquivos no software	166
9.1	Logo e documentação do <i>software</i>	169
9.2	Código fonte da classe CSimuladorPoco, no <i>Doxxygen</i>	169
10.1	Versão 1.1, Menu de interface do software	172
10.2	Versão 1.1, interface do modulo 01 do software	172
10.3	Versão 1.1, interface do modulo 02 software	173
10.4	Acesso às funcionalidades de configuração do poço e dos fluidos	175
10.5	Exemplo de estrutura de arquivo .dat para importação de dados	176
10.6	Exemplo de estrutura de arquivo .dat para importação de dados	177

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	20

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	19
2.4	Casos de Uso do Software	19
2.4.1	Diagrama de caso de uso geral	19
2.4.2	Diagrama de caso de uso específico	20
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	42
4.1 Diagramas de classes	42
4.1.1 Dicionário de Classes	44
4.2 Diagrama de Sequência – Eventos e Mensagens	45
4.2.1 Diagrama de Sequência	45
4.3 Diagrama de Comunicação – Colaboração	47
4.4 Diagrama de Máquina de Estado	48
4.5 Diagrama de Atividades	48
5 Projeto	50
5.1 Projeto do sistema	50
5.2 Diagrama de componentes	51
6 Ciclos de planejamento/detalhamento	53
6.1 Versão 1.0 – Interação via Terminal e Geração de Gráficos com Gnuplot . .	53
6.2 Versão 1.1 – Otimização de Entrada, Validação e Armazenamento Automático de Dados	54
6.3 Versão 2.0 – Interface Gráfica, Amigável e Intuitiva Utilizando o Framework Qt	56
6.4 Versão 2.6 – Consolidação Visual, Barra de Tarefas e Automação de Processos	57
6.5 Versão 3.0 – Navegação por Módulos e Simulação Mecânica de Variação de Comprimento (ΔL)	59
7 Ciclos Construção - Implementação	62
7.1 Código-Fonte (modelo)	62
7.1.1 Código Qt (interface)	135
8 Resultados	152
8.1 Metodologia de Validação do Software	152
8.2 Casos de Teste	153
8.3 Validação da pressão Hidrostática	153
8.4 Validação da perda de fricção pelo modelo Newtoniano no Poço e Anular .	154
8.5 Validação da perda de fricção pelo modelo <i>Bingham</i> no Poço e Anular . .	156
8.6 Validação da perda de fricção pelo modelo Potência no Poço e Anular . .	158
8.7 Validação da variação do comprimento do tubo devido a força pistão . .	160
8.8 Validação da variação do comprimento do tubo devido ao efeito balão . .	162
8.9 Validação da variação do comprimento do tubo devido ao efeito da temperatura	163
8.10 Validação da importação e exportação do documento (“salvar como...”) . .	165
8.11 Conclusão	166

9 Documentação	168
9.1 Documentação do usuário	168
9.2 Documentação do desenvolvedor	168
10 Manual do desenvolvedor	170
10.1 Instalação	170
10.1.1 Dependências obrigatórias para rodar o software	170
10.1.2 Bibliotecas padrão da linguagem C++ (não precisam ser instaladas separadamente)	171
10.1.3 Observações importantes	171
10.1.4 Execução sem compilador	171
10.2 Interface gráfica	172
10.3 Funcionalidades	173
10.3.1 Módulo 1 – Hidráulica de Perfuração	174
Referências Bibliográficas	178

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 o paradigma da programação orientada a objetos, a modelagem UML, a linguagem de programação C++ e a biblioteca gráfica Qt, 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 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 usando orientação a objetos e UML.
 - Calcular propriedades hidrodinâmicas e reológicas associadas ao poço.
 - Realizar simulações computacionais para teste e validação dos algoritmos desenvolvidos.
 - Desenvolver o manual do usuário, um manual simplificado para orientar o uso do software.
 - Desenvolver o manual técnico científico, um manual que apresenta os objetivos, metodologia, conceitos teóricos, ferramentas utilizadas e os códigos desenvolvidos e testes do software (este documento).

1.3 Metodologia utilizada

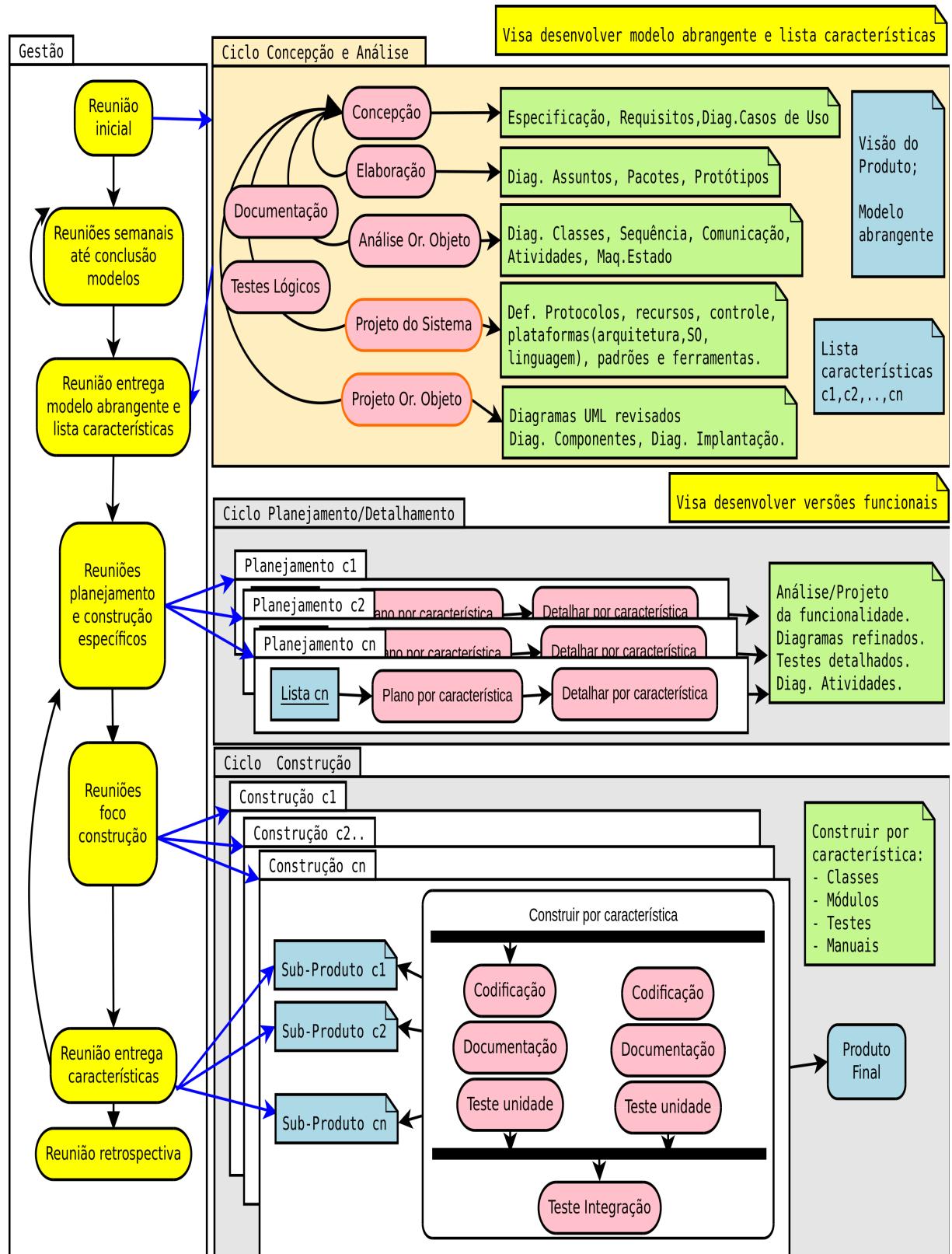
A metodologia empregada para o desenvolvimento do software fundamenta-se nos ciclos propostos pelo Prof. André Duarte Bueno, cujos materiais foram disponibilizados via GitHub do LDSC e complementados por conteúdos ministrados nas disciplinas:

- LEP01348 - Introdução ao Projeto de Engenharia ([site](#))
- LEP01447 - Programação Orientada a Objetos com C++([site](#)).
- LEP01449 - Projeto de Software Aplicado à Engenharia ([site](#)).

A metodologia é descrita aqui e um repositório com o modelo completo [aqui](#).

O processo foi estruturado em três fases: ciclo concepção e análise, ciclo planejamento detalhado e ciclo construção (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. Para comunicações usamos o Telegram e as reuniões foram realizadas de forma presencial ou remota usando Google Meet.

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



Fonte: Apostila da disciplina "LEP01348 - Introdução ao Projeto de Engenharia" 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 (ASCII), oferece visualização gráfica dos resultados e permite a exportação dos dados em formato .dat (ASCII), 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 Soluções 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, <i>packer</i> e <i>crossover</i> nas análises de carga.
RF-11	O sistema deve contemplar variações térmicas, efeito balão, atuação de pistão, <i>packer</i> e <i>crossover</i> nas análises de carga.
RF-12	O sistema deve permitir a navegação por meio de digitação direta de números no menu de opções, otimizando o fluxo de trabalho e eliminando a necessidade de múltiplos comandos sequenciais.
RF-13	O sistema deve salvar automaticamente os dados da simulação em arquivos nomeados conforme o poço selecionado, mantendo um histórico completo das ações do usuário
RF-14	O sistema deve realizar a verificação automática de entradas inválidas, garantindo estabilidade e prevenindo falhas durante a execução.
RF-15	O sistema deve oferecer atalhos de teclado, como Ctrl+N para nova simulação e Ctrl+S para salvar, otimizando a interação com o usuário.
RF-16	O sistema deve disponibilizar uma seção de ajuda com instruções de uso e descrição dos modelos implementados.

2.3.2 Requisitos Não Funcionais

RNF-01	Suas primeiras versões devem suportar os sistemas operacionais GNU/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.
RNF-08	Os arquivos devem usar a codificação de caracteres UTF-8

2.4 Casos de Uso do Software

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

2.4.1 Diagrama de caso de uso geral

As condições do caso de uso geral são apresentadas na Tabela 2.2. Este cenário é representado graficamente pelo diagrama de caso de uso geral da Figura 2.1. O mesmo 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.

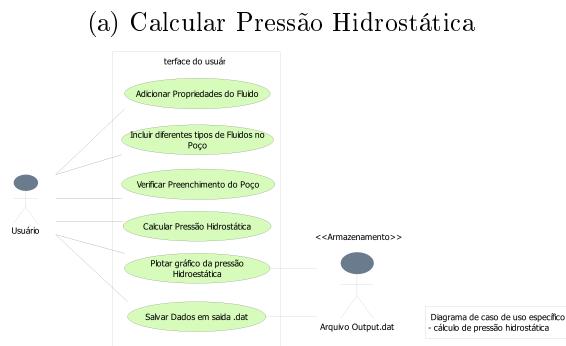
Tabela 2.2: Caso de uso 1

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 fluido 2. Adicionar propriedades do poço 3. Incluir diferentes tipos de fluidos no poço 4. Calcular pressão hidrostática do poço 5. Calcular densidade do fluido 6. Calcular a queda de pressão devido a perdas por fricção 7. Calcular Variações de comprimento 8. Plotar Perfis do Poço 9. Exportar imagem da tela 10. Salvar dados em saída .dat 11. Importar dados em arquivo .dat

2.4.2 Diagrama de caso de uso específico

O caso de uso específico da Figura 2.2 mostra o cenário onde o usuário deseja “Calcular a pressão hidrostática” e “Calcular densidade do fluido” configurados no poço.

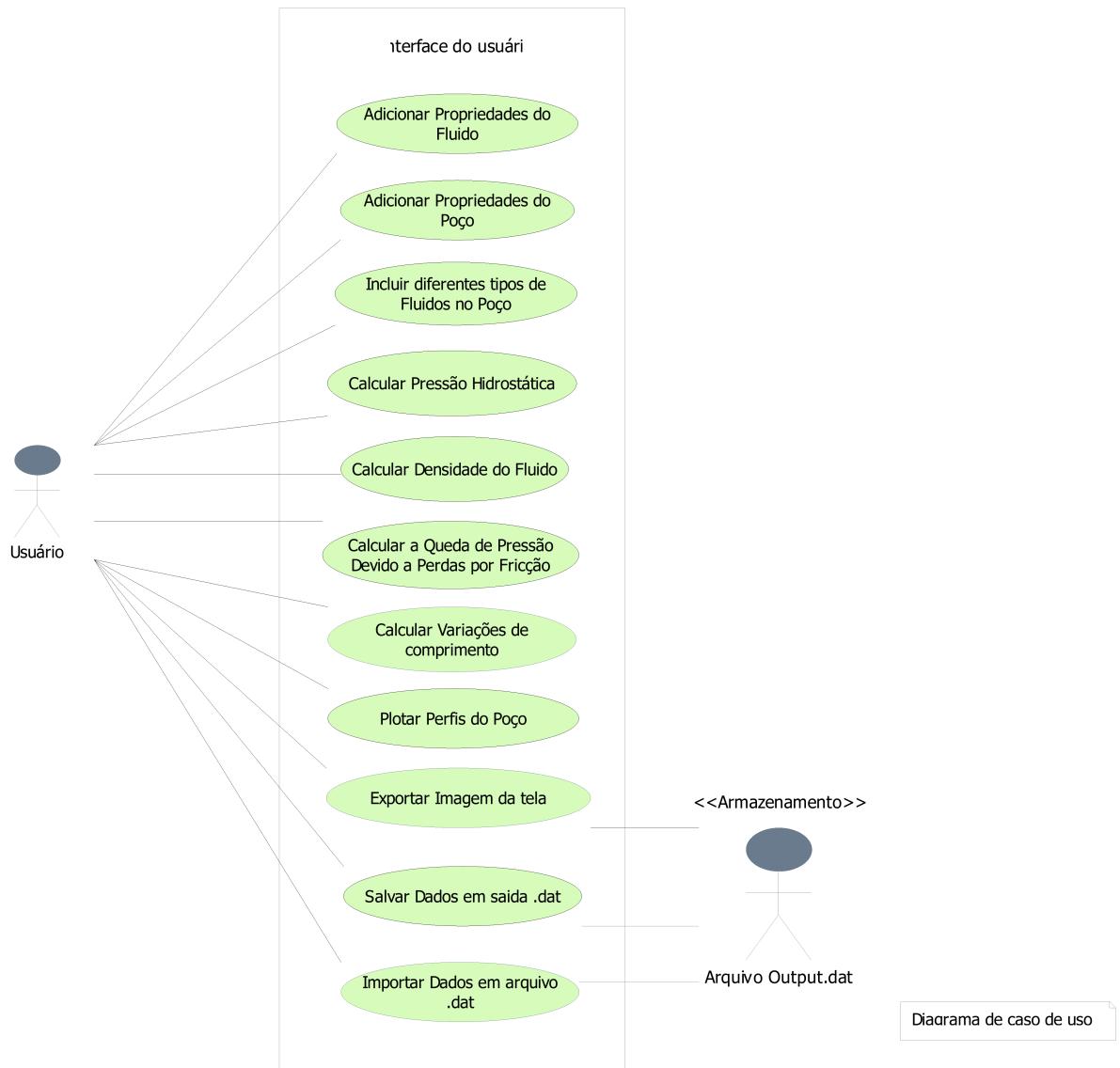
Figura 2.2: Diagrama de caso de uso específico: “calcular pressão hidrostática”



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



Fonte: Produzido pelo autor.

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];
- v é o coeficiente de Poisson;
- E Módulo de elasticidade [psi];
- ΔP_{in} Variação de pressão na parte interna do poço [psi];
- ΔP_{out} Variação de pressão no anular [psi];
- ΔL_b Variação de comprimento devido efeito balão [ft];
- ΔL_{packer} Variação de comprimento devido força pistão [ft];
- $\Delta L_{crossover}$ Variação de comprimento devido ao crossover [ft];

3.2.3 Hidráulica de Perfuração

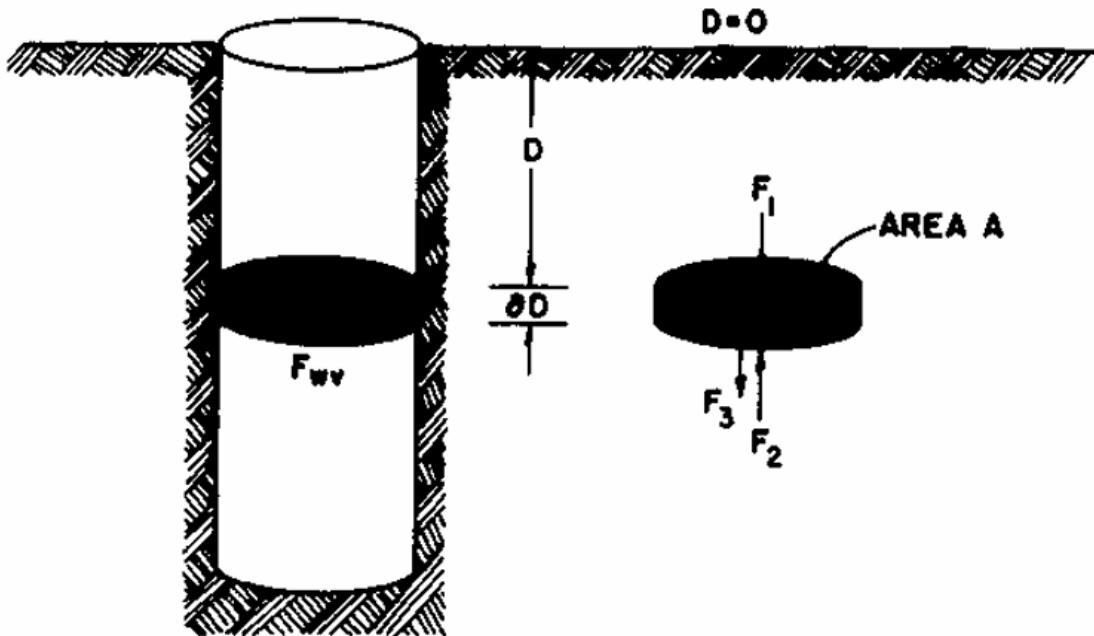
Na engenharia de perfuração, um fluido de perfuração possui três funções principais: transportar cascalho, prevenir o influxo de fluidos e manter a estabilidade do poço. Para cumprir essas funções, o fluido depende do seu escoamento na tubulação e das pressões associadas. Para que o engenheiro possa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

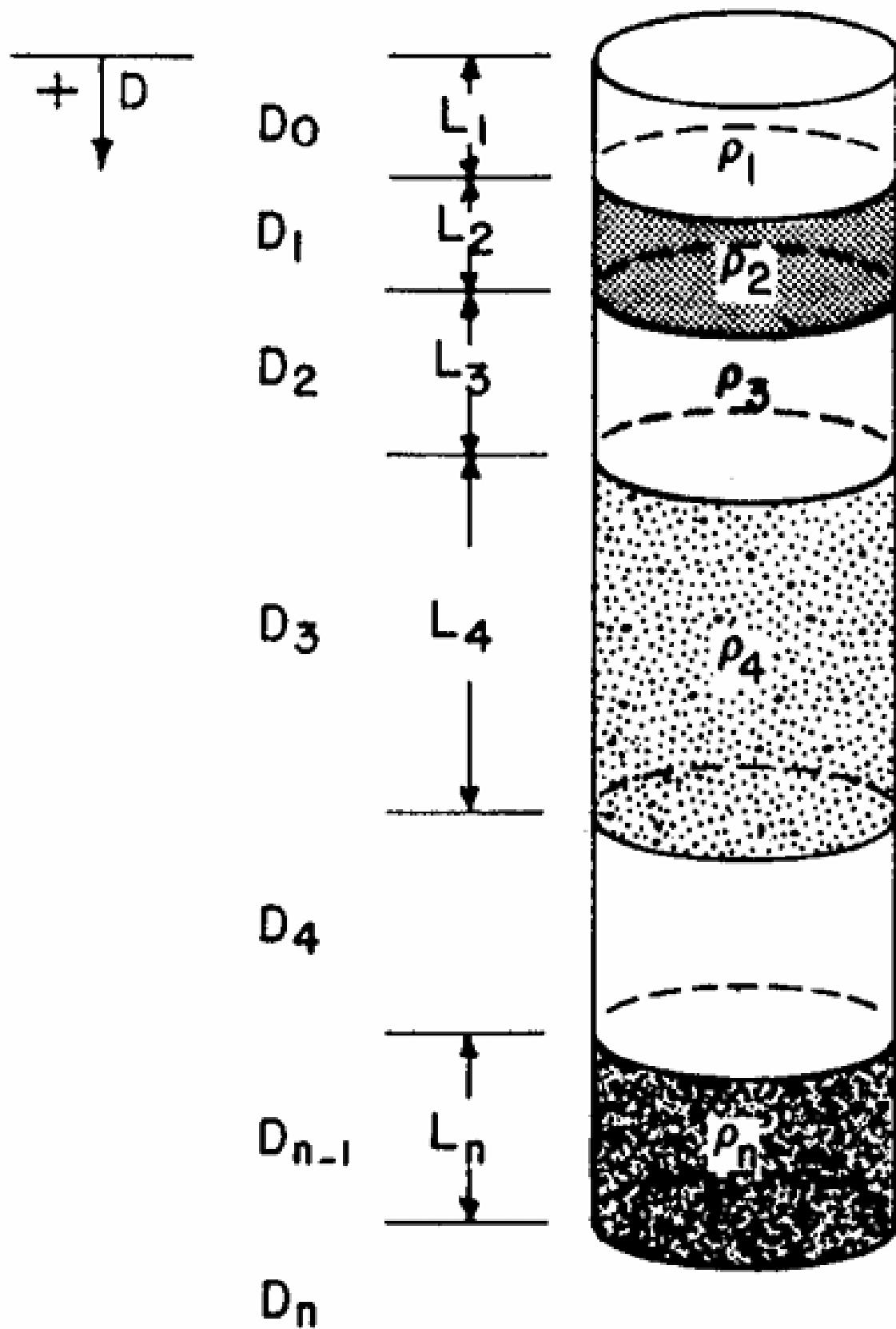
$$p = p_0 \exp \left(\frac{M \Delta Z}{1544 z T} \right) \quad (3.4)$$

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.

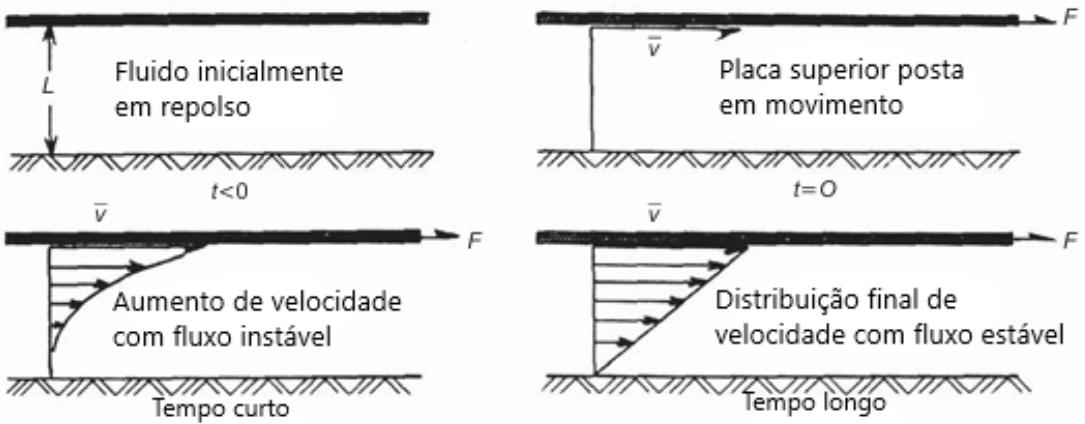
Na maioria dos casos, as propriedades elásticas dos fluidos de perfuração e seus efeitos durante o escoamento no poço são desprezíveis, sendo consideradas apenas as forças viscosas nos cálculos. Entretanto, com o avanço tecnológico, lamas cada vez mais complexas vêm sendo desenvolvidas, e, portanto,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

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

Onde

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

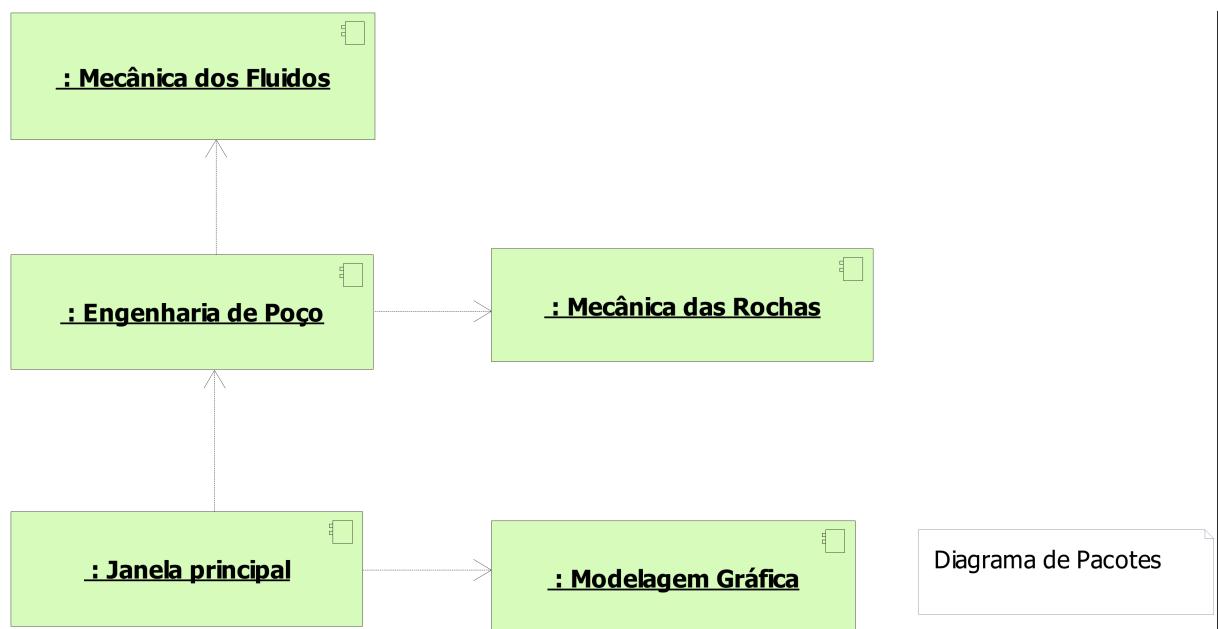
- Pacote engenharia de poço:
 - O pacote engenharia de poço é responsável por relacionar os pacotes mecânicas dos fluidos, mecânica das rochas e equações analíticas de forma a tornar possível e coerente os resultados obtidos pela simulação.
- Pacote mecânica dos fluidos:
 - É o pacote que relaciona todas 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



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 - *Unified Modeling Language* (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).

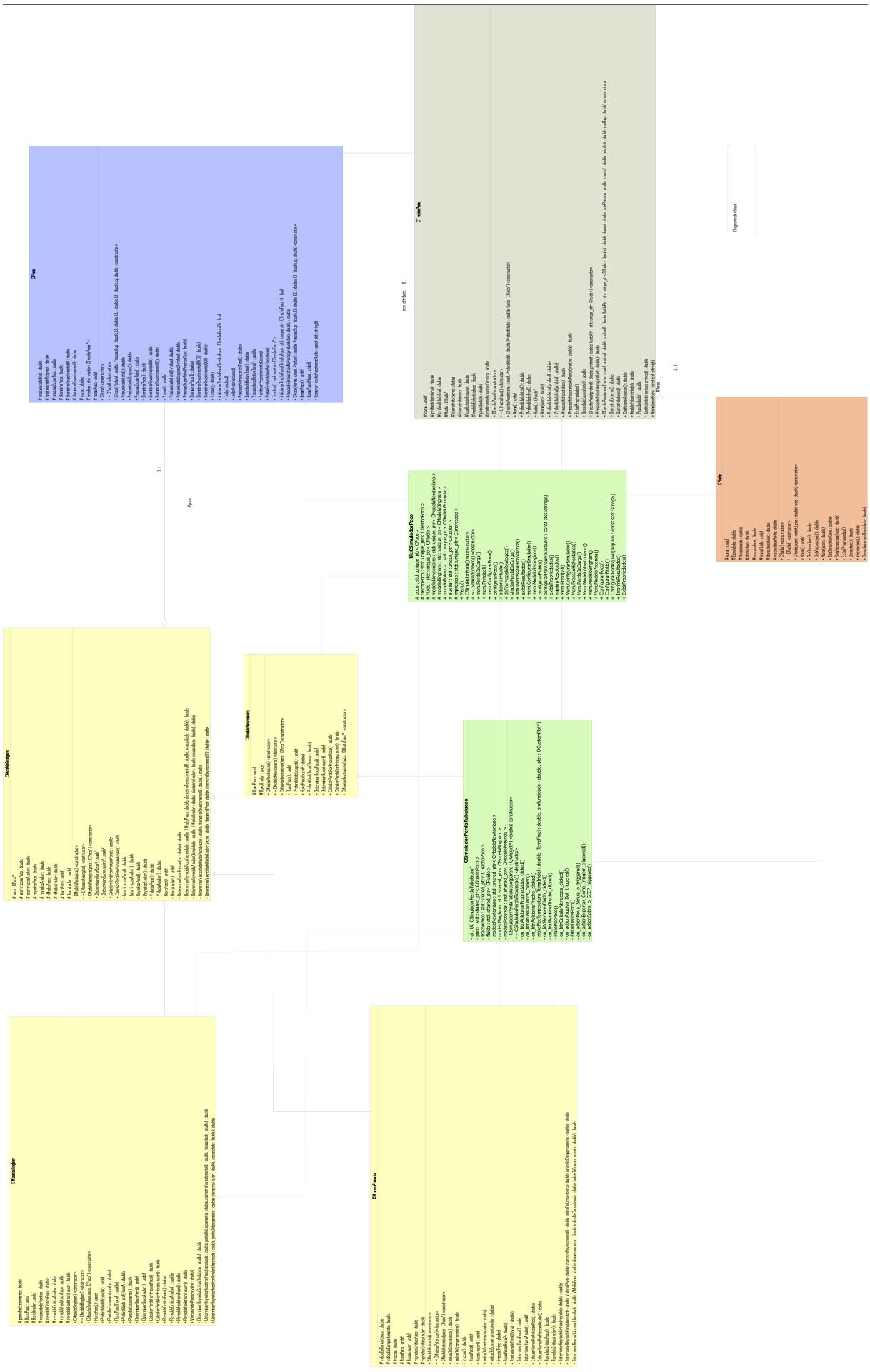
4.1 Diagramas de classes

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 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 4.1. 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 classe integra os modelos reológicos e executa os cálculos associados às propriedades dos fluidos e trechos do poço.
- **CSimuladorPerdaTubulacao:** esta classe é 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 com funcionalidades específicas de acordo com o nome. Foram criadas com o **Qt Creator**, e são responsáveis por fornecer interfaces gráficas para entrada e visualização de dados.
- **QCustomPlot:** biblioteca externa utilizada para renderização de gráficos científicos, como perfis de pressão e densidade (site).

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

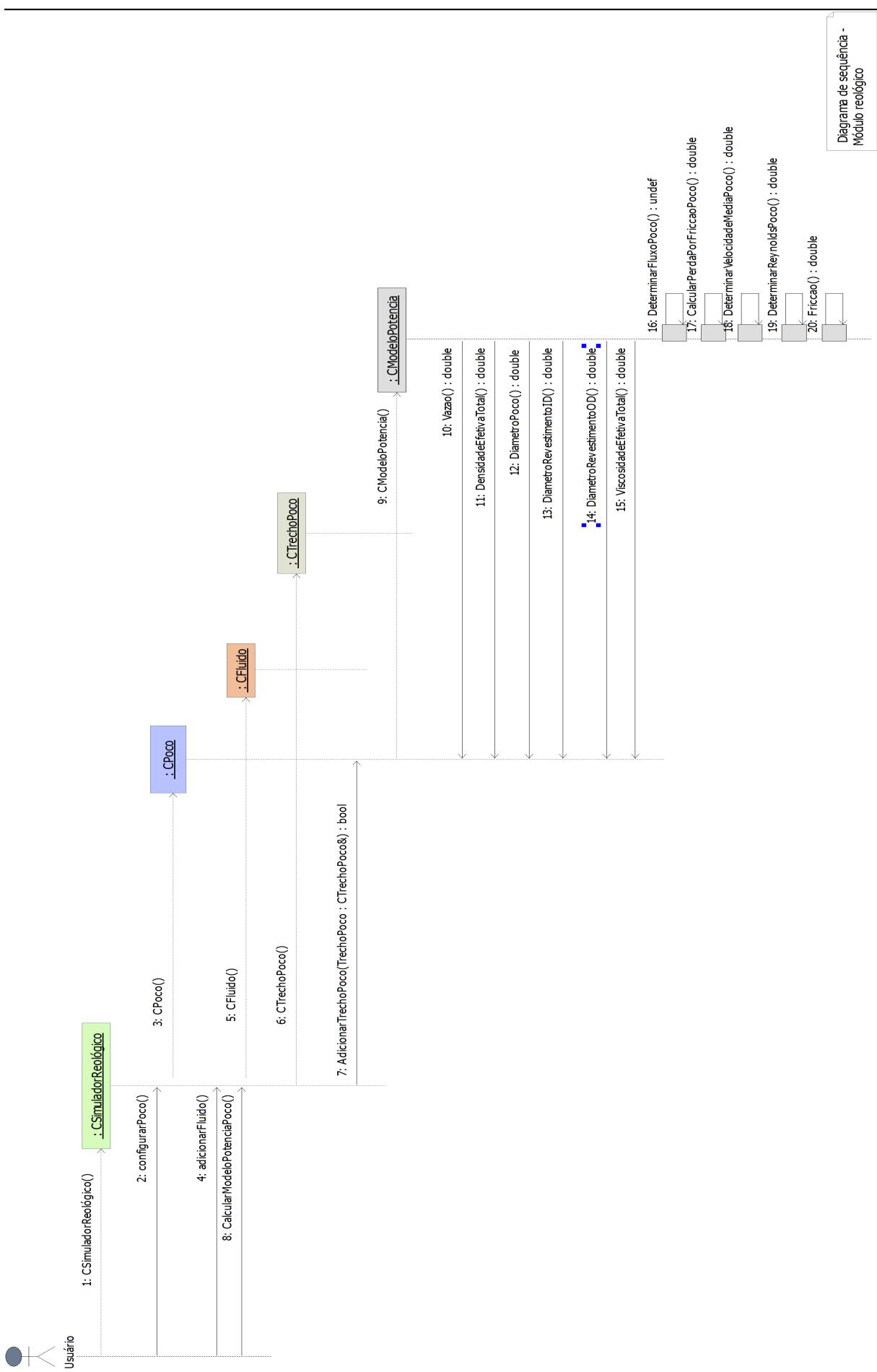
O diagrama de sequência descreve a interação entre os objetos do sistema e os elementos externos, evidenciando a ordem temporal das mensagens trocadas durante a execução de uma funcionalidade. Ele apresenta o fluxo de controle do sistema de forma cronológica, detalhando as chamadas de métodos e as respostas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

O diagrama de comunicação do módulo reológico 4.2 mostra a troca de mensagens entre os objetos centrais durante a simulação de propriedades hidráulicas. O objeto CSimuladorReologico representa a interface principal que orquestra toda a lógica. Ele cria um poço (CObjetoPoco), adiciona trechos com fluidos (CTrechoPoco + CFluido), e por fim seleciona um modelo reológico (CModeloPotencia, CModeloBingham, ou CModeloNewtoniano). As setas indicam chamadas de função entre os objetos, como criação de trechos, atribuição de fluido e cálculo de propriedades como Reynolds, perda por fricção e tipo de escoamento. Essa comunicação em rede reforça o acoplamento entre os componentes do simulador e garante que todos os dados físicos estejam interligados corretamente.

Figura 4.2: Diagrama de sequência - Módulo reológico



Fonte: Produzido pelo autor.

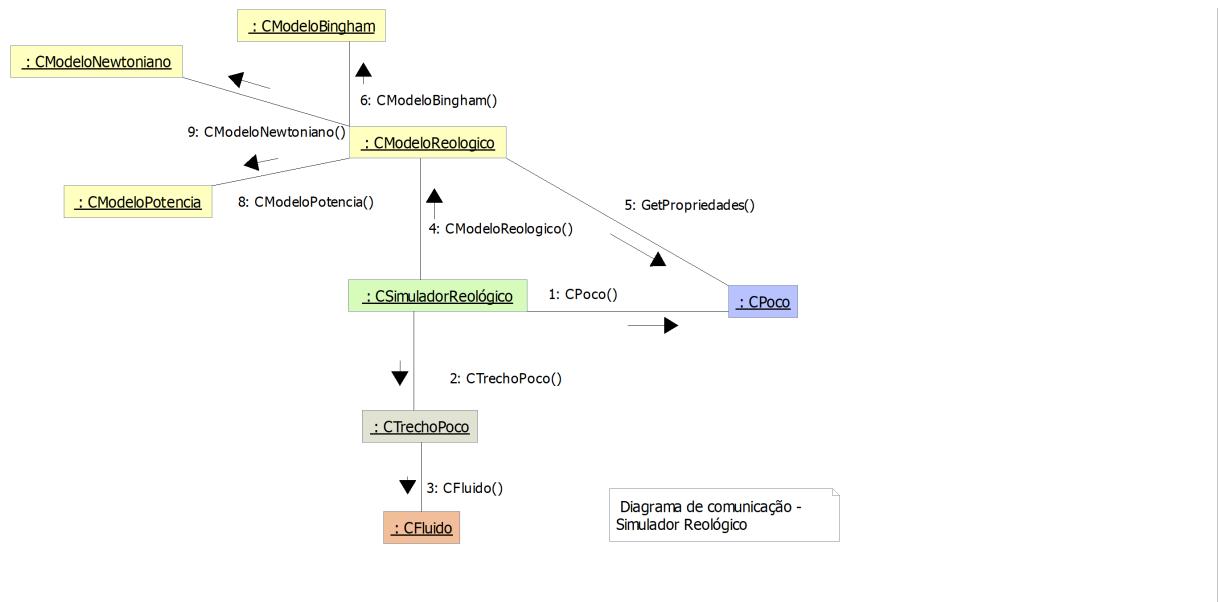
No módulo térmico-mecânico, o diagrama de comunicação evidencia as interações entre a interface (CSimuladorPerdaTubulacao), o poço (CObjetoPoco), os trechos (CTrechoPoco) e os fluidos (CFluido). Após definir os dados gerais do poço, o usuário adiciona trechos, e o simulador realiza cálculos como variações de comprimento axial (ΔL) causadas por temperatura, efeito balão e efeito pistão com ou sem packer. As setas indicam chamadas sequenciais de funções específicas que somam as contribuições individuais para estimar a deformação total e a carga restauradora. O diagrama permite visualizar o encadeamento lógico necessário para realizar os cálculos de forma automatizada e confiável.

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.

Neste diagrama de comunicação UML, visualizamos como os objetos se relacionam durante a execução da simulação reológica. A classe CSimuladorReológico atua como controlador principal, criando o objeto CObjetoPoco e adicionando trechos com seus respectivos fluidos. Em seguida, ao selecionar um modelo reológico (como CModeloPotencia), o simulador instancia o modelo passando o poço como argumento. O modelo então interage diretamente com os dados do poço e dos fluidos para realizar os cálculos de perda por fricção, tipo de escoamento e número de Reynolds.

Figura 4.3: Diagrama de comunicação para caso do cálculo usando modelo reológico



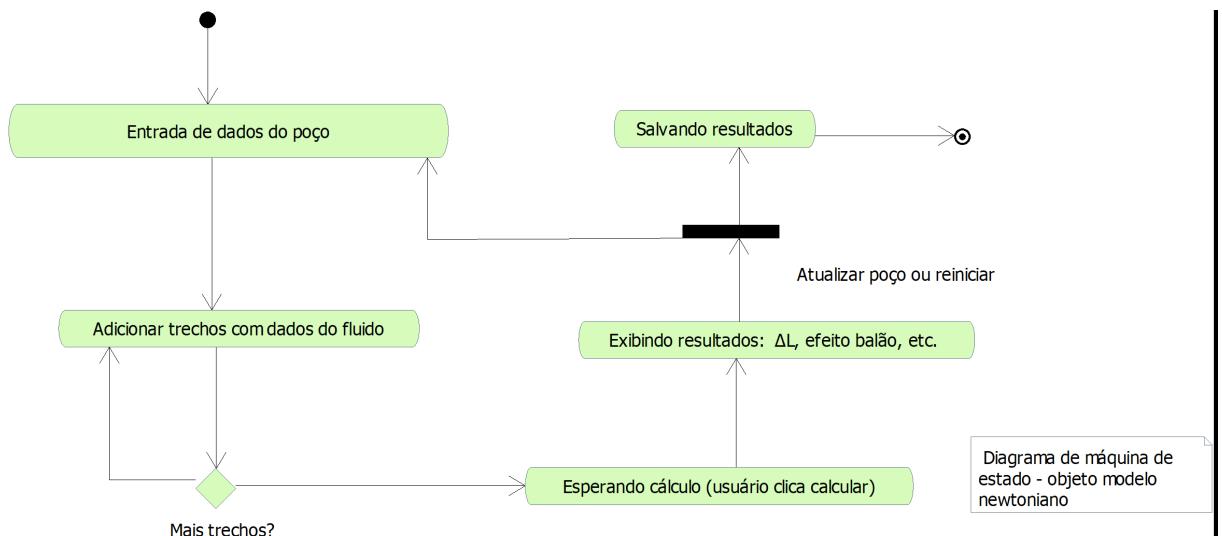
Fonte: Produzido pelo autor.

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.

O diagrama de máquina de estados do módulo de simulação térmico-mecânica 4.4 descreve as possíveis transições entre os estados do sistema conforme o usuário interage com a interface. A simulação começa com a definição do poço, seguida pela adição de trechos e fluidos. Após o preenchimento dos dados, o sistema entra em estado de cálculo, onde são computadas as variações axiais (ΔL) e forças atuantes. O usuário pode então modificar os dados, reiniciar a simulação ou exportar os resultados. Essa estrutura garante robustez e previsibilidade no fluxo de execução do software.

Figura 4.4: Diagrama de máquina de estado do objeto CSIMULADORPERDATUBULACAO



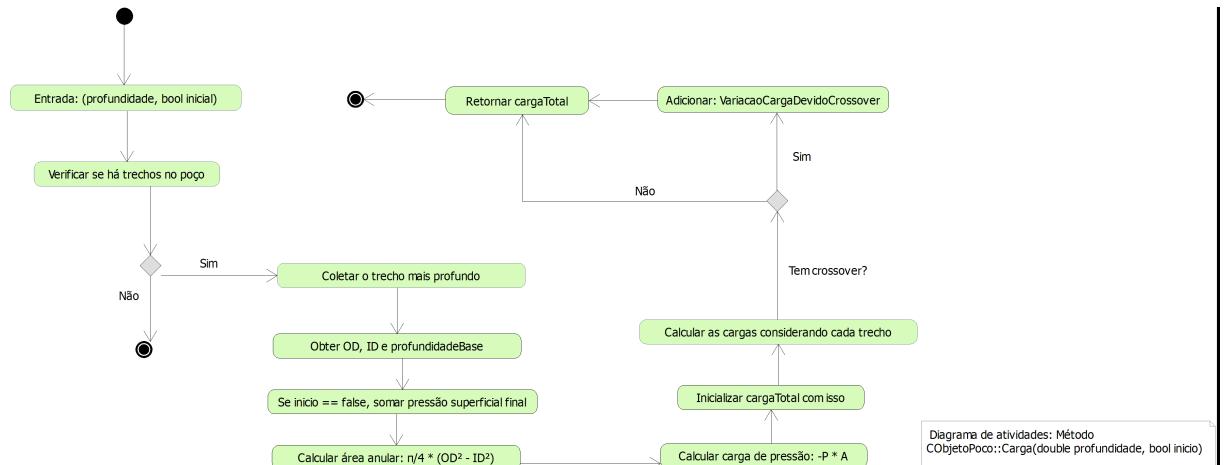
Fonte: Produzido pelo autor.

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 **Carga**, pertencente à classe **CObjetoPoco**.

O método Carga calcula a carga axial total atuante sobre a coluna na profundidade especificada, levando em conta o peso dos trechos acima, a carga por pressão hidrostática no fundo do poço, e os efeitos mecânicos em interfaces de diâmetro (crossover). O fluxo é bifurcado com base no parâmetro inicio, que altera o tratamento da pressão superficial final e da carga pistão. O diagrama de atividade acima explicita esse encadeamento lógico passo a passo, facilitando a validação do código e a compreensão dos cálculos estruturais realizados pelo simulador.

Figura 4.5: Diagrama de atividades: CObjetoPoco::Carga(double profundidade, bool inicio)



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 apresentado na Figura 3.4 ilustra a arquitetura modular do software de simulação em Engenharia de Poço, evidenciando a organização das dependências entre bibliotecas e módulos fundamentais. Esse modelo visa garantir a escalabilidade, legibilidade e manutenção eficiente do código-fonte.

A Biblioteca QT está destacada como um subsistema central, sendo composta pelos módulos QtCore, QtGui, QtWidgets e QPrintSupport, que oferecem suporte à interface gráfica, manipulação de eventos, impressão e componentes básicos da aplicação. Esses módulos se comunicam diretamente com a Biblioteca C++ / STL, responsável por prover estruturas de dados e algoritmos da linguagem padrão C++.

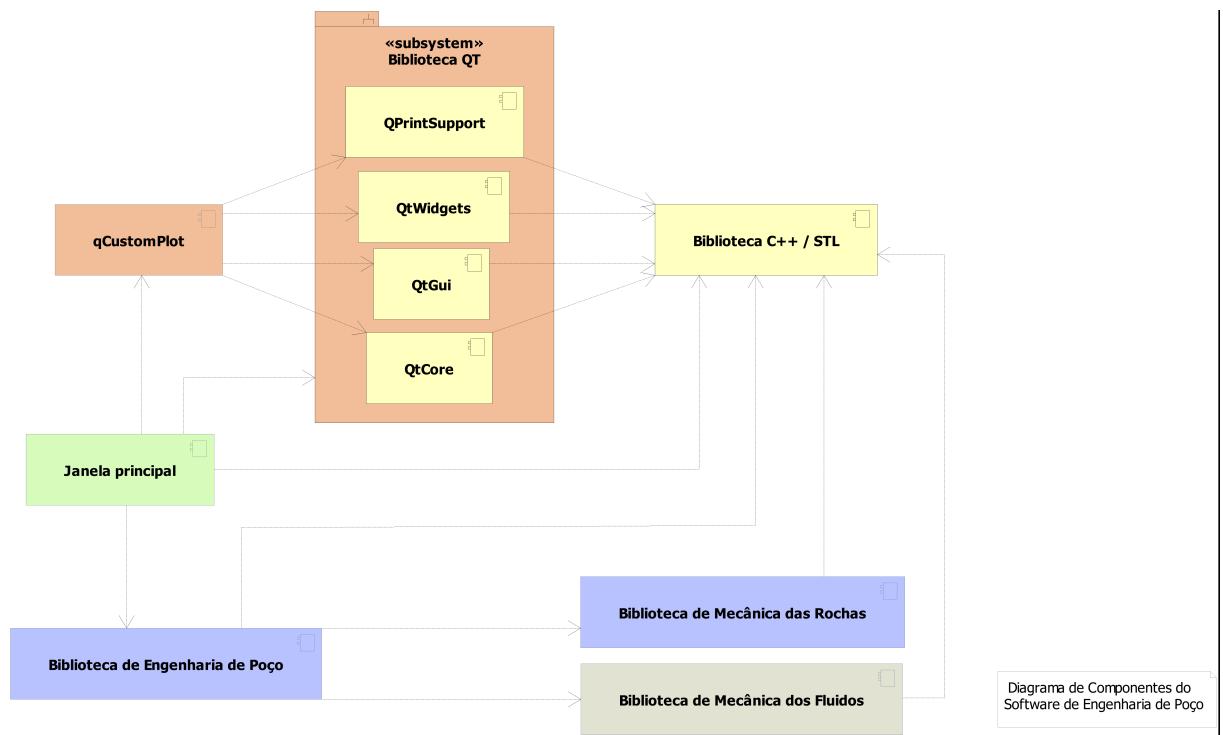
A interface gráfica do sistema é centralizada na Janela Principal, que depende diretamente da biblioteca qCustomPlot para renderização de gráficos técnicos, e se comunica com os módulos do QT. A partir dessa interface, são acionadas funcionalidades específicas implementadas em três bibliotecas técnicas do domínio:

- **Biblioteca de Engenharia de Poço:** agrupa funcionalidades gerais para cálculos de projeto, como propriedades do poço, tubulação e perfil térmico.
- **Biblioteca de Mecânica das Rochas:** trata dos aspectos relacionados ao comportamento mecânico da formação geológica, incluindo tensões, colapsos e integridade estrutural.
- **Biblioteca de Mecânica dos Fluidos:** dedicada à simulação de propriedades dos fluidos de perfuração e completação, modelos reológicos e perda de carga.

Todas essas bibliotecas especializadas são implementadas em C++ e utilizam a Biblioteca C++ / STL, demonstrando uma separação clara entre a lógica computacional (backend) e a interface gráfica (frontend).

Esse modelo de componentes reflete boas práticas de engenharia de software, como a separação de responsabilidades, modularização e reuso de bibliotecas externas. Tal arquitetura também facilita a substituição ou atualização de partes do sistema com baixo acoplamento entre módulos.

Figura 5.1: Diagrama de componentes



Fonte: Produzido pelo autor.

Capítulo 6

Ciclos de planejamento/detalhamento

Apresenta-se neste capítulo as versões do software desenvolvido.

6.1 Versão 1.0 – Interação via Terminal e Geração de Gráficos com Gnuplot

Versão: 1.0

Componentes: Módulo de cálculo com interface CLI/terminal e biblioteca CGnuplot (Gnuplot)

Desenvolvida na disciplina: LEP01449 - Projeto de Software Aplicado à Engenharia (site).

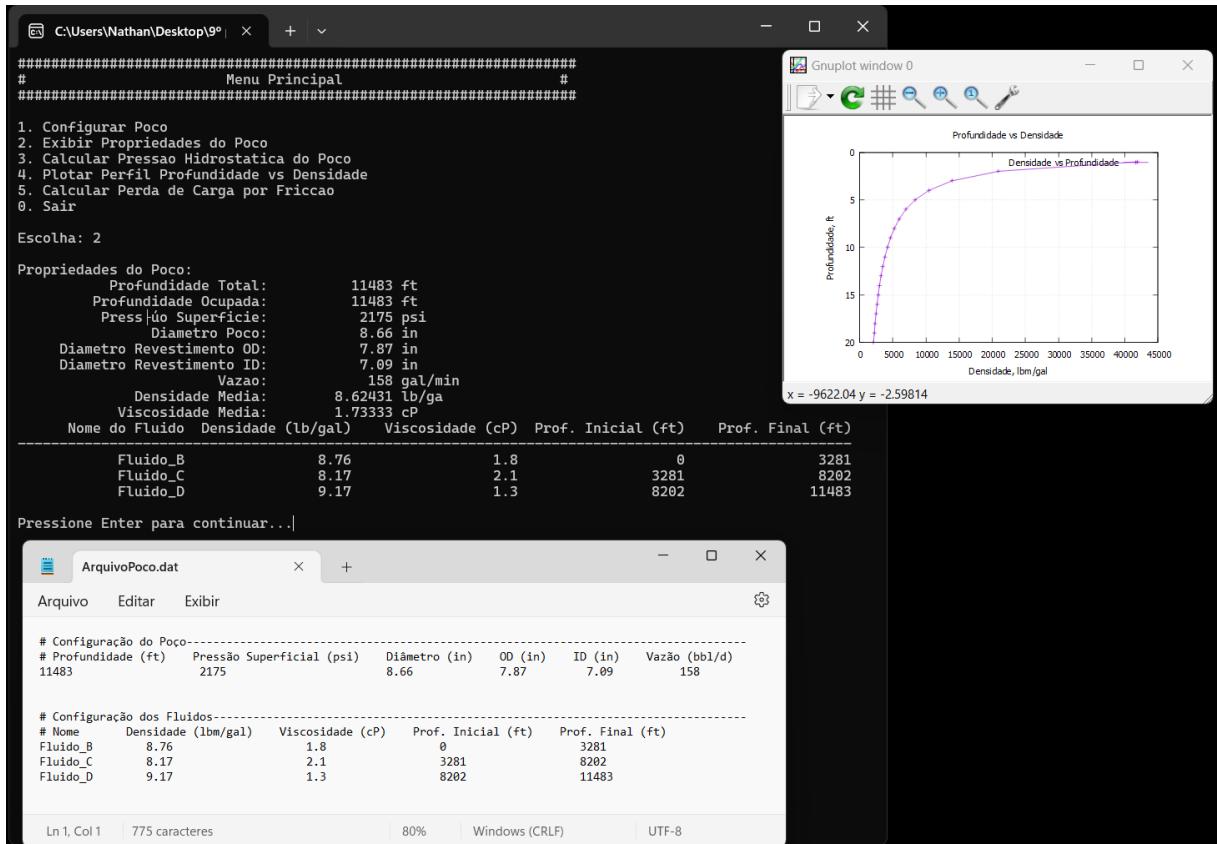
Registro da versão 1.0: v1.0 Lançada 15/12/2024

Na versão 1 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. Veja Figura 6.1.

Figura 6.1: Versão 1.0, interface do software



Fonte: Produzido pelo autor.

6.2 Versão 1.1 – Otimização de Entrada, Validação e Armazenamento Automático de Dados

Versão: 1.1

Componentes: Módulo de cálculo com interface CLI/terminal e biblioteca CGnuplot (Gnuplot)

Desenvolvida na disciplina: LEP01449 - Projeto de Software Aplicado à Engenharia (site).

Registro da versão 1.1: v1.1 Lançada 19/12/2024

A versão 1.1 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.

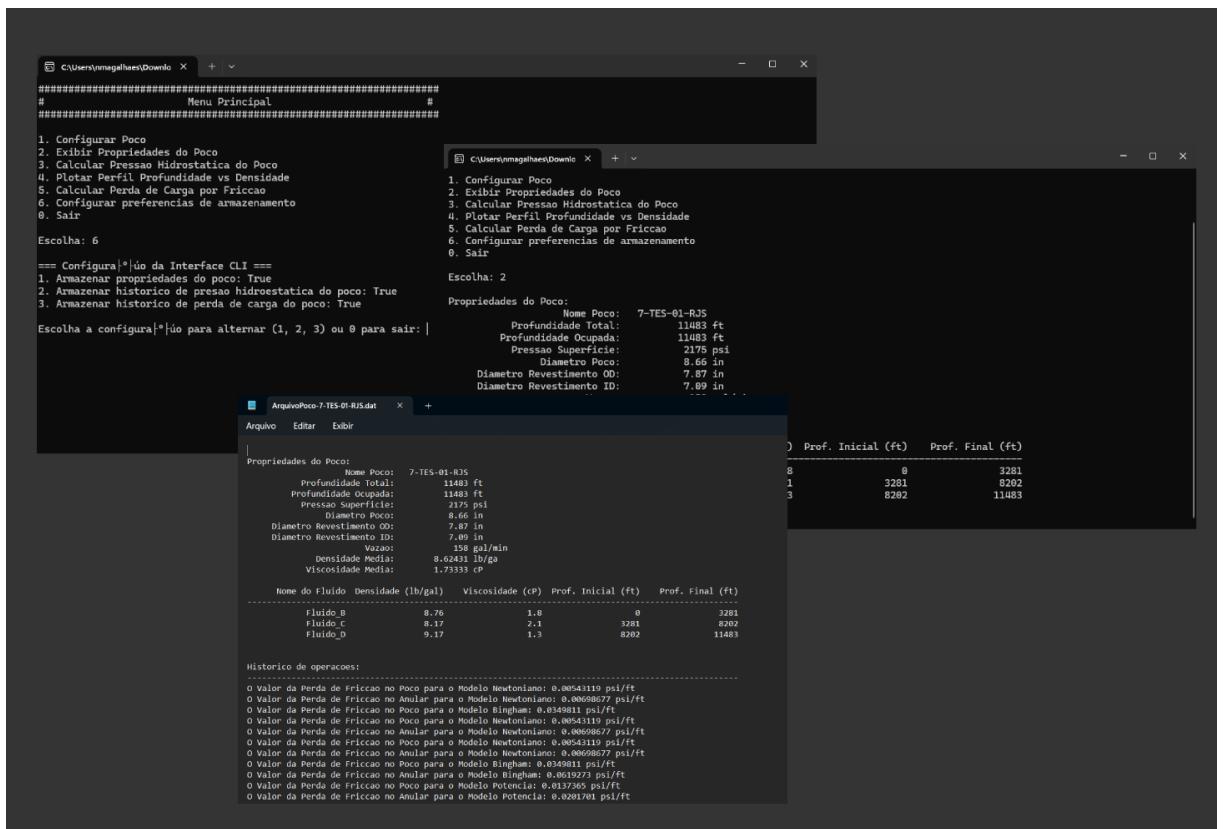
Principais aprimoramentos implementados:

- 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 1.1, interface do software



Produzido pelo autor.

Fonte:

6.3 Versão 2.0 – Interface Gráfica, Amigável e Intuitiva Utilizando o Framework Qt

Versão: 2.0

Componentes: Módulo de cálculo com interface gráfica desenvolvida com *Qt Framework* (*Qt Framework*) e visualização de dados usando a *QCustomPlot* (*QCustomPlot*)

Desenvolvida na disciplina: LEP- 01559: Trabalho de Conclusão de Curso I (site).

Registro da versão 2.0: v2.0 Lançada 25/02/2025

A versão 2.0 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.

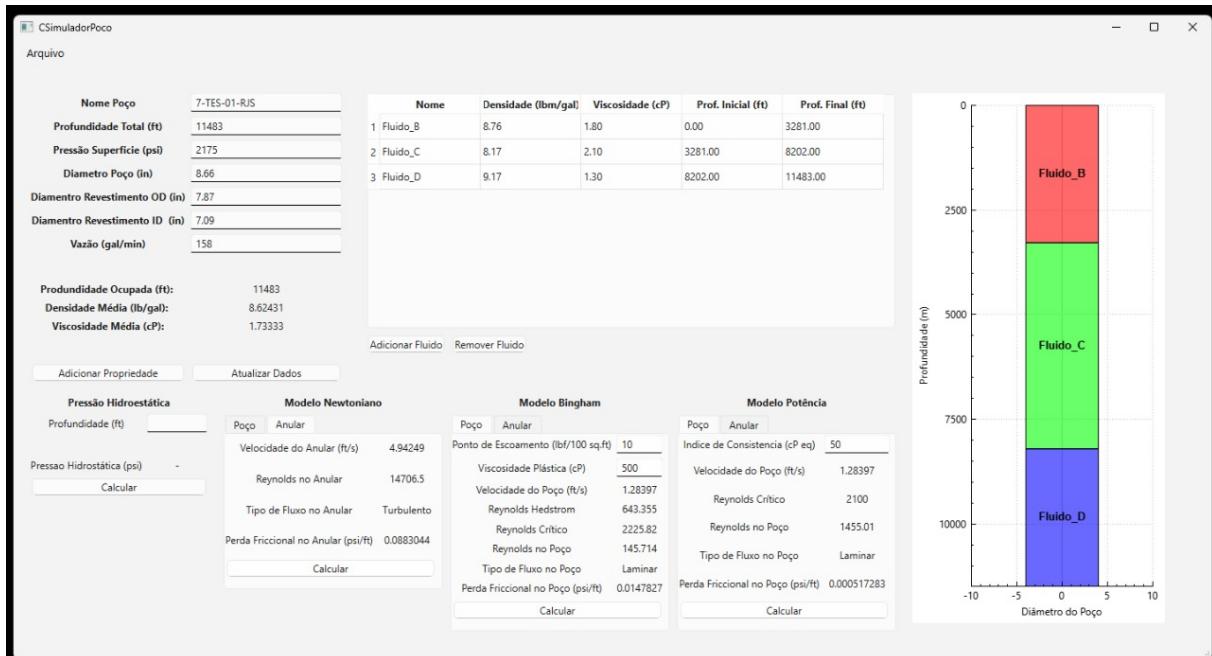
Com a introdução da interface gráfica, o usuário passou a contar com as seguintes facilidades:

- 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.

Veja Figura 6.3.

Figura 6.3: Versão 2.0, interface do software



Fonte: Produzido pelo autor.

6.4 Versão 2.6 – Consolidação Visual, Barra de Tarefas e Automação de Processos

Versão: 2.6

Componentes: Módulo de cálculo com interface gráfica desenvolvida com *Qt Framework* (*Qt Framework*) e visualização de dados usando a *QCustomPlot* (*QCustomPlot*)

Desenvolvida na disciplina: UENF016 - Projeto de Trabalho de Conclusão de Curso

Registro da versão 2.6: v2.6 Lançada 16/04/2025

A versão 2.6 do software manteve todas as funcionalidades introduzidas na versão 2.0, 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 (quando o usuário atualiza um determinado valor imediatamente são atualizados os demais parâmetros) 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 menus com novas funcionalidades organizadas em menus acessíveis, incluindo:

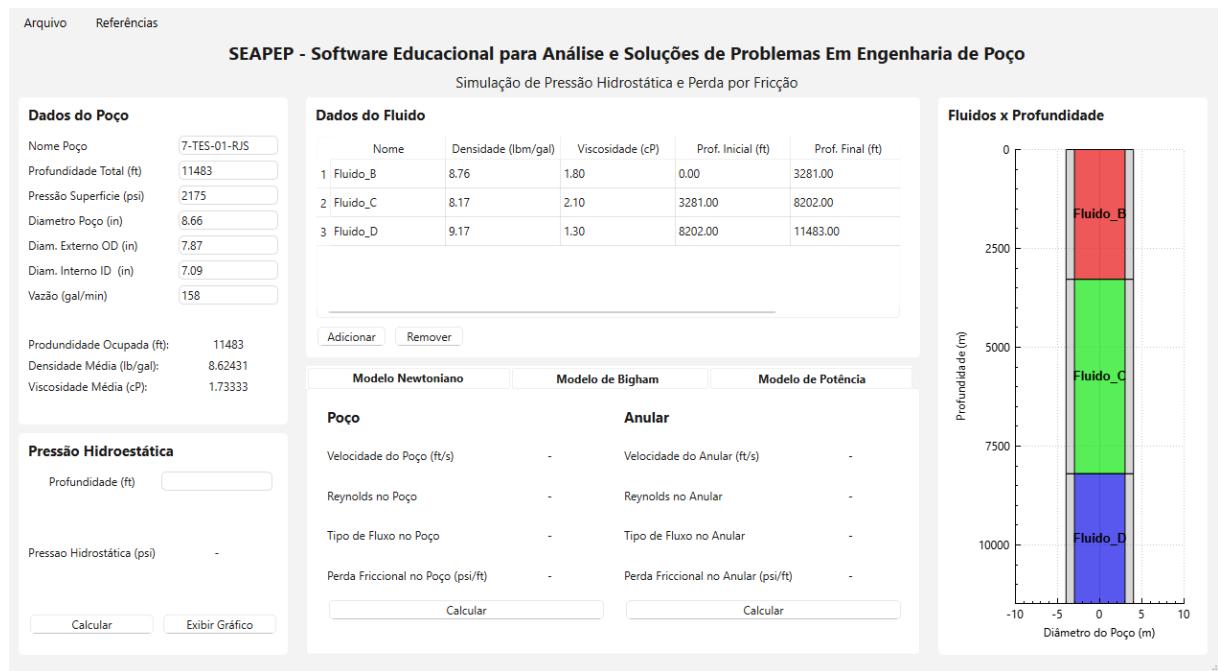
- **Arquivo:** opções para iniciar nova simulação, salvar o projeto atual, importar dados e exportar a interface como imagem;

- **Referências:** acesso ao manual do usuário e à documentação dos modelos reológicos implementados; Permite acessar todos os manuais em pdf.
- **Ajuda:** Acesso ao manual do usuário (como instalar e usar o software); Acesso ao manual técnico científico (modelos reológicos implementados); Sobre o software;
- **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 2.6 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 2.6, interface do software



Fonte: Produzido pelo autor.

6.5 Versão 3.0 – Navegação por Módulos e Simulação Mecânica de Variação de Comprimento (ΔL)

Versão: 3.0

Componentes: Módulo de cálculo com interface gráfica desenvolvida com *Qt Framework* (*Qt Framework*) e visualização de dados usando a *QCustomPlot* (*QCustomPlot*)

Desenvolvida na disciplina: UENF016 - Projeto de Trabalho de Conclusão de Curso

Registro da versão 3.0: v3.0 Lançada 28 de Maio de 2025

A versão 3.0 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 2.6, 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 3.0: 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*.

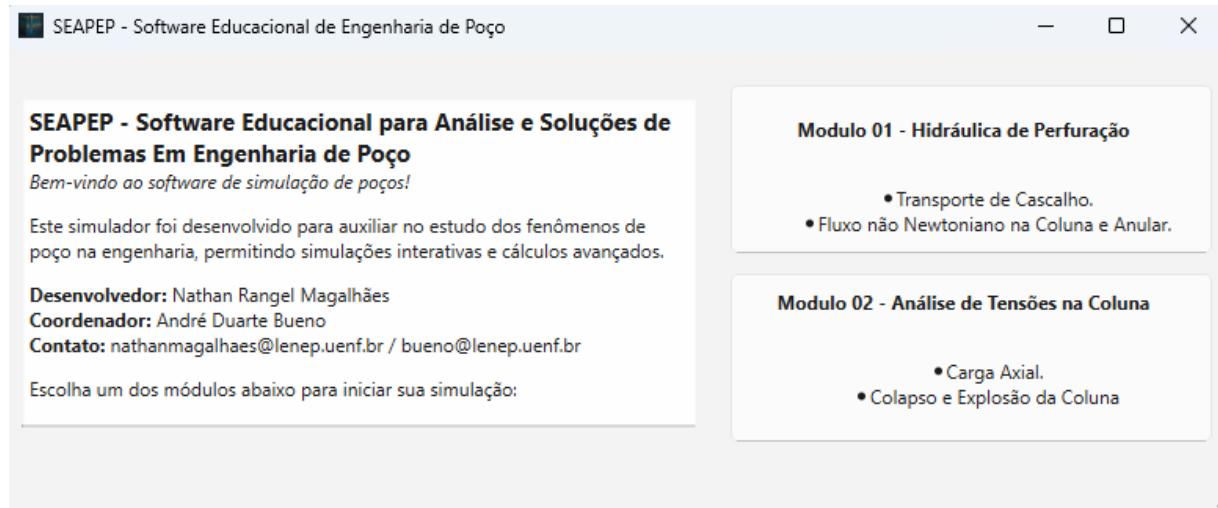
Além disso, o poço continua sendo visualizado graficamente conforme a profundidade, agora com suporte ao novo conjunto de propriedades exigidas pela análise mecânica.

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.

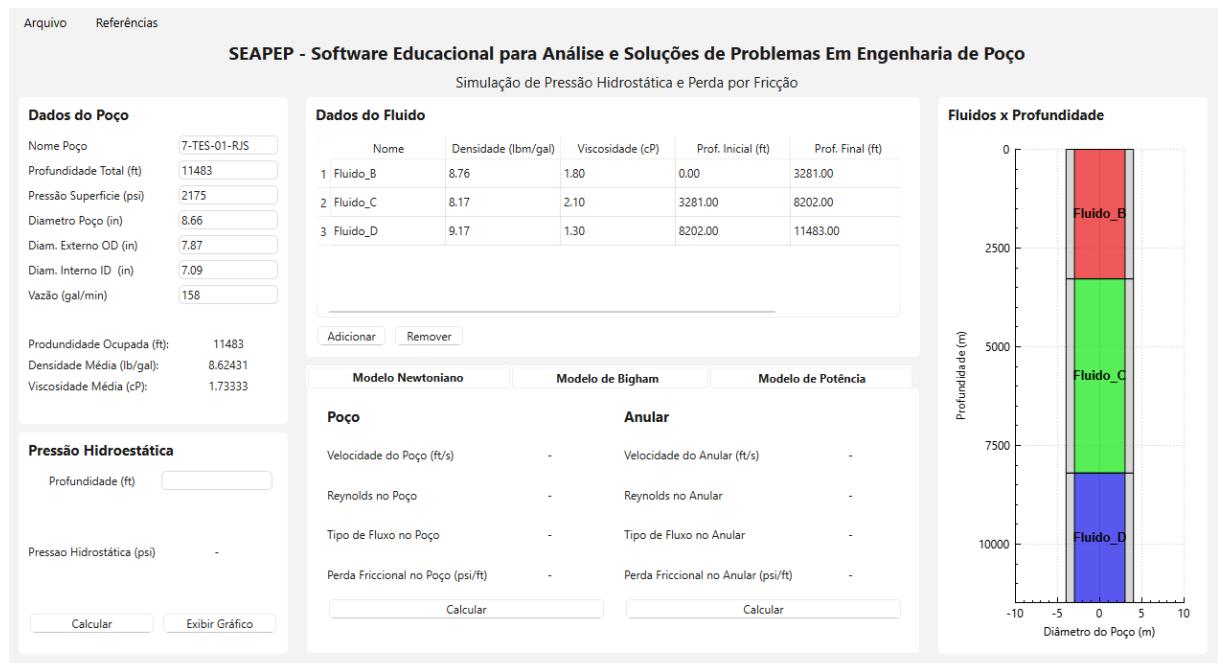
Com essa versão, o sistema passa a incorporar, além da análise hidráulica, uma abordagem mecânica de simulação, ampliando significativamente seu escopo didático e técnico na área de Engenharia de Poço.

Figura 6.5: Versão 3.0, Menu de interface do software



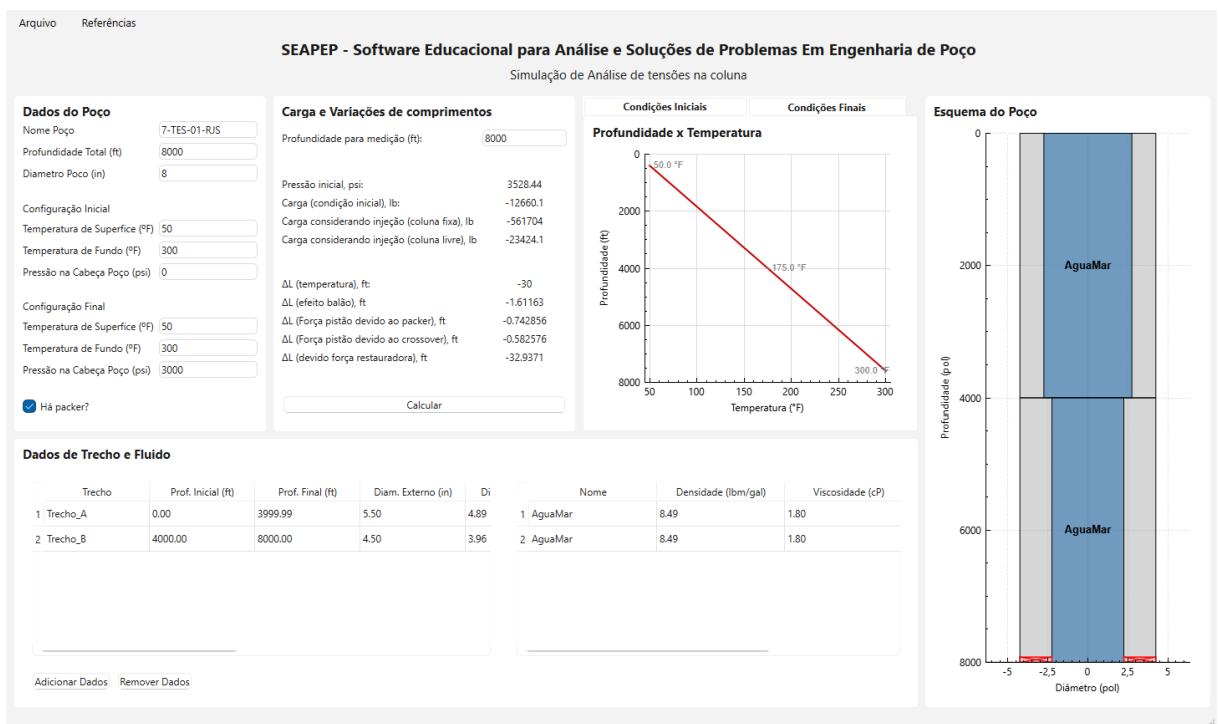
Fonte: Produzido pelo autor.

Figura 6.6: Versão 3.0, interface do módulo 01 do software



Fonte: Produzido pelo autor.

Figura 6.7: Versão 3.0, interface do módulo 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, double n);
49     double DeterminarReynoldsPoco();
50     double DeterminarReynoldsPoco(double viscosidade);
51     double DeterminarReynoldsAnular();
52     double DeterminarReynoldsAnular(double viscosidade);
53     double DeterminarVelocidadeMediaPoco();
54     double DeterminarVelocidadeMediaAnular();
55
56     // M todos puros (devem ser implementados pelas classes filhas
57     )
58     virtual std::string DeterminarFluxoPoco() = 0;
59     virtual std::string DeterminarFluxoAnular() = 0;
60     virtual double CalcularPerdaPorFriccaoPoco() = 0;
61     virtual double CalcularPerdaPorFriccaoAnular() = 0;
62 };
63 #endif // CMODELOREOLOGICO_H

```

Fonte: produzido pelo autor.

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

Listing 7.3: Arquivo de implementação da classe CModeloReologico

```

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

```



```

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

```

Fonte: produzido pelo autor.

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

Listing 7.4: Arquivo de implementação da classe CModeloReologico

```

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

```

```

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

```

```

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

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

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

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

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

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

```

Fonte: produzido pelo autor.

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

Listing 7.5: Arquivo de implementação da classe CModeloReológico

```

1 #include "CSimuladorPerdaTubulacao.h"
2 #include "ui_CSimuladorPerdaTubulacao.h"
3 #include "CJanelaAdicionarFluido.h"
4 #include "CJanelaAdicionarTrechoTubulacao.h"
5 #include "CJanelaSobreSoftware.h"
6
7 #include <iostream> // para std::cerr e std::endl
8 #include <fstream> // para std::ifstream
9 #include <sstream>
10 #include <QMessageBox>
11 #include <QScreen>
12 #include <QFileDialog>
13
14 CSimuladorPerdaTubulacao::CSimuladorPerdaTubulacao(QWidget *parent)
15     : QMainWindow(parent)
16     , ui(new Ui::CSimuladorPerdaTubulacao)
17 {
18     ui->setupUi(this);
19
20     // ativa edicao de celulas da tabela com clique duplo ou tecla
21     // Enter
22     ui->tblFluidos->setEditTriggers(QAbstractItemView::
23         DoubleClicked | QAbstractItemView::EditKeyPressed);
24     ui->tblTrechos->setEditTriggers(QAbstractItemView::
25         DoubleClicked | QAbstractItemView::EditKeyPressed);
26
27     // Sinal para alterar os valores das caixas
28     QObject::connect(ui->editNomePoco, &QLineEdit::editingFinished,
29         this, &CSimuladorPerdaTubulacao::EditarDadosPoco);
30     QObject::connect(ui->editProfundidadeTotal, &QLineEdit::
31         editingFinished, this, &CSimuladorPerdaTubulacao::
32         EditarDadosPoco);
33     QObject::connect(ui->editPressaoSupInicial, &QLineEdit::
34         editingFinished, this, &CSimuladorPerdaTubulacao::
35         EditarDadosPoco);
36     QObject::connect(ui->editPressaoSupFinal, &QLineEdit::
37         editingFinished, this, &CSimuladorPerdaTubulacao::
38         EditarDadosPoco);
39     QObject::connect(ui->editTemperaturaSuperiorInicial, &QLineEdit
40         ::editingFinished, this, &CSimuladorPerdaTubulacao::
41         EditarDadosPoco);

```

```

31     QObject::connect(ui->editTemperaturaFundoInicial, &QLineEdit::  
32         editingFinished, this, &CSimuladorPerdaTubulacao::  
33             EditarDadosPoco);  
34     QObject::connect(ui->editTemperaturaSuperiorFinal, &QLineEdit::  
35         editingFinished, this, &CSimuladorPerdaTubulacao::  
36             EditarDadosPoco);  
37  
38     // iniciar com botões desativado  
39     ui->btnAdicionarTrecho->setEnabled(false);  
40     ui->btnRemoverTrecho->setEnabled(false);  
41     ui->btnCalcularVariacoes->setEnabled(false);  
42  
43     QObject::connect(ui->tblFluidos, &QTableWidget::cellChanged,  
44         this, &CSimuladorPerdaTubulacao::EditarLinhaTabela);  
45     QObject::connect(ui->tblTrechos, &QTableWidget::cellChanged,  
46         this, &CSimuladorPerdaTubulacao::EditarLinhaTabela);  
47  
48     //abrir janela no meio do monitor  
49     QScreen *screen = QGuiApplication::primaryScreen();  
50     QRect screenGeometry = screen->geometry();  
51  
52     int x = (screenGeometry.width() - this->width()) / 2;  
53     int y = (screenGeometry.height() - this->height()) / 2;  
54  
55     this->move(x, y);  
56 }  
57  
58 CSimuladorPerdaTubulacao::~CSimuladorPerdaTubulacao()  
59 {  
60     delete ui;  
61 }

```

```

62
63 void CSimuladorPerdaTubulacao::EditarDadosPoco() {
64     QString nome = ui->editNomePoco->text();
65     bool ok1, ok2, ok3, ok4, ok5, ok6, ok7, ok8;
66     double profund = ui->editProfundidadeTotal->text().toDouble(&
67         ok1);
68     double diamPoco = ui->editDiametroPoco->text().toDouble(&ok2);
69     double pressao = ui->editPressaoSupInicial->text().toDouble(&
70         ok3);
71     double pressaoFim = ui->editPressaoSupFinal->text().toDouble(&
72         ok4);
73     double temperaturaSuperiorInicial = ui->
74         editTemperaturaSuperiorInicial->text().toDouble(&ok5);
75     double temperaturaFundoInicial = ui->
76         editTemperaturaFundoInicial->text().toDouble(&ok6);
77     double temperaturaSuperiorFinal = ui->
78         editTemperaturaSuperiorFinal->text().toDouble(&ok7);
79     double temperaturaFundoFinal = ui->editTemperaturaFundoFinal->
80         text().toDouble(&ok8);
81     bool haPacker = ui->checkBoxPacker->isChecked();
82
83     if (!nome.isEmpty() && ok1 && ok2 && ok3 && ok4 && ok5 && ok6
84         && ok7 && ok8) {
85         if (!poco) {
86             // Cria o p o o
87
88             poco = std::make_unique<CObjetoPoco>(
89                 CObjetoPoco::CriarParaModulo02(nome.toStdString(),
90                     profund, diamPoco, pressao, pressaoFim,
91                     temperaturaSuperiorInicial,
92                     temperaturaFundoInicial,
93                     temperaturaSuperiorFinal, temperaturaFundoFinal,
94                     haPacker)
95             );
96
97             ui->btnAdicionarTrecho->setEnabled(true);
98             ui->btnRemoverTrecho->setEnabled(true);
99             ui->btnCalcularVariacoes->setEnabled(true);
100
101             ui->statusbar->showMessage("Po o criado com Sucesso!")
102                 ;
103         } else {

```

```

90         // Atualiza dados do po o j existente
91         poco->NomePoco(nome.toStdString());
92         poco->ProfundidadeTotal(profund);
93         poco->PressaoSuperficie(pressao);
94         poco->PressaoSuperficieFim(pressaoFim);
95         poco->TemperaturaTopoInicial(temperaturaSuperiorInicial
96             );
97         poco->TemperaturaFundoInicial(temperaturaFundoInicial);
98         poco->TemperaturaTopoFinal(temperaturaSuperiorFinal);
99         poco->TemperaturaFundoFinal(temperaturaFundoFinal);
100        poco->Packer(haPacker);
101        ui->statusbar->showMessage("Dados de Poco Atualizado com Sucesso !");
102    }
103
104    AtualizarDados(); // Atualiza os dados calculados e a
105    interface
106}
107void CSimuladorPerdaTubulacao::on_btnAdicionarPropriedades_clicked()
108{
109    std::string nome;
110    double profundidade,diamPoco, pressaoSup, pressaoSupFim,
111        temperaturaSuperiorInicial, temperaturaFundoInicial,
112        temperaturaSuperiorFinal, temperaturaFundoFinal;
113    bool haPacker;
114
115    QString text;
116
117    text = ui->editNomePoco->text();
118    nome = text.toStdString();
119    text = ui->editPressaoSupInicial->text();
120    pressaoSup = text.toDouble();
121    text = ui->editPressaoSupFinal->text();
122    pressaoSupFim = text.toDouble();
123    text = ui->editProfundidadeTotal->text();
124    profundidade = text.toDouble();
125    text = ui->editDiametroPoco->text();
126    diamPoco = text.toDouble();
127    text = ui->editTemperaturaSuperiorInicial->text();

```

```

126     temperaturaSuperiorInicial = text.toDouble();
127     text = ui->editTemperaturaFundoInicial->text();
128     temperaturaFundoInicial = text.toDouble();
129     text = ui->editTemperaturaSuperiorFinal->text();
130     temperaturaSuperiorFinal = text.toDouble();
131     text = ui->editTemperaturaFundoFinal->text();
132     temperaturaFundoFinal = text.toDouble();
133     haPacker = ui->checkBoxPacker->isChecked();
134
135
136     if (poco) {
137         QMessageBox::StandardButton resposta = QMessageBox::
138             question(
139                 this,
140                 "",
141                 "Ao confirmar, todos os fluidos ser\u00e3o deletados! Tem
142                 certeza?",
143                 QMessageBox::Yes | QMessageBox::No
144             );
145
146     if (resposta == QMessageBox::Yes) {
147         poco = std::make_unique<CObjetoPoco>(
148             CObjetoPoco::CriarParaModulo02(nome, profundidade,
149             diamPoco, pressaoSup, pressaoSupFim,
150             temperaturaSuperiorInicial,
151             temperaturaFundoInicial,
152             temperaturaSuperiorFinal, temperaturaFundoFinal,
153             haPacker)
154         );
155     }
156 }
```

```

157     makePlotTemperatura(temperaturaSuperiorInicial ,
158         temperaturaFundoInicial , profundidade , ui ->
159         customPlotTemperaturaInicial);
160     makePlotTemperatura(temperaturaSuperiorFinal ,
161         temperaturaFundoFinal , profundidade , ui ->
162         customPlotTemperaturaFinal);
163     makePlotPoco ();
164 }
165
166 void CSimuladorPerdaTubulacao::AtualizarDados()
167 {
168     ui ->tblFluidos ->blockSignals(true);
169     ui ->tblTrechos ->blockSignals(true);
170
171     if (poco){
172         // Atualiza os valores dos QLineEdits com os dados do
173         // objeto poco
174         ui ->editNomePoco ->setText(QString::fromStdString(poco->
175             NomePoco()));           // Profundidade total do po o
176         ui ->editProfundidadeTotal ->setText(QString::number(poco->
177             ProfundidadeTotal()));           // Profundidade total do
178             po o
179         ui ->editDiametroPoco ->setText(QString::number(poco->
180             DiametroPoco()));
181         ui ->editPressaoSupInicial ->setText(QString::number(poco->
182             PressaoSuperficie())); // Profundidade ocupada
183         ui ->editPressaoSupFinal ->setText(QString::number(poco->
184             PressaoSuperficieFim()));
185         ui ->editTemperaturaSuperiorInicial ->setText(QString::number(
186             (poco->TemperaturaTopoInicial())));
187             // Di metro do po o
188         ui ->editTemperaturaFundoInicial ->setText(QString::number(
189             poco->TemperaturaFundoInicial()));           // Di metro
190             externo do revestimento (ID)
191         ui ->editTemperaturaSuperiorFinal ->setText(QString::number(
192             poco->TemperaturaTopoFinal()));
193             // Di metro interno do revestimento (ID)
194         ui ->editTemperaturaFundoFinal ->setText(QString::number(poco
195             ->TemperaturaFundoFinal()));
196             // Vaz o do fluido no po o
197         if (poco->Packer() == true) {
198             ui ->checkBoxPacker ->setChecked(true);

```

```

180     } else {
181         ui->checkBoxPacker->setChecked(false);
182     }
183
184
185
186     // Atualizar QTableWidget com os dados dos trechos
187     ui->tblTrechos->setRowCount(static_cast<int>(poco->Trechos
188                                     ().size()));
189     ui->tblFluidos->setRowCount(static_cast<int>(poco->Trechos
190                                     ().size()));
191     int row = 0;
192     for (const auto& trecho : poco->Trechos()) {
193
194         ui->tblTrechos->setItem(row, 0, new QTableWidgetItem(
195             QString::fromStdString(trecho->Nome())));
196         ui->tblTrechos->setItem(row, 1, new QTableWidgetItem(
197             QString::number(trecho->ProfundidadeInicial(), 'f',
198                             2)));
199         ui->tblTrechos->setItem(row, 2, new QTableWidgetItem(
200             QString::number(trecho->ProfundidadeFinal(), 'f',
201                             2)));
202         ui->tblTrechos->setItem(row, 3, new QTableWidgetItem(
203             QString::number(trecho->DiametroExterno(), 'f',
204                             2)));
205         ui->tblTrechos->setItem(row, 4, new QTableWidgetItem(
206             QString::number(trecho->DiametroInterno(), 'f',
207                             2)));
208         ui->tblTrechos->setItem(row, 5, new QTableWidgetItem(
209             QString::number(trecho->CoeficientePoisson(), 'f',
210                             2)));
211         ui->tblTrechos->setItem(row, 6, new QTableWidgetItem(
212             QString::number(trecho->CoeficienteExpancaoTermica(),
213                             'f', 8)));
214         ui->tblTrechos->setItem(row, 7, new QTableWidgetItem(
215             QString::number(trecho->ModuloElasticidade(), 'f',
216                             2)));
217         ui->tblTrechos->setItem(row, 8, new QTableWidgetItem(
218             QString::number(trecho->PesoUnidade(), 'f',
219                             2)));
220
221         ui->tblFluidos->setItem(row, 0, new QTableWidgetItem(
222             QString::fromStdString(trecho->Fluido()->Nome())));

```

```

203     ui ->tblFluidos ->setItem(row, 1, new QTableWidgetItem(
204         QString::number(trecho->Fluido()->Densidade(), 'f',
205         2)));
206     ui ->tblFluidos ->setItem(row, 2, new QTableWidgetItem(
207         QString::number(trecho->Fluido()->Viscosidade(), 'f',
208         2)));
209
210     ++row;
211 }
212
213 makePlotTemperatura(poco->TemperaturaTopoInicial(), poco->
214 TemperaturaFundoInicial(), poco->ProfundidadeTotal(), ui->
215 customPlotTemperaturaInicial);
216 makePlotTemperatura(poco->TemperaturaTopoFinal(), poco->
217 TemperaturaFundoFinal(), poco->ProfundidadeTotal(), ui->
218 customPlotTemperaturaFinal);
219 makePlotPoco();
220
221
222 ui ->tblFluidos ->blockSignals(false);
223 ui ->tblTreichos ->blockSignals(false);
224 }
225
226 void CSimuladorPerdaTubulacao::on_btnAdicionarTrecho_clicked()
227 {
228     ui ->tblTreichos ->setEditTriggers(QAbstractItemView::
229         NoEditTriggers);
230     ui ->tblFluidos ->setEditTriggers(QAbstractItemView::
231         NoEditTriggers);
232
233     if (!poco) {
234         QMessageBox::warning(this, "Erro", "As propriedades do ponto
235             precisam estar preenchidas!");
236     }
237
238     else{
239         CJanelaAdicionarTrechoTubulacao JanelaTrecho;
240         JanelaTrecho.exec();
241
242         if (JanelaTrecho.Trecho() != "" &&
243             JanelaTrecho.ProfundidadeInicial() != "" &&
244             JanelaTrecho.ProfundidadeFinal() != "" &&
245             JanelaTrecho.DiametroExterno() != "") {
246             poco->adicionarTrecho(JanelaTrecho.Trecho(),
247             JanelaTrecho.ProfundidadeInicial(),
248             JanelaTrecho.ProfundidadeFinal(),
249             JanelaTrecho.DiametroExterno());
250             ui ->tblTreichos ->appendRow(ui ->tblTreichos ->create
251                 Model());
252             ui ->tblTreichos ->selectRow(ui ->tblTreichos ->currentRow());
253         }
254     }
255 }

```

```

234     JanelaTrecho.DiametroInterno() != "" &&
235     JanelaTrecho.CoefficientePoisson() != "" &&
236     JanelaTrecho.CoefficienteExpansaoTermica() != "" &&
237     JanelaTrecho.ModuloElasticidade() != "" &&
238     JanelaTrecho.PesoUnidade() != "" &&
239     JanelaTrecho.NomeFluido() != "" &&
240     JanelaTrecho.Densidade() != "" &&
241     JanelaTrecho.Viscosidade() != ""){

242
243
244     int numLinhas = ui->tblTrechos->rowCount();
245
246     ui->tblTrechos->insertRow(numLinhas);
247     ui->tblTrechos->setItem(numLinhas, 0, new
248         QTableWidgetItem(JanelaTrecho.Trecho()));
249     ui->tblTrechos->setItem(numLinhas, 1, new
250         QTableWidgetItem(JanelaTrecho.ProfundidadeInicial()))
251         );
252     ui->tblTrechos->setItem(numLinhas, 2, new
253         QTableWidgetItem(JanelaTrecho.ProfundidadeFinal()));
254     ui->tblTrechos->setItem(numLinhas, 3, new
255         QTableWidgetItem(JanelaTrecho.DiametroExterno()));
256     ui->tblTrechos->setItem(numLinhas, 4, new
257         QTableWidgetItem(JanelaTrecho.DiametroInterno()));
258     ui->tblTrechos->setItem(numLinhas, 5, new
259         QTableWidgetItem(JanelaTrecho.CoefficientePoisson()))
260         ;
261     ui->tblTrechos->setItem(numLinhas, 6, new
262         QTableWidgetItem(JanelaTrecho.
263             CoeficienteExpansaoTermica()));
264     ui->tblTrechos->setItem(numLinhas, 7, new
265         QTableWidgetItem(JanelaTrecho.ModuloElasticidade()))
266         ;
267     ui->tblTrechos->setItem(numLinhas, 8, new
268         QTableWidgetItem(JanelaTrecho.PesoUnidade()));

269
270     ui->tblFluidos->insertRow(numLinhas);
271     ui->tblFluidos->setItem(numLinhas, 0, new
272         QTableWidgetItem(JanelaTrecho.NomeFluido()));
273     ui->tblFluidos->setItem(numLinhas, 1, new
274         QTableWidgetItem(JanelaTrecho.Densidade()));
275     ui->tblFluidos->setItem(numLinhas, 2, new

```

```

        QTableWidgetItem(JanelaTrecho.Viscosidade()));

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

280     std::string nome = JanelaTrecho.NomeFluido().
281         toStdString();
282     double densidade = JanelaTrecho.Densidade().toDouble();
283     double viscosidade = JanelaTrecho.Viscosidade().
284         toDouble();

285     auto fluido = std::make_unique<CFluido>(nome, densidade,
286         viscosidade);
287     auto trechoPoco = std::make_unique<CTrechoPoco>(
288         NomeTrecho, profundInicial, profundFinal, std::move(
289             fluido), diametroExterno, diametroInterno,
290         coeficientePoisson, coeficienteExpansaoTermica,
291         moduloElasticidade, pesoUnidade);
292     poco->AdicionarTrechoPoco(std::move(trechoPoco));

293     AtualizarDados();
294 }

295 }

296 }

297 }

298 }

299 }

300 }

301 }

302 }

303 }

304 }

305 }

306 }

307 }

308 }

309 }

310 }

311 }

312 }

313 }

314 }

315 }

316 }

317 }

318 }

319 }

320 }

321 }

322 }

323 }

324 }

325 }

326 }

327 }

328 }

329 }

330 }

331 }

332 }

333 }

334 }

335 }

336 }

337 }

338 }

339 }

340 }

341 }

342 }

343 }

344 }

345 }

346 }

347 }

348 }

349 }

350 }

351 }

352 }

353 }

354 }

355 }

356 }

357 }

358 }

359 }

360 }

361 }

362 }

363 }

364 }

365 }

366 }

367 }

368 }

369 }

370 }

371 }

372 }

373 }

374 }

375 }

376 }

377 }

378 }

379 }

380 }

381 }

382 }

383 }

384 }

385 }

386 }

387 }

388 }

389 }

390 }

391 }

392 }

393 }

394 }

395 }

396 }

397 }

398 }

399 }

400 }

401 }

402 }

403 }

404 }

405 }

406 }

407 }

408 }

409 }

410 }

411 }

412 }

413 }

414 }

415 }

416 }

417 }

418 }

419 }

420 }

421 }

422 }

423 }

424 }

425 }

426 }

427 }

428 }

429 }

430 }

431 }

432 }

433 }

434 }

435 }

436 }

437 }

438 }

439 }

440 }

441 }

442 }

443 }

444 }

445 }

446 }

447 }

448 }

449 }

450 }

451 }

452 }

453 }

454 }

455 }

456 }

457 }

458 }

459 }

460 }

461 }

462 }

463 }

464 }

465 }

466 }

467 }

468 }

469 }

470 }

471 }

472 }

473 }

474 }

475 }

476 }

477 }

478 }

479 }

480 }

481 }

482 }

483 }

484 }

485 }

486 }

487 }

488 }

489 }

490 }

491 }

492 }

493 }

494 }

495 }

496 }

497 }

498 }

499 }

500 }

501 }

502 }

503 }

504 }

505 }

506 }

507 }

508 }

509 }

510 }

511 }

512 }

513 }

514 }

515 }

516 }

517 }

518 }

519 }

520 }

521 }

522 }

523 }

524 }

525 }

526 }

527 }

528 }

529 }

530 }

531 }

532 }

533 }

534 }

535 }

536 }

537 }

538 }

539 }

540 }

541 }

542 }

543 }

544 }

545 }

546 }

547 }

548 }

549 }

550 }

551 }

552 }

553 }

554 }

555 }

556 }

557 }

558 }

559 }

560 }

561 }

562 }

563 }

564 }

565 }

566 }

567 }

568 }

569 }

570 }

571 }

572 }

573 }

574 }

575 }

576 }

577 }

578 }

579 }

580 }

581 }

582 }

583 }

584 }

585 }

586 }

587 }

588 }

589 }

590 }

591 }

592 }

593 }

594 }

595 }

596 }

597 }

598 }

599 }

599 }
```

```

TempInicial, double TempFinal, double profundidade, QCustomPlot*
plot)

286 {
287     // Limpa graficos anteriores
288     plot->clearItems();
289     plot->clearPlottables();
290
291     // Configura eixos
292     plot->xAxis->setLabel("Temperatura ( F )");
293     plot->yAxis->setLabel("Profundidade (ft)");
294     plot->yAxis->setRangeReversed(true); // profundidade cresce pra
295         baixo
296
297     // Adiciona "respiro" nos extremos
298     double margemProfundidade = profundidade * 0.05; // 5% de
299         margem visual
300
301     // Define pontos
302     double tempMeio = (TempInicial + TempFinal) / 2.0;
303     QVector<double> temperaturas = {TempInicial, tempMeio,
304         TempFinal};
305     QVector<double> profundidades = {0.0 + margemProfundidade,
306         profundidade / 2.0, profundidade - margemProfundidade};
307
308     // Ajuste de range
309     double tempMin = *std::min_element(temperaturas.begin(),
310         temperaturas.end());
311     double tempMax = *std::max_element(temperaturas.begin(),
312         temperaturas.end());
313
314     plot->xAxis->setRange(tempMin - 5, tempMax + 5);
315     plot->yAxis->setRange(0.0, profundidade); // Mantem 0 a
316         profundidade total, o respiro e visual apenas
317
318     // Grid leve
319     plot->xAxis->grid()->setVisible(true);
320     plot->yAxis->grid()->setVisible(true);
321     plot->xAxis->grid()->setPen(QPen(QColor(220, 220, 220)));
322     plot->yAxis->grid()->setPen(QPen(QColor(220, 220, 220)));
323
324     // Linha do perfil
325     QCPGraph *perfilTemp = plot->addGraph();

```

```

319     perfilTemp->setData(temperaturas, profundidades);
320     perfilTemp->setPen(QPen(QColor(200, 0, 0), 2));
321
322     // Marcar e rotular os 3 pontos (topo, meio e fundo)
323     for (int i = 0; i < temperaturas.size(); ++i) {
324         QCPIItemEllipse *ponto = new QCPIItemEllipse(plot);
325         ponto->topLeft->setCoords(temperaturas[i] - 2,
326                                     profundidades[i] - 20);
327         ponto->bottomRight->setCoords(temperaturas[i] + 2,
328                                         profundidades[i] + 20);
329         ponto->setPen(Qt::NoPen);
330         ponto->setBrush(QBrush(Qt::darkGray));
331
332         QCPIItemText *rotulo = new QCPIItemText(plot);
333         rotulo->position->setCoords(temperaturas[i] + 4,
334                                     profundidades[i]);
335         rotulo->setText(QString::number(temperaturas[i], 'f', 1) +
336                         " ° F ");
337         rotulo->setFont(QFont("Arial", 8));
338         rotulo->setColor(Qt::darkGray);
339         // Ajusta alinhamento do rotulo do ultimo ponto para dentro
340         // do grafico
341         if (i == temperaturas.size() - 1)
342             rotulo->setPositionAlignment(Qt::AlignRight | Qt::
343                                           AlignVCenter); // alinha para a esquerda do ponto
344             final
345         else
346             rotulo->setPositionAlignment(Qt::AlignLeft | Qt::
347                                           AlignVCenter);
347     }
348
349     // Limpeza estatica
350     plot->legend->setVisible(false);
351     plot->setBackground(Qt::white);
352     plot->xAxis->setBasePen(QPen(Qt::black));
353     plot->yAxis->setBasePen(QPen(Qt::black));
354     plot->xAxis->setTickPen(QPen(Qt::black));
355     plot->yAxis->setTickPen(QPen(Qt::black));
356     plot->xAxis->setTickLabelColor(Qt::black);
357     plot->yAxis->setTickLabelColor(Qt::black);
358
359     // Replotar

```



```

391         // se clicou so na tabela de fluido, avisa que deve usar a
392         // outra
393         QMessageBox::warning(this, "Aviso", "A\u00e9 remo o \u00f3deve ser\u00e1
394         // feita\u00a5pela\u00a5tabela\u00a5de\u00a5trechos.");
395     } else {
396         // nenhuma linha foi selecionada
397         QMessageBox::warning(this, "Erro", "Nenhuma\u00a5linha\u00a5foiu
398         // selecionada.");
399     }
400 }
401
402 void CSimuladorPerdaTubulacao::makePlotPoco()
403 {
404     ui->customPlotPoco->clearItems();
405     ui->customPlotPoco->xAxis->setLabel("Di\u00e9metro(pol)");
406     ui->customPlotPoco->yAxis->setLabel("Profundidade(pol)");
407     ui->customPlotPoco->yAxis->setRangeReversed(true);
408
409     if (!poco || poco->Trechos().empty())
410         return;
411
412     // 1. Profundidade m\u00f3xima e maior di\u00e9metro externo
413     double profundidadeMaxima = 0.0;
414     double maiorDiametroExterno = 0.0;
415     for (const auto& trecho : poco->Trechos()) {
416         profundidadeMaxima = std::max(profundidadeMaxima, trecho->
417             ProfundidadeFinal());
418         maiorDiametroExterno = std::max(maiorDiametroExterno,
419             trecho->DiametroExterno());
420     }
421
422     // 2. Buraco = maior di\u00e9metro externo + 3 polegadas
423     double diametroBuraco = maiorDiametroExterno + 3.0;
424
425     // 3. Ajuste visual no eixo X: 1.5 vezes o furo
426     double larguraGrafico = diametroBuraco * 1.5;
427     ui->customPlotPoco->xAxis->setRange(-larguraGrafico / 2.0,
428         larguraGrafico / 2.0);
429     ui->customPlotPoco->yAxis->setRange(0, profundidadeMaxima);
430
431     // 4. Cores dos fluidos

```

```

427     QMap<QString, QColor> mapaCores;
428     QVector<QColor> coresDisponiveis = {
429         QColor(70, 130, 180, 180), QColor(255, 0, 0, 150),
430         QColor(0, 255, 0, 150), QColor(0, 0, 255, 150),
431         QColor(255, 165, 0, 150), QColor(128, 0, 128, 150)
432     };
433     int corIndex = 0;
434
435     // 5. Desenho dos trechos
436     for (const auto& trecho : poco->Trechos()) {
437         double z1 = trecho->ProfundidadeInicial();
438         double z2 = trecho->ProfundidadeFinal();
439         double dExt = trecho->DiametroExterno();
440         QString nomeFluido = QString::fromStdString(trecho->Fluido
441             ()->Nome());
442
443         if (!mapaCores.contains(nomeFluido)) {
444             mapaCores[nomeFluido] = coresDisponiveis[corIndex++ %
445                 coresDisponiveis.size()];
446         }
447         QColor corFluido = mapaCores[nomeFluido];
448
449         // === Retângulo cinza (buraco do poço) ===
450         QCPIItemRect *rectBuraco = new QCPIItemRect(ui->
451             customPlotPoco);
452         rectBuraco->topLeft->setCoords(-diametroBuraco / 2.0, z1);
453         rectBuraco->bottomRight->setCoords(diametroBuraco / 2.0, z2
454             );
455         rectBuraco->setPen(QPen(Qt::black));
456         rectBuraco->setBrush(QBrush(QColor(150, 150, 150, 100)));
457
458         // === Retângulo da seção (fluido) ===
459         QCPIItemRect *rectSecao = new QCPIItemRect(ui->customPlotPoco
460             );
461         rectSecao->topLeft->setCoords(-dExt / 2.0, z1);
462         rectSecao->bottomRight->setCoords(dExt / 2.0, z2);
463         rectSecao->setPen(QPen(Qt::black));
464         rectSecao->setBrush(QBrush(corFluido));
465
466         // === Rótulo ===
467         QCPIItemText *label = new QCPIItemText(ui->customPlotPoco);
468         label->position->setCoords(0, (z1 + z2) / 2.0);

```

```

464     label->setText(nomeFluido);
465     label->setFont(QFont("Arial", 10, QFont::Bold));
466     label->setColor(Qt::black);
467     label->setPositionAlignment(Qt::AlignCenter);
468 }
469 // 6. Desenhar packer se existir
470 bool haPacker = poco->Packer();
471 if (haPacker == true) {
472
473     // Pega a profundidade do ltimo trecho como profundidade do
474     // packer
475     double profundidadePacker = 0.0;
476     if (!poco->Trechos().empty()) {
477         profundidadePacker = poco->Trechos().back()->
478             ProfundidadeFinal();
479     }
480
481     double alturaPacker = std::max(profundidadeMaxima * 0.01, 12.0)
482         ;
483     double zTop, zBottom;
484
485     // Se o packer estiver exatamente na profundidade mxima ,
486     // desenha ele todo acima
487     if (std::abs(profundidadePacker - profundidadeMaxima) < 1e-3) {
488         zBottom = profundidadePacker;
489         zTop = profundidadePacker - alturaPacker;
490     } else {
491         zTop = profundidadePacker - alturaPacker / 2.0;
492         zBottom = profundidadePacker + alturaPacker / 2.0;
493     }
494
495     // Encontrar o trecho correspondente profundidade do packer
496     double diametroNoPacker = 0.0;
497     for (const auto& trecho : poco->Trechos()) {
498         if (profundidadePacker >= trecho->ProfundidadeInicial() &&
499             profundidadePacker <= trecho->ProfundidadeFinal()) {
500             diametroNoPacker = trecho->DiametroExterno();
501             break;
502         }
503     }
504
505     // Se n o achou trecho correspondente, usa o maior conhecido

```

```

        como fallback

502     if (diametroNoPacker == 0.0)
503         diametroNoPacker = maiorDiametroExterno;

504
505     // Coordenadas horizontais para os quadrados laterais
506     double xEsq1 = -diametroBuraco / 2.0;
507     double xEsq2 = -diametroNoPacker / 2.0;
508     double xDir1 = diametroNoPacker / 2.0;
509     double xDir2 = diametroBuraco / 2.0;

510
511     // === Quadrado esquerdo ===
512     QCPIItemRect* rectPackerEsq = new QCPIItemRect(ui->
513             customPlotPoco);
514     rectPackerEsq->topLeft->setCoords(xEsq1, zTop);
515     rectPackerEsq->bottomRight->setCoords(xEsq2, zBottom);
516     rectPackerEsq->setPen(QPen(Qt::red, 1.5));
517     rectPackerEsq->setBrush(Qt::NoBrush);

518     // === Quadrado direito ===
519     QCPIItemRect* rectPackerDir = new QCPIItemRect(ui->
520             customPlotPoco);
521     rectPackerDir->topLeft->setCoords(xDir1, zTop);
522     rectPackerDir->bottomRight->setCoords(xDir2, zBottom);
523     rectPackerDir->setPen(QPen(Qt::red, 1.5));
524     rectPackerDir->setBrush(Qt::NoBrush);

525     // === X vermelho esquerdo ===
526     QCPIItemLine* linha1Esq = new QCPIItemLine(ui->customPlotPoco
527             );
528     linha1Esq->start->setCoords(xEsq1, zTop);
529     linha1Esq->end->setCoords(xEsq2, zBottom);
530     linha1Esq->setPen(QPen(Qt::red, 1.5));

531     QCPIItemLine* linha2Esq = new QCPIItemLine(ui->customPlotPoco
532             );
533     linha2Esq->start->setCoords(xEsq2, zTop);
534     linha2Esq->end->setCoords(xEsq1, zBottom);
535     linha2Esq->setPen(QPen(Qt::red, 1.5));

536     // === X vermelho direito ===
537     QCPIItemLine* linha1Dir = new QCPIItemLine(ui->customPlotPoco
538             );

```

```

538     linha1Dir->start->setCoords(xDir1, zTop);
539     linha1Dir->end->setCoords(xDir2, zBottom);
540     linha1Dir->setPen(QPen(Qt::red, 1.5));
541
542     QCPItemLine* linha2Dir = new QCPItemLine(ui->customPlotPoco
543         );
544     linha2Dir->start->setCoords(xDir2, zTop);
545     linha2Dir->end->setCoords(xDir1, zBottom);
546     linha2Dir->setPen(QPen(Qt::red, 1.5));
547 }
548 // 7. Linha tracejada indicando profundidade de medí o , se
549 // v lida
550 bool ok = false;
551 double profundidadeMedicao = ui->editProfundidadeMedicao->text
552     () .toDouble(&ok);
553
554 // So desenha se a conversao foi bem-sucedida e o valor for
555 // maior que zero
556 if (ok && profundidadeMedicao > 0.0) {
557     QCPItemLine* linhaMedicao = new QCPItemLine(ui->
558         customPlotPoco);
559     linhaMedicao->start->setCoords(-larguraGrafico / 2.0,
560         profundidadeMedicao);
561     linhaMedicao->end->setCoords(larguraGrafico / 2.0,
562         profundidadeMedicao);
563
564     // Define o estilo como linha tracejada preta
565     QPen pen(Qt::black);
566     pen.setStyle(Qt::DashLine);
567     pen.setWidthF(1.5);
568     linhaMedicao->setPen(pen);
569 }
570
571 ui->customPlotPoco->replot();
572 }
573
574 void CSimuladorPerdaTubulacao::on_btnCalcularVariacoes_clicked()
575 {
576     double pressaoCabeca = ui->editPressaoSupFinal->text() .toDouble
577     ();

```

```

572
573     QString profundidadeStr = ui->editProfundidadeMedicao->text();
574     double profundidade = profundidadeStr.toDouble();
575
576     ui->lbnPressaoHidroestatica->setText(QString::number( (poco->
577         PressaoHidroestaticaNoPonto(profundidade)) ));
578     ui->lbnCargaInicial->setText(QString::number(poco->Carga(
579         profundidade, true)));
580
581     ui->lbnTituloDeltaLTemperatura->setText(QString::number(poco->
582         DeltaLTemperatura(profundidade)));
583     ui->lbnCargaInjecaoColunaFixa->setText(QString::number(poco->
584         CargaInjecao(profundidade)));
585     ui->lbnCargaInjecaoColunaLivre->setText(QString::number(poco->
586         Carga(profundidade, false)));
587     ui->lbnDeltaLPistaoPacker->setText(QString::number(poco->
588         DeltaLPistaoPacker(profundidade)));
589     ui->lbnTituloDeltaLBalao->setText(QString::number(poco->
590         DeltaLEfeitoBalao(profundidade)));
591     ui->lbnDeltaLPistaoCrossover->setText(QString::number(poco->
592         DeltaLPistaoCrossover(profundidade)));
593     ui->lbnDeltaLForcaRestauradora->setText(QString::number(poco->
594         DeltaLForcaRestauradora(profundidade)));
595
596 }
597
598
599 void CSimuladorPerdaTubulacao::on_actionArquivo_Dat_triggered()
600 {
601     // Abre uma janelinha pro usuario escolher o arquivo .dat
602     QString caminhoDoArquivo = QFileDialog::getOpenFileName(
603         this,
604         "Selecione um arquivo",
605         "",
606         "Todos os arquivos (*.*)"
607     );
608
609     // Verifica se o usuario nao escolheu nada
610     if (caminhoDoArquivo.isEmpty())
611         return;
612
613     std::string caminhoDoArquivoStr = caminhoDoArquivo.toStdString()
614         ();

```

```
604     std::ifstream file(caminhoDoArquivoStr);
605
606     if (!file.is_open()) {
607         ui->statusbar->showMessage("Falha ao abrir o arquivo!");
608         return;
609     }
610
611     std::string linha;
612     bool lendoTreichos = false;
613     bool leuPoco = false;
614     bool leuTrecho = false;
615
616     while (std::getline(file, linha)) {
617         // Verifica se chegou na parte de trechos e fluidos
618         if (linha.find("Configuracao dos Fluidos") != std::string::npos) {
619             lendoTreichos = true;
620             continue;
621         }
622
623         // Ignora linhas vazias ou de comentario
624         if (linha.empty() || linha[0] == '#') {
625             continue;
626         }
627
628         if (!lendoTreichos) {
629             // Aqui lemos a linha com os dados principais do poço
630             std::istringstream iss(linha);
631             std::string nome;
632             double profundidade, diamPoco, pressaoSup,
633                 pressaoSupFim;
634             double temperaturaSuperiorInicial,
635                 temperaturaFundoInicial;
636             double temperaturaSuperiorFinal, temperaturaFundoFinal;
637             std::string strHaPacker;
638
639             if (iss >> nome >> profundidade >> diamPoco >>
640                 pressaoSup >> pressaoSupFim
641                 >> temperaturaSuperiorInicial >>
642                     temperaturaFundoInicial
643                 >> temperaturaSuperiorFinal >>
644                     temperaturaFundoFinal >> strHaPacker) {
```

```

640
641         bool haPacker = (strHaPacker == "true" ||
642                           strHaPacker == "1");
643
644         ui->checkBoxPacker->setChecked(haPacker);
645         ui->btnAdicionarTrecho->setEnabled(true);
646         ui->btnRemoverTrecho->setEnabled(true);
647         ui->btnCalcularVariacoes->setEnabled(true);
648
649         poco = std::make_unique<CObjetoPoco>(
650             CObjetoPoco::CriarParaModulo02(nome,
651                 profundidade, diamPoco, pressaoSup,
652                 pressaoSupFim,
653
654                 temperaturaSuperiorInicial
655
656                 ,
657                 temperaturaFundoInicial
658
659                 ,
660                 temperaturaSuperiorFinal
661
662                 ,
663                 temperaturaFundoFinal
664
665                 ,
666                 haPacker)
667
668         );
669
670         leuPoco = true;
671
672     } else {
673
674         std::cerr << "Erro ao ler linha de po o : " <<
675             linha << std::endl;
676     }
677
678 } else {
679
680     // Aqui lemos os trechos e fluidos do po o
681
682     std::istringstream iss(linha);
683
684     std::string nomeTrecho, nomeFluido;
685
686     double profundInicial, profundFinal;
687
688     double diametroExterno, diametroInterno;
689
690     double coeficientePoisson, coeficienteExpansaoTermica;
691
692     double moduloElasticidade, pesoUnidade;
693
694     double densidade, viscosidade;
695
696
697     if (iss >> nomeTrecho >> profundInicial >> profundFinal
698
699         >> diametroExterno >> diametroInterno

```

```

672         >> coeficientePoisson >> coeficienteExpansaoTermica
673         >> moduloElasticidade >> pesoUnidade
674         >> nomeFluido >> densidade >> viscosidade) {
675
676         auto fluido = std::make_unique<CFluido>(nomeFluido,
677                                         densidade, viscosidade);
677         auto trechoPoco = std::make_unique<CTrechoPoco>(
678                                         nomeTrecho,
678                                         profundInicia
679                                         ,
679                                         profundFinal
679                                         ,
679                                         std
679                                         ::

679                                         move
679                                         (
679                                         fluido
679                                         )
679                                         ,
679                                         diametroExterno
679                                         ,
679                                         diametroIntern
679                                         ,
680                                         coeficienteF
680                                         ,
680                                         coeficienteE
680                                         ,
681                                         moduloElastico
681                                         ,
681                                         pesoUnida
682                                         );
683
684     if (!poco->AdicionarTrechoPoco(std::move(trechoPoco

```

```

        )) ) {
685         std::cerr << "Falha ao adicionar trecho ao
686             po o.\n";
687     } else {
688         leuTrecho = true;
689     }
690
691     } else {
692         std::cerr << "Erro ao ler trecho:" << linha << std
693             ::endl;
694     }
695
696     file.close();
697
698 // Se nao leu nem o po o ou nenhum trecho, mostra alerta
699 if (!leuPoco || !leuTrecho) {
700     QMessageBox::warning(this, "Arquivo Incorreto",
701                         "Aten o :o arquivo selecionado n o
702                             est no formato esperado.\n"
703                         "Por favor, abra um arquivo deu
704                             configura o v lido.");
705
706     return;
707 }
708
709
710
711
712 void CSimuladorPerdaTubulacao::on_actionNova_Simula_o_triggered()
713 {
714     QMessageBox::StandardButton resposta = QMessageBox::question(
715         this,
716         "",
717         "Tem certeza que deseja iniciar uma nova simula o?",
718         QMessageBox::Yes | QMessageBox::No
719     );
720
721     if (resposta == QMessageBox::Yes) {

```

```

722     CSimuladorPerdaTubulacao *newWindow = new
723         CSimuladorPerdaTubulacao();
724     newWindow->show();
725     this->close();
726 }
727
728
729 void CSimuladorPerdaTubulacao::
730     on_actionExportar_Como_Imagem_triggered()
731 {
732     QString fileName = QFileDialog::getSaveFileName(this, "Salvar"
733         " imagem", "", "PNG (*.png);;JPEG (*.jpg)");
734
735     if (!fileName.isEmpty()) {
736         QPixmap pixmap = this->grab();
737         pixmap.save(fileName);
738     }
739
740 void CSimuladorPerdaTubulacao::on_actionSobre_o_SEEP_triggered()
741 {
742     CJanelaSobreSoftware janelaSobre;
743     janelaSobre.setWindowTitle("Sobre o Software");
744     janelaSobre.exec();
745 }
746
747
748 void CSimuladorPerdaTubulacao::SalvarArquivo(bool salvarComo)
749 {
750     QString caminho;
751
752     // Se for salvarComo ou ainda não tiver caminho, abrir o
753     // diálogo
754     if (salvarComo || CaminhoArquivo().isEmpty()) {
755         caminho = QFileDialog::getSaveFileName(this, "Salvar"
756             " Arquivo", "", "Arquivo DAT (*.dat)");
757         if (caminho.isEmpty()) return; // usuário cancelou
758         CaminhoArquivo(caminho);
759         NomeArquivo(QFileInfo(caminho).fileName());
760     } else {

```

```

759     caminho = CaminhoArquivo(); // salva direto
760 }
761
762 QFile arquivo(caminho);
763 if (!arquivo.open(QIODevice::WriteOnly | QIODevice::Text)) {
764     QMessageBox::warning(this, "Erro", "Nao foi possivel salvar
765         o arquivo.");
766     return;
767 }
768 QTextStream out(&arquivo);
769
770 out << "# Configuracao do Poco
771 -----n";
772 out << "#"
773     << QString("Nome").leftJustified(35, ' ')
774     << QString("Profundidade(ft)").leftJustified(35, ' ')
775     << QString("Diametro do Poco(ft)").leftJustified(35, ' ')
776     << QString("Pressao Sup. Inicial(psi)").leftJustified(35,
777                 ' ')
778     << QString("Pressao Sup. Final(psi)").leftJustified(35, ' ')
779     << QString("Temp Sup. Inicial(F)").leftJustified(35, ' ')
780     << QString("Temp Fund. Inicial(F)").leftJustified(35, ' ')
781     << QString("Temp Sup. Final(F)").leftJustified(35, ' ')
782     << QString("Temp Fund. Final(F)").leftJustified(35, ' ')
783     << QString("Prof Packer(ft)").leftJustified(35, ' ')
784     << "\n";
785
786 out << " "
787     << ui->editNomePoco->text().leftJustified(35, ' ')
788     << ui->editProfundidadeTotal->text().leftJustified(35, ' ')
789     << ui->editDiametroPoco->text().leftJustified(35, ' ')
790     << ui->editPressaoSupInicial->text().leftJustified(35, ' ')
791     << ui->editPressaoSupFinal->text().leftJustified(35, ' ')
792     << ui->editTemperaturaSuperiorInicial->text().leftJustified
793         (35, ' ')
794     << ui->editTemperaturaFundoInicial->text().leftJustified
795         (35, ' ')

```

```

792     << ui->editTemperaturaSuperiorFinal->text().leftJustified
793         (35, ' ')
794     << ui->editTemperaturaFundoFinal->text().leftJustified(35,
795         ' ');
796
797     // Checkbox de saida
798     if (ui->checkBoxPacker->isChecked()) {
799         out << "true";
800     } else {
801         out << "false";
802     }
803
804
805     // Escreve os dados dos fluidos
806     out << "\n\n\n#\u00c9Configurac\u00e3o\u00e3o\u00d7Fluidos\u00d7
807
808     -----
809     n";
810
811     out << "#"
812
813         << QString("Nome\u20acTrecho").leftJustified(25, ' ')
814         << QString("Prof.\u20acInicial\u20ac(ft)").leftJustified(25, ' ')
815         << QString("Prof.\u20acFinal\u20ac(ft)").leftJustified(25, ' ')
816         << QString("Diam.\u20acexterno\u20ac(in)").leftJustified(25, ' ')
817         << QString("Diam.\u20acinterno\u20ac(in)").leftJustified(25, ' ')
818         << QString("Coef.\u20acPoisson").leftJustified(25, ' ')
819         << QString("Coef.\u20acExp.\u20acTerm.\u20ac(1/F)").leftJustified(25, ' ')
820         << QString("Mod.\u20acElast.\u20ac(psi)").leftJustified(25, ' ')
821         << QString("Peso/unid\u20ac(lb/ft)").leftJustified(25, ' ')
822         << QString("Nome\u20acfluido").leftJustified(25, ' ')
823         << QString("Densidade\u20ac(lbm/gal)").leftJustified(25, ' ')
824         << QString("Viscosidade\u20ac(cP)").leftJustified(25, ' ')
825
826         << "\n";
827
828     int linhas = ui->tblFluidos->rowCount();
829
830     for (int i = 0; i < linhas; ++i) {
831         out << "##" // recuo
832
833         << ui->tblTrechos->item(i, 0)->text().leftJustified(25,
834             ' ')
835
836         << ui->tblTrechos->item(i, 1)->text().leftJustified(25,
837             ' ')
838
839         << ui->tblTrechos->item(i, 2)->text().leftJustified(25,

```

```

        '□')
828     << ui->tblTrechos->item(i, 3)->text().leftJustified(25,
        '□')
829     << ui->tblTrechos->item(i, 4)->text().leftJustified(25,
        '□')
830     << ui->tblTrechos->item(i, 5)->text().leftJustified(25,
        '□')
831     << ui->tblTrechos->item(i, 6)->text().leftJustified(25,
        '□')
832     << ui->tblTrechos->item(i, 7)->text().leftJustified(25,
        '□')
833     << ui->tblTrechos->item(i, 8)->text().leftJustified(25,
        '□')
834     << ui->tblFluidos->item(i, 0)->text().leftJustified(25,
        '□')
835     << ui->tblFluidos->item(i, 1)->text().leftJustified(25,
        '□')
836     << ui->tblFluidos->item(i, 2)->text().leftJustified(25,
        '□')
837     << "\n";
838 }
839
840     arquivo.close();
841
842     // Atualiza o caminho salvo apenas se for novo
843     if (CaminhoArquivo().isEmpty())
844     {
845         CaminhoArquivo(caminho);
846         NomeArquivo(QFileInfo(caminho).fileName());
847     }
848
849     QMessageBox::information(this, "Salvo", "Arquivo salvo com
        sucesso!");
850 }
851
852 void CSimuladorPerdaTubulacao::on_actionSalvar_triggered()
853 {
854     SalvarArquivo(false); // salvar direto
855 }
856
857 void CSimuladorPerdaTubulacao::on_actionSalvar_como_triggered()
858 {

```

```

859     SalvarArquivo(true); // for ar abrir QFileDialog
860 }
861
862 // essa funcao edita os dados do fluido e do trecho com base na
863 // linha da tabela de fluidos
863 void CSimuladorPerdaTubulacao::EditarLinhaTabela(int row)
864 {
865     // garante que o ndice da linha v lido
866     if (row < 0 || row >= poco->Trechos().size()) {
867         return;
868     }
869
870     // obtém o trecho e fluido da mesma linha
871     CTrechoPoco* trecho = poco->Trechos().at(row);
872     CFluido* fluido = trecho->Fluido();
873
874     if (!fluido) return;
875
876     trecho->Nome(ui->tblTrechos->item(row, 0)->text().toStdString())
877         );
878     trecho->ProfundidadeInicial(ui->tblTrechos->item(row, 1)->text()
879         .toDouble());
880     trecho->ProfundidadeFinal(ui->tblTrechos->item(row, 2)->text()
881         .toDouble());
882     trecho->DiametroExterno(ui->tblTrechos->item(row, 3)->text()
883         .toDouble());
884     trecho->DiametroInterno(ui->tblTrechos->item(row, 4)->text()
885         .toDouble());
886     trecho->CoeficientePoisson(ui->tblTrechos->item(row, 5)->text()
887         .toDouble());
888     trecho->CoeficienteExpancaoTermica(ui->tblTrechos->item(row, 6)
889         ->text().toDouble());
890     trecho->ModuloElasticidade(ui->tblTrechos->item(row, 7)->text()
891         .toDouble());
892     trecho->PesoUnidade(ui->tblTrechos->item(row, 8)->text()
893         .toDouble());
894
895     fluido->Nome(ui->tblFluidos->item(row, 0)->text().toStdString())
896         );
897     fluido->Densidade(ui->tblFluidos->item(row, 1)->text().toDouble()
898         );
899     fluido->Viscosidade(ui->tblFluidos->item(row, 2)->text().

```

```
889    toDouble());
890
891     // atualiza valores globais do simulador
892     AtualizarDados();
893
894     ui->statusbar->showMessage("Fluido e profundidades atualizados com sucesso!");
895 }
```

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
   dados do po e permite a execu  o de c lculos e
   simula es
9
10 class CObjetoPoco {
11 protected:
12     std::string nomePoco;
13     double profundidadeFinal = 0.0;
14     double profundidadeOcupada = 0.0;
15     double pressaoSuperficie = 0.0;
16     double diametroPoco = 0.0;
17     double diametroRevestimentoOD = 0.0;
18     double diametroRevestimentoID = 0.0;
19     double vazao = 0.0;
20     std::vector<std::unique_ptr<CTrechoPoco>> trechos;
21
22     double pressaoSuperficieFim = 0.0;
23     double temperaturaTopoInicial = 0.0;
24     double temperaturaFundoInicial = 0.0;
25     double temperaturaTopoFinal = 0.0;
26     double temperaturaFundoFinal = 0.0;
27     double packer = true;
28 }
```

```

29 public:
30     // Construtor e destrutor padrão
31     CObjetoPoco() = default;
32     ~CObjetoPoco() = default;
33
34     // Evita cópia (pra evitar duplicar desnecessariamente os
35     // trechos)
36     CObjetoPoco(const CObjetoPoco&) = delete;
37     CObjetoPoco& operator=(const CObjetoPoco&) = delete;
38
39     // Permite movimentação (move semantics)
40     CObjetoPoco(CObjetoPoco&&) = default;
41     CObjetoPoco& operator=(CObjetoPoco&&) = default;
42
43     // Métodos de criação estáticos, separando claramente os
44     // dados exigidos por cada módulo
45     static CObjetoPoco CriarParaModulo01(std::string Nome, double
46                                         Profund, double PressaoSup, double D, double OD, double ID,
47                                         double q);
48     static CObjetoPoco CriarParaModulo02(std::string Nome, double
49                                         Profund, double diamPoco, double PressaoSup, double
50                                         PressaoSupFinal, double TempTopoInicial, double
51                                         TempFundoInicial, double TempTopoFinal, double
52                                         TempFundoFinal, bool haPacker);
53
54     // Getters para acessar os atributos de forma segura
55     std::string NomePoco() const { return nomePoco; }
56     double ProfundidadeTotal() const { return profundidadeFinal; }
57     double ProfundidadeOcupada() const { return profundidadeOcupada;
58     }
59     double PressaoSuperficie() const { return pressaoSuperficie; }
60     double PressaoSuperficieFim() const { return
61         pressaoSuperficieFim; }
62     double DiametroPoco() const { return diametroPoco; }
63     double DiametroRevestimentoOD() const { return
64         diametroRevestimentoOD; }
65     double DiametroRevestimentoID() const { return
66         diametroRevestimentoID; }
67     double Vazao() const { return vazao; }
68     double TemperaturaTopoInicial() const { return
69         temperaturaTopoInicial; }
70     double TemperaturaFundoInicial() const { return

```

```

        temperaturaFundoInicial; }

58     double TemperaturaTopoFinal() const { return
        temperaturaTopoFinal; }

59     double TemperaturaFundoFinal() const { return
        temperaturaFundoFinal; }

60     bool Packer() const { return packer; }

61

62     // Retorna os trechos adicionados ao po o (em forma de
        ponteiros brutos)

63     std::vector<CTrechoPoco*> Trechos() const;

64

65     // Setters permitem modificar os atributos do po o
66     void NomePoco(std::string Nome) { nomePoco = Nome; }
67     void ProfundidadeTotal(double Profundidade) { profundidadeFinal
        = Profundidade; }
68     void ProfundidadeOcupada(double Profundidade) {
        profundidadeOcupada = Profundidade; }
69     void PressaoSuperficie(double PressaoSuperior) {
        pressaoSuperficie = PressaoSuperior; }
70     void PressaoSuperficieFim(double PressaoSuperior) {
        pressaoSuperficieFim = PressaoSuperior; }
71     void DiametroPoco(double D) { diametroPoco = D; }
72     void DiametroRevestimentoOD(double DiametroExterno) {
        diametroRevestimentoOD = DiametroExterno; }
73     void DiametroRevestimentoID(double DiametroInterno) {
        diametroRevestimentoID = DiametroInterno; }
74     void Vazao(double q) { vazao = q; }
75     void TemperaturaTopoInicial(double temperatura) {
        temperaturaTopoInicial = temperatura; }
76     void TemperaturaFundoInicial(double temperatura) {
        temperaturaFundoInicial = temperatura; }
77     void TemperaturaTopoFinal(double temperatura) {
        temperaturaTopoFinal = temperatura; }
78     void TemperaturaFundoFinal(double temperatura) {
        temperaturaFundoFinal = temperatura; }
79     void Packer(bool haPacker) { packer = haPacker; }

80

81     // M todos de c lculo
82     double PressaoHidroestaticaTotal() const;
83     double PressaoHidroestaticaNoPonto(double profundidade) const;
84     double DensidadeEfetivaTotal() const;

```

```

86     double ViscosidadeEfetivaTotal() const;
87     bool VerificarPreenchimentoColuna();
88     double Carga(double profundidade, bool inicio) const;
89     double DeltaLTemperatura(double profundidade) const;
90     double DeltaLEfeitoBalao(double profundidade) const;
91     double VariacaoCargaDevidoCrossover(double profundidade, bool
92         deCimaParaBaixo, bool inicio) const;
93     double VariacaoCargaEfeitoPistao(double profundidade, double ID
94         , double OD) const;
95     double DeltaLPistaoPacker(double profundidade) const;
96     double DeltaLPistaoCrossover(double profundidade) const;
97     double DeltaLForcaRestauradora(double profundidade) const;
98     double CargaInjecao(double profundidade) const;
99     double TemperaturaNoPonto(double profundidade, double T_topo,
100        double T_Fundo) const;
101
102
103     bool AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco>
104         trechoParaAdicionar); // Função para adicionar um novo
105         trecho ao poço
106     void RemoverFluidoPoco(const std::string& nomeFluido); // Remove
107         trecho do poço com base no nome do fluido
108     void RemoverTrechoPoco(const std::string& nomeTrecho);
109
110     // Métodos que retornam dados para gráficos
111     std::pair<std::vector<double>, std::vector<double>>
112         PlotarProfundidadePorPressao();
113     std::pair<std::vector<double>, std::vector<double>>
114         PlotarProfundidadePorPressaoMedia();
115 };
116
117
118 #endif

```

Fonte: produzido pelo autor.

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

Listing 7.7: Arquivo de implementação da classe CObjetoPoco

```

1 #include "CObjetoPoco.h"
2 #include <iostream>
3 #include <vector>
4 #include <fstream>
5 #include <cstdlib>
6 #include <numbers>
7 #include <math.h>

```

```

8 #include <QDebug>
9
10
11 CObjetoPoco CObjetoPoco::CriarParaModulo01(std::string nomeDoPoco,
12                                             double profundidadeFinalDoPoco, double pressaoNaSuperficie,
13                                             double diametroDoPoco,
14                                             double
15                                             diametroRevestimentoExterno
16                                             , double
17                                             diametroRevestimentoInterno
18                                             , double vazaoDoPoco)
19 {
20
21     CObjetoPoco objetoPoco;
22
23     objetoPoco.nomePoco = nomeDoPoco;
24     objetoPoco.profundidadeFinal = profundidadeFinalDoPoco;
25     objetoPoco.pressaoSuperficie = pressaoNaSuperficie;
26     objetoPoco.diametroPoco = diametroDoPoco;
27     objetoPoco.diametroRevestimento0D = diametroRevestimentoExterno
28     ;
29     objetoPoco.diametroRevestimentoID = diametroRevestimentoInterno
30     ;
31     objetoPoco.vazao = vazaoDoPoco;
32
33     return objetoPoco;
34 }
35
36
37
38 CObjetoPoco CObjetoPoco::CriarParaModulo02(std::string nomeDoPoco,
39                                             double profundidadeFinalDoPoco, double diamPoco, double
40                                             pressaoNaSuperficie, double pressaoNaSuperficieFim,
41                                             double
42                                             temperaturaTopoInicial
43                                             , double
44                                             temperaturaFundoInicial
45                                             ,
46                                             double
47                                             temperaturaTopoFinal,
48                                             double
49                                             temperaturaFundoFinal
50                                             , bool haPacker) {
51
52     CObjetoPoco objetoPoco;

```

```

31
32     objetoPoco.nomePoco = nomeDoPoco;
33     objetoPoco.profundidadeFinal = profundidadeFinalDoPoco;
34     objetoPoco.diametroPoco = diamPoco;
35     objetoPoco.pressaoSuperficie = pressaoNaSuperficie;
36     objetoPoco.pressaoSuperficieFim = pressaoNaSuperficieFim;
37     objetoPoco.temperaturaTopoInicial = temperaturaTopoInicial;
38     objetoPoco.temperaturaFundoInicial = temperaturaFundoInicial;
39     objetoPoco.temperaturaTopoFinal = temperaturaTopoFinal;
40     objetoPoco.temperaturaFundoFinal = temperaturaFundoFinal;
41     objetoPoco.packer = haPacker;
42
43     return objetoPoco;
44 }
45
46
47 std::vector<CTrechoPoco*> CObjetoPoco::Trechos() const {
48     std::vector<CTrechoPoco*> vetorDePonteirosParaTrechos;
49
50     // percorre todos os trechos armazenados no po o
51     for (const auto& trechoUnico : trechos) {
52         // adiciona o ponteiro cru (n o nico ) de cada trecho ao
53         // vetor de sa da
54         vetorDePonteirosParaTrechos.push_back(trechoUnico.get());
55     }
56
57     // retorna o vetor contendo os ponteiros dos trechos
58     return vetorDePonteirosParaTrechos;
59 }
60
61 bool CObjetoPoco::AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco>
62                                         trechoParaAdicionar) {
63     // calcula o comprimento do trecho com base nas profundidades
64     // inicial e final
65     double comprimentoDoTrecho = trechoParaAdicionar->
66         ProfundidadeFinal() - trechoParaAdicionar->
67         ProfundidadeInicial();
68
69     // move o trecho para dentro do vetor principal do po o
70     trechos.push_back(std::move(trechoParaAdicionar));
71
72     // atualiza a profundidade total ocupada no po o somando o

```

```

        novo trecho
68     profundidadeOcupada += comprimentoDoTrecho;
69
70     return true; // opera o realizada com sucesso
71 }
72
73
74 void CObjetoPoco::RemoverFluidoPoco(const std::string& nomeFluido)
{
    for (auto it = trechos.begin(); it != trechos.end();) {
        if ((*it)->Fluido()->Nome() == nomeFluido) {
            double comprimento = (*it)->ProfundidadeFinal() - (*it)
                ->ProfundidadeInicial();
            it = trechos.erase(it);
            profundidadeOcupada -= comprimento;
        } else {
            ++it;
        }
    }
}
85
86 void CObjetoPoco::RemoverTrechoPoco(const std::string& nomeTrecho)
{
    for (auto it = trechos.begin(); it != trechos.end();) {
        if ((*it)->Nome() == nomeTrecho) {
            double comprimento = (*it)->ProfundidadeFinal() - (*it)
                ->ProfundidadeInicial();
            it = trechos.erase(it);
            profundidadeOcupada -= comprimento;
        } else {
            ++it;
        }
    }
}
97
98 double CObjetoPoco::PressaoHidroestaticaTotal() const {
99     double pressaoTotalHidrostatica = 0.0;
100
101     // soma a pressao hidrostatica de todos os trechos do p o o
102     for (const auto& trechoAtual : trechos) {
103         pressaoTotalHidrostatica += trechoAtual->
            PressaoHidroestatica();

```

```

104    }
105
106    // adiciona a pressao da superficie para obter a pressao total
107    return pressaoTotalHidrostatica + pressaoSuperficie;
108}
109
110 double CObjetoPoco::PressaoHidroestaticaNoPonto(double
111     profundidadeDesejada) const {
112     double pressaoAcumulada = pressaoSuperficie;
113     double profundidadeJaCalculada = 0.0;
114
115     // percorre cada trecho verificando se ele contem a
116     // profundidade desejada
117     for (const auto& trechoAtual : trechos) {
118         double comprimentoDoTrecho = trechoAtual->ProfundidadeFinal
119             () - trechoAtual->ProfundidadeInicial();
120
121         // se a profundidade estiver dentro do trecho atual
122         if (profundidadeDesejada <= profundidadeJaCalculada +
123             comprimentoDoTrecho) {
124             double profundidadeDentroDoTrecho =
125                 profundidadeDesejada - profundidadeJaCalculada;
126
127             // adiciona somente a parte proporcional da pressao no
128             // trecho
129             pressaoAcumulada += trechoAtual->PressaoHidroestatica(
130                 profundidadeDentroDoTrecho);
131             break;
132         } else {
133             // adiciona a pressao total do trecho completo
134             pressaoAcumulada += trechoAtual->PressaoHidroestatica()
135                 ;
136             profundidadeJaCalculada += comprimentoDoTrecho;
137         }
138     }
139
140     return pressaoAcumulada;
141 }
142
143 bool CObjetoPoco::VerificarPreenchimentoColuna() {
144     double profundidadeNaoPreenchida = profundidadeFinal -
145         profundidadeOcupada;

```

```

137
138     if (profundidadeNaoPreenchida > 0) {
139         std::cout << "Uma coluna de " << profundidadeNaoPreenchida
140             << " fluido precisa ser adicionada!" << std::endl;
141         return false; // coluna incompleta
142     } else {
143         std::cout << "A coluna de fluidos equivale a profundade "
144             << total do topo!" << std::endl;
145         return true; // coluna completamente preenchida
146     }
147 }
148 double CObjetoPoco::DensidadeEfetivaTotal() const {
149     double somaDensidadePonderada = 0.0;
150     double comprimentoTotalDaColuna = 0.0;
151
152     for (const auto& trechoAtual : trechos) {
153         double comprimentoTrecho = trechoAtual->ProfundidadeFinal()
154             - trechoAtual->ProfundidadeInicial();
155
156         // multiplica a densidade equivalente pelo comprimento do
157         // trecho para ponderar
158         somaDensidadePonderada += trechoAtual->DensidadeEquivalente
159             () * comprimentoTrecho;
160         comprimentoTotalDaColuna += comprimentoTrecho;
161     }
162
163     return somaDensidadePonderada / comprimentoTotalDaColuna;
164 }
165
166 double CObjetoPoco::ViscosidadeEfetivaTotal() const {
167     double somaDasViscosidades = 0.0;
168
169     // percorre todos os trechos para somar as viscosidades dos
170     // fluidos
171     for (const auto& trechoAtual : trechos) {
172         somaDasViscosidades += trechoAtual->Fluido()->Viscosidade()
173             ;
174     }
175
176     // retorna a media simples das viscosidades

```

```

172     return somaDasViscosidades / trechos.size();
173 }
174
175 std::pair<std::vector<double>, std::vector<double>> CObjetoPoco::
176     PlotarProfundidadePorPressaoMedia() {
177         std::vector<double> vetorDeProfundidades;
178         std::vector<double> vetorDePressoes;
179
180         // percorre cada metro ao longo da profundidade total do po o
181         for (double profundidadeAtual = 0.0; profundidadeAtual <=
182             ProfundidadeTotal(); profundidadeAtual += 1.0) {
183             vetorDeProfundidades.push_back(profundidadeAtual);
184
185             // calcula a pressao no ponto atual usando a funcao
186             // apropriada
187             double pressaoNoPonto = PressaoHidroestaticaNoPonto(
188                 profundidadeAtual);
189             vetorDePressoes.push_back(pressaoNoPonto);
190         }
191
192         // retorna as duas listas para serem usadas na geracao do
193         // grafico
194         return std::make_pair(vetorDeProfundidades, vetorDePressoes);
195     }
196
197     double CObjetoPoco::Carga(double profundidade, bool inicio) const {
198         double cargaTotal = 0.0;
199         double pi = 3.141592653589793;
200
201         if (trechos.empty())
202             return 0.0;
203
204         // 1. Pressao no fundo do poco (base)
205         double profundidadeBase = 0.0;
206         double OD_base = 0.0;
207         double ID_base = 0.0;
208
209         for (const auto& trecho : trechos) {
210             if (trecho->ProfundidadeFinal() > profundidadeBase) {
211                 profundidadeBase = trecho->ProfundidadeFinal();
212                 OD_base = trecho->DiametroExterno();
213                 ID_base = trecho->DiametroInterno();
214             }
215         }
216
217         // 2. Pressao no topo do poco (superficie)
218         double profundidadeSuperficie = profundidadeBase +
219             (profundidade - profundidadeBase) * OD_base / (OD_base - ID_base);
220
221         double pressaoSuperficie = PressaoHidroestaticaNoPonto(
222             profundidadeSuperficie);
223
224         // 3. Calcula a diferenca de pressao entre o fundo e a superficie
225         double diferencaPressao = pressaoSuperficie - pressaoNoPonto;
226
227         // 4. Calcula a carga total
228         double cargaUnitaria = diferencaPressao * OD_base * pi;
229         double cargaTotal = cargaUnitaria * profundidadeBase;
230
231         if (inicio)
232             return -cargaTotal;
233         else
234             return +cargaTotal;
235     }

```

```

209     }
210 }
211
212 // 2. Carga por pressao no fundo (negativa)
213 double pressaoBase = PressaoHidroestaticaNoPonto(
214     profundidadeBase);
215 if (inicio == false){
216     pressaoBase = PressaoHidroestaticaNoPonto(profundidadeBase)
217         + PressaoSuperficieFim();
218 }
219
220 double areaBase = (pi / 4.0) * (OD_base * OD_base - ID_base *
221     ID_base);
222 double cargaPressao = -1.0 * pressaoBase * areaBase;
223 cargaTotal += cargaPressao;
224
225 // 3. Soma pesos de trechos abaixo da profundidade
226 for (const auto& trecho : trechos) {
227     double z_i = trecho->ProfundidadeInicial();
228     double z_f = trecho->ProfundidadeFinal();
229     double pesoUnit = trecho->PesoUnidade();
230
231     // Caso 1: trecho totalmente abaixo da profundidade
232     if (z_i >= profundidade) {
233         double L_total = z_f - z_i;
234         double cargaPeso = L_total * pesoUnit;
235         cargaTotal += cargaPeso;
236     }
237
238     // Caso 2: profundidade dentro do trecho
239     else if (profundidade > z_i && profundidade < z_f) {
240         double L_parcial = z_f - profundidade;
241         double cargaPeso = L_parcial * pesoUnit;
242         cargaTotal += cargaPeso;
243     }
244
245 // 4. considerando as Cargas por efeito pisto
246 if (inicio){
247     cargaTotal += VariacaoCargaDevidoCrossover(profundidade,

```

```

                false , true);
248 }
249 else{
250     cargaTotal += VariacaoCargaDevidoCrossover(profundidade ,
251                                                 false , false);
252 }
253 return cargaTotal;
254 }
255
256 double CObjetoPoco::DeltaLTemperatura(double profundidade) const {
257     double deltaLTotal = 0.0;
258
259     for (const auto& trecho : trechos) {
260         double z_i = trecho->ProfundidadeInicial();
261         double z_f = trecho->ProfundidadeFinal();
262         double ct = trecho->CoeficienteExpancaoTermica();
263
264         // Ignora trechos que estao totalmente abaixo da
265         // profundidade informada
266         if (z_i >= profundidade)
267             continue;
268
269         // Calcula temperatura media inicial e final do trecho
270         double tMedioInicial = (
271             TemperaturaNoPonto(z_i ,
272                                 TemperaturaTopoInicial() ,
273                                 TemperaturaFundoInicial()) +
274             TemperaturaNoPonto(z_f ,
275                                 TemperaturaTopoInicial() ,
276                                 TemperaturaFundoInicial())
277             ) / 2.0;
278
279         double tMedioFinal = (
280             TemperaturaNoPonto(z_i ,
281                                 TemperaturaTopoFinal() ,
282                                 TemperaturaFundoFinal()) +
283             TemperaturaNoPonto(z_f ,
284                                 TemperaturaTopoFinal() ,
285                                 TemperaturaFundoFinal())
286             ) / 2.0;
287
288     }
289 }
```

```

279     // Variacao de temperatura final - inicial
280     double deltaT = tMedioFinal - tMedioInicial;
281
282     // Caso raro onde nao houve variacao de temperatura entre
283     // as duas condicoes
284     // A gente for a um deltaT usando a temperatura no ponto
285     // zero como referencia
286     if (deltaT == 0) {
287         double media = (
288             TemperaturaNoPonto(z_i,
289                 TemperaturaTopoFinal(),
290                 TemperaturaFundoFinal()) +
291             TemperaturaNoPonto(z_f,
292                 TemperaturaTopoFinal(),
293                 TemperaturaFundoFinal())
294         ) / 2.0;
295
296         deltaT = TemperaturaNoPonto(0, TemperaturaTopoFinal(),
297             TemperaturaFundoFinal()) - media;
298     }
299
300     double L = 0.0;
301
302     // Se a profundidade estiver dentro do trecho, calcula o
303     // comprimento parcial
304     if (profundidade > z_i && profundidade < z_f) {
305         L = profundidade - z_i;
306     }
307     // Se o trecho estiver totalmente acima da profundidade,
308     // considera L total
309     else if (z_f <= profundidade) {
310         L = z_f - z_i;
311     }
312
313     // Soma o deltaL do trecho na variavel acumuladora
314     deltaLTotal += ct * L * deltaT;
315
316 }
317
318     return deltaLTotal;
319 }
320
321

```

```

312
313
314 double CObjetoPoco::TemperaturaNoPonto(double profundidade, double
315     T_topo, double T_Fundo) const {
316
317     double inclinacao = (T_Fundo - T_topo) / ProfundidadeOcupada();
318     double temperatura = T_topo + inclinacao * profundidade;
319
320 }
321
322 double CObjetoPoco::VariacaoCargaDevidoCrossover(double
323     profundidade, bool deCimaParaBaixo, bool inicio) const {
324
325     double pi = 3.141592653589793;
326     double variacao_total = 0.0;
327
328
329     if (trechos.size() < 2)
330         return 0.0;
331
332     // percorre todas as interfaces entre os trechos consecutivos
333     for (size_t i = 1; i < trechos.size(); ++i) {
334
335         const auto& trecho_acima = trechos[i - 1];
336         const auto& trecho_abixo = trechos[i];
337
338         double z_interface = trecho_abixo->ProfundidadeInicial();
339
340         // define se a interface deve ser considerada com base na
341         // direcao da analise
342         bool considerar = false;
343         if (!deCimaParaBaixo && z_interface >= profundidade -
344             tolerancia)
345             considerar = true;
346         if (deCimaParaBaixo && z_interface <= profundidade +
347             tolerancia)
348             considerar = true;
349
350         if (!considerar)
351             continue;
352
353         if (deCimaParaBaixo)
354             variacao_total += pi * (z_interface - profundidade) *
355                 (trecho_acima->ProfundidadeFinal() - trecho_abixo-
356                 >ProfundidadeFinal());
357         else
358             variacao_total -= pi * (z_interface - profundidade) *
359                 (trecho_acima->ProfundidadeFinal() - trecho_abixo-
360                 >ProfundidadeFinal());
361
362     }
363
364     return variacao_total;
365 }

```

```

349     // coleta diametros internos e externos dos dois trechos
350     double ID_acima = trecho_acima->DiametroInterno();
351     double ID_abixo = trecho_abixo->DiametroInterno();
352     double OD_acima = trecho_acima->DiametroExterno();
353     double OD_abixo = trecho_abixo->DiametroExterno();

354
355     // se nao houver mudanca de diametro, nao ha efeito pistao
356     if (std::abs(ID_acima - ID_abixo) < 1e-6 &&
357         std::abs(OD_acima - OD_abixo) < 1e-6) {
358         continue;
359     }

360
361     // calcula diferenca de area interna e externa
362     double Ai = (pi / 4.0) * (ID_acima * ID_acima - ID_abixo *
363                               ID_abixo);
364     double Ao = (pi / 4.0) * (OD_acima * OD_acima - OD_abixo *
365                               OD_abixo);

366     // pressao no ponto da interface
367     double pressao = PressaoHidroestaticaNoPonto(z_interface);
368     double carga_crossover = 0.0;

369     // tratamento com ou sem packer
370     if (Packer()) {
371         if (inicio) {
372             // no inicio da coluna: pressao interna = externa =
373             // pressao hidrostatica
374             carga_crossover = pressao * (Ai - Ao);
375         } else {
376             // no fim da coluna: pressao interna = pressao
377             // hidrostatica + pressao superficial final
378             carga_crossover = (pressao - (pressao +
379             PressaoSuperficieFim())) * Ai;
380         }
381     } else {
382         // sem packer: pressao interna e externa sao iguais
383         carga_crossover = pressao * (Ai - Ao);
384     }

385     variacao_total += carga_crossover;
}

```

```

386     return variacao_total;
387 }
388
389
390 #include <QDebug>
391 double CObjetoPoco::VariacaoCargaEfeitoPistao(double profundidade,
392                                                 double ID, double OD) const {
393     const double pi = 3.141592653589793;
394
395     // rea da parede interna do tubo
396     double Ain = (pi / 4.0) * (OD * OD - ID * ID);
397
398     double Aout = Ain;
399
400
401     // Press es internas
402     double Pin_inicio = PressaoHidroestaticaNoPonto(profundidade) +
403                     PressaoSuperficie();
404     double Pin_fim = PressaoHidroestaticaNoPonto(profundidade) +
405                     PressaoSuperficieFim();
406     double deltaPin = Pin_fim - Pin_inicio;
407
408     // Press es externas
409     double Pout_inicio;
410     double Pout_fim;
411
412     if (Packer() == true) {
413         Pout_inicio = 0;
414         Pout_fim = 0;
415     } else {
416         Pout_inicio = 0;
417         Pout_fim = 0;
418     }
419
420     // Resultado final
421     if (Packer() == true) {
422         return -deltaPin * Ain - deltaPout * Aout;
423     } else {
424         return deltaPin * Ain - deltaPout * Aout;

```

```

425
426     }
427 }
428
429 double CObjetoPoco::DeltaLPistaoPacker(double profundidade) const {
430     double pi = 3.141592653589793;
431     double deltaL_total = 0.0;
432
433     // verifica se ha qualquer mudanca de diametro externo entre os
434     // trechos
435     bool possuiDiferencaOD = false;
436     double primeiroOD = trechos[0] -> DiametroExterno();
437     for (const auto& trecho : trechos) {
438         if (std::abs(trecho->DiametroExterno() - primeiroOD) > 1e
439             -6) {
440             possuiDiferencaOD = true;
441             break;
442         }
443     }
444
445     for (size_t i = 0; i < trechos.size(); ++i) {
446         const auto& trecho = trechos[i];
447         double z_i = trecho->ProfundidadeInicial();
448         double z_f = trecho->ProfundidadeFinal();
449         double OD = trecho->DiametroExterno();
450         double ID = trecho->DiametroInterno();
451         double E = trecho->ModuloElasticidade();
452
453         double area_anular = (pi / 4.0) * (OD * OD - ID * ID);
454         if (E <= 0.0 || area_anular <= 0.0)
455             continue;
456
457         double L = 0.0;
458
459         if (profundidade > z_i && profundidade < z_f) {
460             L = profundidade - z_i;
461         } else if (z_f <= profundidade) {
462             L = z_f - z_i;
463         } else {
464             continue;
465         }
466     }

```

```

465     double CargaPistao = 0.0;
466     if (possuiDiferencaOD) {
467         // qualquer mudanca de OD: usa ID e OD do trecho
468         CargaPistao = VariacaoCargaEfeitoPistao(z_i, ID, OD);
469     } else {
470         // todos os trechos iguais: usa ID e o diametro total
471         // do poco
472         CargaPistao = VariacaoCargaEfeitoPistao(z_i, ID,
473                                         DiametroPoco());
474     }
475
476     double deltaL_trecho = (CargaPistao * L) / (E * area_anular
477 );
478     deltaL_total += deltaL_trecho;
479 }
480
481
482
483 double CObjetoPoco::DeltaLEfeitoBalao(double profundidade) const {
484     double pi = 3.141592653589793;
485     double deltaL_total = 0.0;
486
487     for (size_t i = 0; i < trechos.size(); ++i) {
488         const auto& trecho = trechos[i];
489         double z_i = trecho->ProfundidadeInicial();
490         double z_f = trecho->ProfundidadeFinal();
491         double OD = trecho->DiametroExterno();
492         double ID = trecho->DiametroInterno();
493         double E = trecho->ModuloElasticidade();
494         double v = trecho->CoeficientePoisson();
495
496         double area_anular = (pi / 4.0) * (OD * OD - ID * ID);
497
498         if (E <= 0.0 || area_anular <= 0.0)
499             continue;
500
501         double L = 0.0;
502
503         if (profundidade > z_i && profundidade < z_f) {

```

```

504         // trecho parcialmente acima da profundidade
505         L = profundidade - z_i;
506     } else if (z_f <= profundidade) {
507         // trecho totalmente acima da profundidade
508         L = z_f - z_i;
509     } else {
510         continue; // trecho abaixo da profundidade, ignora
511     }
512
513     double CargaPistao = VariacaoCargaEfeitoPistao(z_i, 0, ID);
514     double deltaL_trecho = 2 * v * (CargaPistao * (L)) / (E *
515         area_anular); // o x12      uma unidade de conversao de
516         ft para in
517
518     deltaL_total += deltaL_trecho;
519 }
520
521
522 double CObjetoPoco::DeltaLPistaoCrossover(double profundidade)
523 {
524     const {
525         double pi = 3.141592653589793;
526         double deltaL_total = 0.0;
527
528         for (size_t i = 1; i < trechos.size(); ++i) {
529             const auto& trechoAcima = trechos[i - 1];
530             const auto& trechoAbaixo = trechos[i];
531
532             // Profundidade onde ocorre a mudanca de trecho
533             double z_interface = trechoAbaixo->ProfundidadeInicial();
534
535             // Ignora se a interface estiver abaixo da profundidade
536             // analisada
537             if (z_interface > profundidade)
538                 continue;
539
540             // Coleta os diametros externos para comparar se ha mudanca
541             double OD_acima = trechoAcima->DiametroExterno();
542             double ID_acima = trechoAcima->DiametroInterno();
543             double OD_abai

```

```

542         // Se os diametros forem quase iguais, nao ha efeito pistao
543         // relevante
544         if (std::abs(OD_acima - OD_abixo) < 1e-6)
545             continue;
546
547         // Area da secao transversal da coluna acima (quem vai
548         // deformar)
549         double area_secao = (pi / 4.0) * (OD_acima * OD_acima -
550                                         ID_acima * ID_acima);
551
552         // Obtem o modulo de elasticidade medio entre os dois
553         // trechos
554         double E1 = trechoAcima->ModuloElasticidade();
555         double E2 = trechoAbaixo->ModuloElasticidade();
556         double E_medio = (E1 + E2) / 2.0;
557
558         if (E_medio <= 0.0 || area_secao <= 0.0)
559             continue;
560
561         // Carga causada pelo efeito pistao na interface
562         double cargaPistao = VariacaoCargaEfeitoPistao(z_interface,
563                                                       OD_abixo, OD_acima);
564
565         // L = comprimento da coluna acima da interface
566         double L = z_interface;
567
568         // Deformacao axial provocada pela carga
569         double deltaL = (cargaPistao * L) / (E_medio * area_secao);
570         deltaL_total += deltaL;
571     }
572
573     return deltaL_total;
574 }
575
576 double CObjetoPoco::DeltaLForcaRestauradora(double profundidade)
577 const{
578
579     double deltaLTemperatura = DeltaLTemperatura(profundidade);
580     double deltaLBalao = DeltaLEfeitoBalao(profundidade);
581     double deltaLPacker = DeltaLPistaopacker(profundidade);
582     double deltaLCrossover = DeltaLPistaocrossover(profundidade);
583
584 }
```

```

578     return deltaLTemperatura + deltaLBalao + deltaLPacker +
579         deltaLCrossover;
580 }
581
582 double CObjetoPoco::CargaInjecao(double profundidade) const {
583     const double pi = 3.141592653589793;
584     double denominador = 0.0;
585
586     // Deformacao total desejada
587     double deltaL = DeltaLForcaRestauradora(profundidade);
588
589     for (const auto& trecho : trechos) {
590         double z_i = trecho->ProfundidadeInicial();
591         double z_f = trecho->ProfundidadeFinal();
592
593         // Pula trechos abaixo da profundidade desejada
594         if (z_i > profundidade)
595             continue;
596
597         double limiteSuperior = std::min(z_f, profundidade);
598         double comprimentoUtil = limiteSuperior - z_i;
599
600         if (comprimentoUtil <= 0.0)
601             continue;
602
603         double E = trecho->ModuloElasticidade();
604         double OD = trecho->DiametroExterno();
605         double ID = trecho->DiametroInterno();
606         double area = (pi / 4.0) * (OD * OD - ID * ID);
607
608         // soma das flexibilidades: L / (E*A)
609         denominador += comprimentoUtil / (E * area);
610     }
611
612     if (denominador == 0.0)
613         return 0.0;
614
615     // pressao = deformacao total / soma das flexibilidades
616     return deltaL / denominador;
617 }

```

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),
   viscosidadeFluido(viscosidade) {}
25
26     // getters
27     std::string Nome() const { return nomeFluido; }
28     double Densidade() const { return densidadeFluido; }
29     double Viscosidade() const { return viscosidadeFluido; }
30
31     // setters
32     void Nome(const std::string& novoNome) { nomeFluido = novoNome;
   }
33     void Densidade(double novaDensidade) { densidadeFluido =
   novaDensidade; }
34     void Viscosidade(double novaViscosidade) { viscosidadeFluido =
   novaViscosidade; }
```

```
35 };  
36  
37 #endif
```

Fonte: produzido pelo autor.

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

Listing 7.9: Arquivo de implementação da classe CFluido

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

Fonte: produzido pelo autor.

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

Listing 7.10: Arquivo de implementação da classe CModeloReologico

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

```

23
24     // Tipo de fluxo (laminar ou turbulento)
25     std::string fluxoPoco;
26     std::string fluxoAnular;
27
28     // Objeto que contem as propriedades do poco
29     CObjetoPoco* poco;
30
31 public:
32     // Construtores
33     CModeloReologico() {}
34     virtual ~CModeloReologico() {}
35     CModeloReologico(CObjetoPoco* poco) : poco(poco) {}
36
37     // Getters
38     double FatorFriccaoPoco() const { return fatorFriccaoPoco; }
39     double FatorFriccaoAnular() const { return fatorFriccaoAnular; }
40
41     double ReynoldsPoco() const { return reynoldsPoco; }
42     double ReynoldsAnular() const { return reynoldsAnular; }
43     double VMediaPoco() const { return vMediaPoco; }
44     double VMediaAnular() const { return vMediaAnular; }
45     std::string FluxoPoco() const { return fluxoPoco; }
46     std::string FluxoAnular() const { return fluxoAnular; }
47
48     // M todos para determinar fatores e velocidades
49     double DeterminarFatorFriccao(double re, double n);
50     double DeterminarReynoldsPoco();
51     double DeterminarReynoldsPoco(double viscosidade);
52     double DeterminarReynoldsAnular();
53     double DeterminarReynoldsAnular(double viscosidade);
54     double DeterminarVelocidadeMediaPoco();
55     double DeterminarVelocidadeMediaAnular();
56
57     // M todos puros (devem ser implementados pelas classes filhas
58     )
59     virtual std::string DeterminarFluxoPoco() = 0;
60     virtual std::string DeterminarFluxoAnular() = 0;
61     virtual double CalcularPerdaPorFriccaoPoco() = 0;
62     virtual double CalcularPerdaPorFriccaoAnular() = 0;
63 };

```

```
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
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 // Calcula o numero de Reynolds no espaco anular
28 double CModeloReologico::DeterminarReynoldsAnular() {
29     reynoldsAnular = (757 * poco->DensidadeEfetivaTotal() *
30                         vMediaAnular *
31                         (poco->DiametroPoco() - poco->
32                          DiametroRevestimentoOD())));
33
34     return reynoldsAnular;
35
36 // Mesmo calculo para o anular, com viscosidade recebida
```

```

externamente

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

```

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
   deformação
9 A classe herda de CModeloReologico e implementa os métodos

```

```

    especificos para esse tipo de fluido
10 */
11
12 class CModeloNewtoniano : public CModeloReologico {
13
14 public:
15     // Construtores
16     CModeloNewtoniano() {}
17     ~CModeloNewtoniano() {}

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

24
25     // Metodos obrigatorios sobrescritos do modelo base
26     std::string DeterminarFluxoPoco() override;
27     std::string DeterminarFluxoAnular() override;
28     double CalcularPerdaPorFriccaoPoco() override;
29     double CalcularPerdaPorFriccaoAnular() override;
30 };
31
32 #endif // CMODELONEWTONIANO_H

```

Fonte: produzido pelo autor.

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

Listing 7.13: Arquivo de implementação da classe CModeloNewtoniano

```

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

```

```

14 // Determina o tipo de fluxo no espaco anular com base no numero de
15 // Reynolds
16 std::string CModeloNewtoniano::DeterminarFluxoAnular() {
17     DeterminarVelocidadeMediaAnular();
18     DeterminarReynoldsAnular();
19
20     fluxoAnular = (reynoldsAnular <= 2100) ? "Laminar" : "
21         Turbulento";
22
23     return fluxoAnular;
24 }
25
26 // Calcula a perda de carga por friccao no poco para regime laminar
27 // ou turbulento
28
29 double CModeloNewtoniano::CalcularPerdaPorFriccaoPoco() {
30     if (fluxoPoco.empty()) {
31         DeterminarFluxoPoco();
32     }
33
34     if (fluxoPoco == "Laminar") {
35         return (poco->ViscosidadeEfetivaTotal() * vMediaPoco) /
36                 (1500 * std::pow(poco->DiametroRevestimentoID(), 2))
37                 ;
38
39     } else {
40         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std
41                 ::pow(vMediaPoco, 1.75) *
42                 std::pow(poco->ViscosidadeEfetivaTotal(), 0.25)) /
43                 (1800 * std::pow(poco->DiametroRevestimentoID(),
44                         1.25));
45     }
46 }
47
48 // Calcula a perda de carga por friccao no espaco anular
49 double CModeloNewtoniano::CalcularPerdaPorFriccaoAnular() {
50     if (fluxoAnular.empty()) {
51         DeterminarFluxoAnular();
52     }
53
54     double diametroAnular = poco->DiametroPoco() - poco->
55             DiametroRevestimentoOD();
56
57     if (fluxoAnular == "Laminar") {
58         return (poco->ViscosidadeEfetivaTotal() * vMediaAnular) /

```

```

49             (1000 * std::pow(diametroAnular, 2));
50     } else {
51         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std
52             ::pow(vMediaAnular, 1.75) *
53             std::pow(poco->ViscosidadeEfetivaTotal(), 0.25)) /
54             (1396 * std::pow(diametroAnular, 1.25));
55 }

```

Fonte: produzido pelo autor.

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

Listing 7.14: Arquivo de implementação da classe CModeloBingham

```

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

```

```

30     ~CModeloBingham() {}
31
32     // Construtor com ponteiro para o objeto CObjetoPoco
33     CModeloBingham(CObjetoPoco* poco) : CModeloReologico(poco) {}
34
35     // Construtor completo com parametros do modelo
36     CModeloBingham(CObjetoPoco* poco, double viscosidadePlastica,
37                     double pontoDeEscoamento)
38         : CModeloReologico(poco),
39           viscosidadePlastica(viscosidadePlastica),
40           pontoDeEscoamento(pontoDeEscoamento)
41     {
42         DeterminarFluxoPoco();
43         DeterminarFluxoAnular();
44     }
45
46     // Getters
47     double ViscosidadePlastica() const { return viscosidadePlastica;
48         ; }
49     double PontoDeEscoamento() const { return pontoDeEscoamento; }
50     double ReynoldsCriticoPoco() const { return reynoldsCriticoPoco;
51         ; }
52     double ReynoldsCriticoAnular() const { return
53         reynoldsCriticoAnular; }
54     double ReynoldsHedstronPoco() const { return
55         reynoldsHedstronPoco; }
56     double ReynoldsHedstronAnular() const { return
57         reynoldsHedstronAnular; }
58
59     // Setters
60     void ViscosidadePlastica(double valor) { viscosidadePlastica =
61         valor; }
62     void PontoDeEscoamento(double valor) { pontoDeEscoamento =
63         valor; }
64
65     // Metodos especificos do modelo de Bingham
66     double DeterminarReynoldsCritico(double hedstron);
67     double DeterminarReynoldsHedstronPoco();
68     double DeterminarReynoldsHedstronAnular();
69     std::string DeterminarFluxoPoco() override;
70     std::string DeterminarFluxoAnular() override;
71     double CalcularPerdaPorFriccaoPoco() override;

```

```

64     double CalcularPerdaPorFriccaoAnular() override;
65 };
66
67 #endif // CMODELOBINGHAM_H

```

Fonte: produzido pelo autor.

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

Listing 7.15: Arquivo de implementação da classe CModeloBingham

```

1 #include "CModeloBingham.h"
2 #include <cmath>
3 #include <QApplication>
4 #include <QFileDialog>
5 #include <QString>
6 #include <QMessageBox>
7
8 // Determina o valor de Reynolds critico com base no numero de
9 // Hedstrom
10 // Utiliza metodo numerico de Newton-Raphson para resolver a
11 // equacao implicita
12 double CModeloBingham::DeterminarReynoldsCritico(double hedstrom) {
13     // f(x) = ((x / (1 - x)^3) * 16800) - Hedstrom
14     auto func = [hedstrom](double x) {
15         return ((x / std::pow(1 - x, 3)) * 16800) - hedstrom;
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);

```

```

33     // Retorna o Reynolds critico com base em y encontrado
34     return ((1 - ((4.0 / 3.0) * y) + ((1.0 / 3.0) * std::pow(y, 4)))
35         * hedstron) / (8 * y);
36
37 // Calcula o numero de Hedstron para o escoamento no poco
38 double CModeloBingham::DeterminarReynoldsHedstronPoco() {
39     reynoldsHedstronPoco =
40         (37100 * poco->DensidadeEfetivaTotal() * pontoDeEscoamento
41             *
42             std::pow(poco->DiametroRevestimentoID(), 2)) /
43             std::pow(viscosidadePlastica, 2);
44
45     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 // Determina o tipo de fluxo no poco (laminar ou turbulento)
63 std::string CModeloBingham::DeterminarFluxoPoco() {
64     DeterminarVelocidadeMediaPoco();
65     DeterminarReynoldsPoco(viscosidadePlastica);
66     DeterminarReynoldsHedstronPoco();
67
68     reynoldsCriticoPoco = DeterminarReynoldsCritico(
69         reynoldsHedstronPoco);
70
71     fluxoPoco = (reynoldsPoco <= reynoldsCriticoPoco) ? "Laminar" :
72         "Turbulento";
73
74     return fluxoPoco;

```

```

69 }
70
71 // Determina o tipo de fluxo no espaco anular
72 std::string CModeloBingham::DeterminarFluxoAnular() {
73     DeterminarVelocidadeMediaAnular();
74     DeterminarReynoldsAnular(viscosidadePlastica);
75     DeterminarReynoldsHedstronAnular();
76
77     reynoldsCriticoAnular = DeterminarReynoldsCritico(
78         reynoldsHedstronAnular);
79
80     fluxoAnular = (reynoldsAnular <= reynoldsCriticoAnular) ? "
81         Laminar" : "Turbulento";
82     return fluxoAnular;
83 }
84
85 // Calcula a perda de carga por friccao no poco
86 double CModeloBingham::CalcularPerdaPorFriccaoPoco() {
87     if (fluxoPoco == "Laminar") {
88         return ((viscosidadePlastica * vMediaPoco) /
89                 (1500 * std::pow(poco->DiametroRevestimentoID(), 2)
90                  )) +
91                 (pontoDeEscoamento / (225 * poco->
92                     DiametroRevestimentoID())));
93     } else {
94         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std
95                 ::pow(vMediaPoco, 1.75) *
96                     std::pow(viscosidadePlastica, 0.25)) /
97                     (1800 * std::pow(poco->DiametroRevestimentoID(),
98                         1.25));
99     }
100 }
101
102 // Calcula a perda de carga por friccao no espaco anular
103 double CModeloBingham::CalcularPerdaPorFriccaoAnular() {
104     double diametroAnular = poco->DiametroPoco() - poco->
105         DiametroRevestimentoOD();
106
107     if (fluxoAnular == "Laminar") {
108         return ((viscosidadePlastica * vMediaAnular) /
109                 (1000 * std::pow(diametroAnular, 2))) +
110                 (pontoDeEscoamento / (200 * diametroAnular));

```

```

104     } else {
105         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std
106             ::pow(vMediaAnular, 1.75) *
107                 std::pow(viscosidadePlastica, 0.25)) /
108                 (1396 * std::pow(diametroAnular, 1.25));
109     }

```

Fonte: produzido pelo autor.

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

Listing 7.16: Arquivo de implementação da classe CModeloPotencia

```

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

```

```

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

```

```

63 } ;
64
65 #endif // CMODELOPOTENCIA_H

```

Fonte: produzido pelo autor.

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

Listing 7.17: Arquivo de implementação da classe CModeloPotencia

```

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

```

```

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

```

```

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

```

```

        indiceDeComportamento)) *
93            std::pow(((2 + (1 / indiceDeComportamento)) /
94                0.0208), indiceDeComportamento)) /
95            (144000 * std::pow(diametroAnular, 1 +
96                indiceDeComportamento));
97
98    } else {
99        return (fatorFriccaoAnular * poco->DensidadeEfetivaTotal()
101           * std::pow(vMediaAnular, 2)) /
102               (21.1 * diametroAnular);
103    }
104}

```

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 Q_OBJECT
18
19 public:
20     explicit CJanelaAdicionarFluido(QWidget *parent = nullptr);
21     ~CJanelaAdicionarFluido();
22

```

```

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

```

Fonte: produzido pelo autor.

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

Listing 7.19: Arquivo de implementação da classe CJanelaAdicionarFluido

```

1 #include "CJanelaAdicionarFluido.h"
2 #include "ui_CJanelaAdicionarFluido.h"
3 #include <QMessageBox>

```

```

4
5 // Construtor da janela - inicializa a interface grafica
6 CJanelaAdicionarFluido::CJanelaAdicionarFluido(QWidget *parent)
7     : QDialog(parent)
8     , ui(new Ui::CJanelaAdicionarFluido)
9 {
10     ui->setupUi(this);
11 }
12
13 // 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\u00e7\u00e3o favor, \u00e7\u00f5 preencha
38                                     \u2022 todos \u2022 os \u2022 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 propriidades fisicas e dados do fluido
11 Esses dados sao usados para simulacoes de comportamento termico e
12 mecanico da tubulacao
13 */
14
15
16 class CJanelaAdicionarTrechoTubulacao : public QDialog
```

```

17 {
18     Q_OBJECT
19
20 public:
21     explicit CJanelaAdicionarTrechoTubulacao(QWidget *parent =
22         nullptr);
23     ~CJanelaAdicionarTrechoTubulacao();
24
25     // metodos para acessar os dados inseridos
26     QString Trecho() const { return trecho; }
27     QString ProfundidadeInicial() const { return
28         profundidadeInicial; }
29     QString ProfundidadeFinal() const { return profundidadeFinal; }
30     QString DiametroExterno() const { return diametroExterno; }
31     QString DiametroInterno() const { return diametroInterno; }
32     QString CoeficientePoisson() const { return coeficientePoisson;
33         }
34     QString CoeficienteExpansaoTermica() const { return
35         coeficienteExpansaoTermica; }
36     QString ModuloElasticidade() const { return moduloElasticidade;
37         }
38     QString PesoUnidade() const { return pesoUnidade; }
39     QString NomeFluido() const { return nomeFluido; }
40     QString Densidade() const { return densidade; }
41     QString Viscosidade() const { return viscosidade; }
42
43     // metodos para definir os dados
44     void Trecho(const QString& valor) { trecho = valor; }
45     void ProfundidadeInicial(const QString& valor) {
46         profundidadeInicial = valor; }
47     void ProfundidadeFinal(const QString& valor) {
48         profundidadeFinal = valor; }
49     void DiametroExterno(const QString& valor) { diametroExterno =
50         valor; }
51     void DiametroInterno(const QString& valor) { diametroInterno =
52         valor; }
53     void CoeficientePoisson(const QString& valor) {
54         coeficientePoisson = valor; }
55     void CoeficienteExpansaoTermica(const QString& valor) {
56         coeficienteExpansaoTermica = valor; }
57     void ModuloElasticidade(const QString& valor) {
58         moduloElasticidade = valor; }

```

```

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
72     // dados do fluido no trecho
73     QString nomeFluido;
74     QString densidade;
75     QString viscosidade;
76};

77
78#endif // CJANELAADICIONARTRECHOTUBULACAO_H

```

Fonte: produzido pelo autor.

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

Listing 7.21: Arquivo de implementação da classe CJanelaAdicionarTrechoTubulacao

```

1 #include "CJanelaAdicionarTrechoTubulacao.h"
2 #include "ui_CJanelaAdicionarTrechoTubulacao.h"
3 #include <QMessageBox>
4
5 // Construtor: inicializa a interface

```

```

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

```

```

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

```

Fonte: produzido pelo autor.

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

Listing 7.22: Arquivo de implementação da classe CJanelaGraficoPressaoHidroestatica

```

1 #ifndef CJANELAGRAFICOPRESSAOHIDROESTATICA_H
2 #define CJANELAGRAFICOPRESSAOHIDROESTATICA_H
3
4 #include <QDialog>
5 #include <vector>
6
7 /*
8 Classe da interface grafica que exibe o grafico de pressao
9 hidrostatica x profundidade
10 Recebe os dados (profundidade e pressao) e plota o grafico usando
11 QCustomPlot
12 Permite exportar o grafico em imagem
13 */
14 namespace Ui {
15 class CJanelaGraficoPressaoHidroestatica;
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     ~CJanelaGraficoPressaoHidroestatica();
27
28     // metodo para atualizar os dados do grafico
29     void PerfilHidrostatico(const std::pair<std::vector<double>,
30                             std::vector<double>& novoPerfil);
31
32 private slots:
33     // acao ao clicar no botao de exportar imagem
34     void on_BtnExportarGrafico_clicked();
35
36
37     // vetores com dados de profundidade e pressao
38     std::vector<double> profundidades;
39     std::vector<double> pressoes;
40
41     // funcao que monta o grafico
42     void PlotarGraficoPressaoxProfundidade();
43 };
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
8 CJanelaGraficoPressaoHidroestatica::

```

```

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

19
20 // Destrutor: libera memoria
21 CJanelaGraficoPressaoHidroestatica::~
22 CJanelaGraficoPressaoHidroestatica()
23 {
24  delete ui;
25 }

26 // Permite atualizar os dados apos a criacao da janela
27 void CJanelaGraficoPressaoHidroestatica::PerfilHidrostatico(
28  const std::pair<std::vector<double>, std::vector<double>>&
   novoPerfil)
29 {
30  profundidades = novoPerfil.first;
31  pressoes = novoPerfil.second;
32  PlotarGraficoPressaoxProfundidade();
33 }

34
35 // Funcao que plota o grafico com base na relacao profundidade x
36 // pressao
37 // Parte da logica para segmentacao por inclinacao foi adaptada com
38 // auxilio de IA (ChatGPT)
39 void CJanelaGraficoPressaoHidroestatica::
40  PlotarGraficoPressaoxProfundidade()
41 {
42  // QVector<double> x(profundidades.begin(), profundidades.end())
43  //
44  QVector<double> x = QVector<double>::fromStdVector(
45    profundidades);
46  // QVector<double> y(pressoes.begin(), pressoes.end());

```

```

42     QVector<double> y = QVector<double>::fromStdVector(pressoes);
43
44     auto *plot = ui->customPlotPressaoMediaProfundidade;
45     plot->clearGraphs();
46
47     QVector<double> trechoX, trechoY;
48
49     // Funcao lambda para comparar inclinacao entre segmentos
50     auto isMesmaInclinacao = [] (double dy1, double dy2, double
51         tolerancia) {
52         return std::abs(dy1 - dy2) < tolerancia;
53     };
54
55     double tolerancia = 1e-2;
56     double inclinacaoAnterior = (y[1] - y[0]) / (x[1] - x[0]);
57
58     trechoX.push_back(x[0]);
59     trechoY.push_back(y[0]);
60
61     for (int i = 1; i < x.size(); ++i) {
62         double inclinacaoAtual = (y[i] - y[i - 1]) / (x[i] - x[i -
63             1]);
64
65         // Se mudou muito a inclinacao, cria um novo trecho no
66         // grafico
67         if (!isMesmaInclinacao(inclinacaoAtual, inclinacaoAnterior,
68             tolerancia)) {
69             int index = plot->graphCount();
70             plot->addGraph();
71             plot->graph(index)->setData(trechoX, trechoY);
72
73             // Uso de cor e estilo dinamico inspirado em sugestao
74             // de IA para melhorar visualizacao
75             QPen pen;
76             pen.setWidth(1);
77             pen.setColor(QColor::fromHsv((index * 70) % 360, 255,
200));
78             plot->graph(index)->setPen(pen);
79
80             QCPScatterStyle estilo(QCPScatterStyle::ssCircle, 4);
81             plot->graph(index)->setScatterStyle(estilo);
82             plot->graph(index)->setLineStyle(QCPGraph::lsLine);

```

```

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

```

```

        AlignRight | Qt::AlignBottom);
118    plot->legend->setBrush(QBrush(Qt::white));
119    plot->legend->setBorderPen(QPen(Qt::gray));
120    plot->legend->setFont(QFont("Arial", 9));
121
122    // Ativacao do zoom, arrastar e selecao com mouse (configuracao
123    // sugerida com base em IA)
124    plot->setInteraction(QCP::iRangeDrag, true);
125    plot->setInteraction(QCP::iRangeZoom, true);
126    plot->setInteraction(QCP::iSelectPlottables, true);
127
128    // Mostra tooltip com coordenadas ao passar o mouse (recurso
129    // inserido com apoio de IA)
130    // ERRO: linha abaixo n o compila
131    // connect(plot, &QCustomPlot::mouseMove, this, [=](QMouseEvent
132    *event) {
133
134        QObject::connect(plot, &QCustomPlot::mouseMove, this, [=](
135            QMouseEvent *event) { // <--- AQUI ESTA A CORRECAO
136            PRINCIPAL
137
138            double xVal = plot->xAxis->pixelToCoord(event->pos().x());
139            double yVal = plot->yAxis->pixelToCoord(event->pos().y());
140
141            // NOTA: Conforme discutido anteriormente, setToolTip pode
142            // n o ser ideal para
143            // tooltip din mico no gr fico QCustomPlot. Avalie se
144            // voc realmente quer
145            // um tooltip no *widget* QCustomPlot ou uma QLabel na
146            // barra de status.
147
148            plot->setToolTip(QString("Profundidade: %1 ft\nPressao: %2
149                         \u00b5psi")
150                           .arg(xVal, 0, 'f', 2)
151                           .arg(yVal, 0, 'f', 2));
152
153        });
154
155        plot->replot();
156    }
157
158    // Botao de exportar imagem do grafico (logica de exportacao por
159    // QPixmap)
160
161 void CJanelaGraficoPressaoHidroestatica::
162     on_BtnExportarGrafico_clicked()
163 {

```

```
148     QString fileName = QFileDialog::getSaveFileName(this, "Salvar  
149         grafico", "", "PNG (*.png);;JPEG (*.jpg)");  
150  
150     if (!fileName.isEmpty()) {  
151         QPixmap imagem = ui->customPlotPressaoMediaProfundidade->  
152             toPixmap(800, 600);  
153         imagem.save(fileName);  
154     }  
154 }
```

Fonte: produzido pelo autor.

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

Listing 7.24: Arquivo de implementação da classe CJanelaMenu

```
1 #ifndef CJANELAMENU_H
2 #define CJANELAMENU_H
3
4 #include <QMainWindow>
5
6 /*
7 Janela principal do programa, que funciona como menu inicial
8 Aqui o usuario pode escolher entre os dois modulos principais do
    software:
9 - Modulo 1: simulacoes de pressao e escoamento
10 - Modulo 2: analise de deslocamentos axiais da tubulacao
11 */
12
13 namespace Ui {
14 class JanelaMenu;
15 }
16
17 class JanelaMenu : public QMainWindow
18 {
19     Q_OBJECT
20
21 public:
22     explicit JanelaMenu(QWidget *parent = nullptr); // construtor
        da janela principal
23     ~JanelaMenu(); // destrutor
24
25 private slots:
26     void on_btnModulo01_clicked(); // acao ao clicar no Modulo 1
27     void on_btnModulo02_clicked(); // acao ao clicar no Modulo 2
```

```
28
29 private:
30     Ui::JanelaMenu *ui;
31 };
32
33 #endif // CJANELAMENU_H
```

Fonte: produzido pelo autor.

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

Listing 7.25: Arquivo de implementação da classe CJanelaMenu

```

32     ui->lbnModulo01->setAttribute(Qt::WA_TransparentForMouseEvents)
33         ; // Label nao atrapalha clique
34
35     // Configuracao da descricao do Modulo 2
36     ui->lbnModulo02->setAlignment(Qt::AlignCenter);
37
38     QString buttonText_02 =
39         "<b>Modulo 02 - Analise de Tensões na Coluna</b><br>"
40         "<ul>"
41         "<li>Carga Axial.</li>"
42         "<li>Colapso e Explosão da Coluna.</li>"
43         "</ul>";
44
45     ui->lbnModulo02->setText(buttonText_02);
46     ui->btnModulo02->setText("");
47     ui->lbnModulo02->setAttribute(Qt::WA_TransparentForMouseEvents)
48         ;
49 }
50
51 // Destruitor da janela principal
52 JanelaMenu::~JanelaMenu()
53 {
54     delete ui;
55 }
56
57 // Quando o usuario clica no Modulo 1, abre a janela de simulacao
58 // reologica
59 void JanelaMenu::on_btnModulo01_clicked()
60 {
61     CSimuladorReologico *w = new CSimuladorReologico(this);
62     w->setWindowTitle("SEAPEP - Software Educacional de Engenharia"
63                         " de Po o");
64     w->show();
65 }
66
67 // Quando o usuario clica no Modulo 2, abre a janela de simulacao
68 // de perda na tubulacao
69 void JanelaMenu::on_btnModulo02_clicked()
70 {
71     CSimuladorPerdaTubulacao *w = new CSimuladorPerdaTubulacao(this
72                     );
73     w->setWindowTitle("SEAPEP - Software Educacional de Engenharia"

```

```
    de_□ P o  o " ) ;  
68      w -> show ( ) ;  
69 }
```

Fonte: produzido pelo autor.

Capítulo 8

Resultados

Neste capítulo, serão apresentados a validação e os resultados do software. Serão aplicados os conceitos descritos no capítulo de Metodologia.

Inicialmente, o software será validado por meio de comparações com cálculos realizados manualmente, a fim de verificar a precisão dos resultados fornecidos pelo código.

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

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

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

8.1 Metodologia de Validação do Software

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

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

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

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

8.2 Casos de Teste

Os casos de teste visam abranger diversos cenários de uso do software.

8.3 Validação da pressão Hidroestática

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

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

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

Resposta esperada: 136 psig

$$P_H = 0.052\rho L - P_{formação}$$

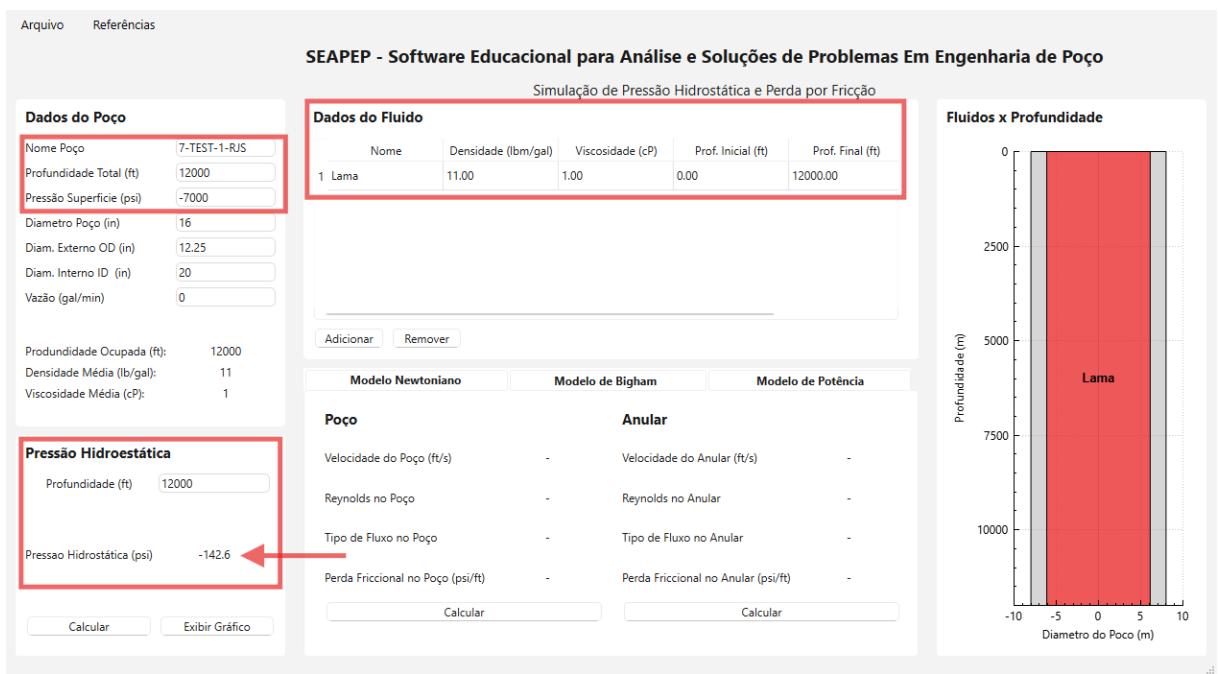
Onde: $\rho=11$ lbm/gal (densidade do fluido); $L=12.000$ ft (profundidade total); $P_{formação}=7000$ ft (pressão da formação);

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

$$P_H = -136psig$$

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

Figura 8.1: Solução gerada para cálculo da pressão hidrostática



Fonte: Produzido pelo autor.

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

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

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

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

Resposta esperada: turbulento

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

Resposta esperada: 51,3 psi/1.000 ft

c. Determine o regime de escoamento no anular.

Resposta esperada: turbulento

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

Resposta esperada: 72,9 psi/1.000 ft

Cálculos no tubo

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

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

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

Onde: ρ = densidade; μ = viscosidade; N_{re} (Número de Reynolds);

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

Onde: dp_f (Perda Friccional);

Cálculos no anular

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

Onde: v (Velocidade do Poço); q (vazão); d_2 (diâmetro do Poço); d_1 (diâmetro Externo);

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

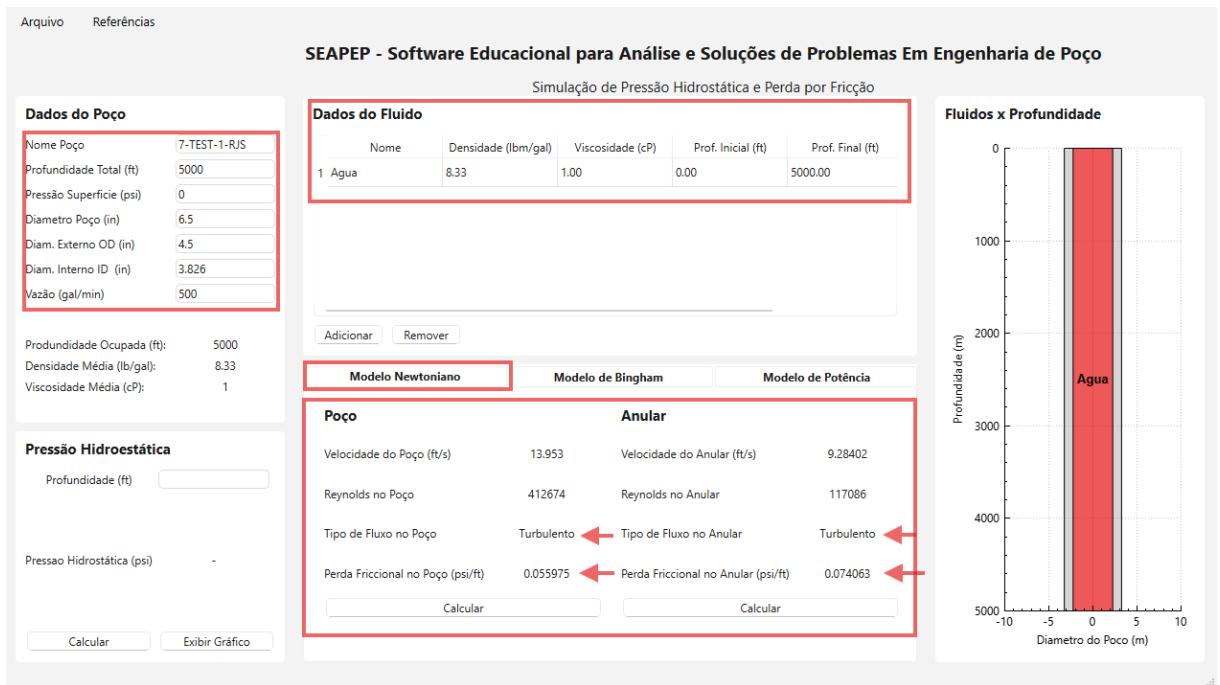
Onde: ρ = densidade; μ = viscosidade; N_{re} (Número de Reynolds);

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

Onde: dp_f (Perda Friccional);

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

Figura 8.2: Soluções geradas pelo código para modelo Newtoniano no poço e anular



Fonte: Produzido pelo autor.

8.5 Validação da perda de fricção pelo modelo *Binghan* no Poço e Anular

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

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

Resposta esperada:

Regime de escoamento: turbulento

Perda de pressão no tubo: 132 psi/1.000 ft

Perda de pressão no anular: 185 psi/1.000 ft

Cálculos no tubo

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

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

$$N_{rec} = 5000 (fig.4.33, [APPLIED1991])$$

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

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

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

Cálculos no anular

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

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

$$N_{rec} = 3000 (fig.4.33, [APPLIED1991])$$

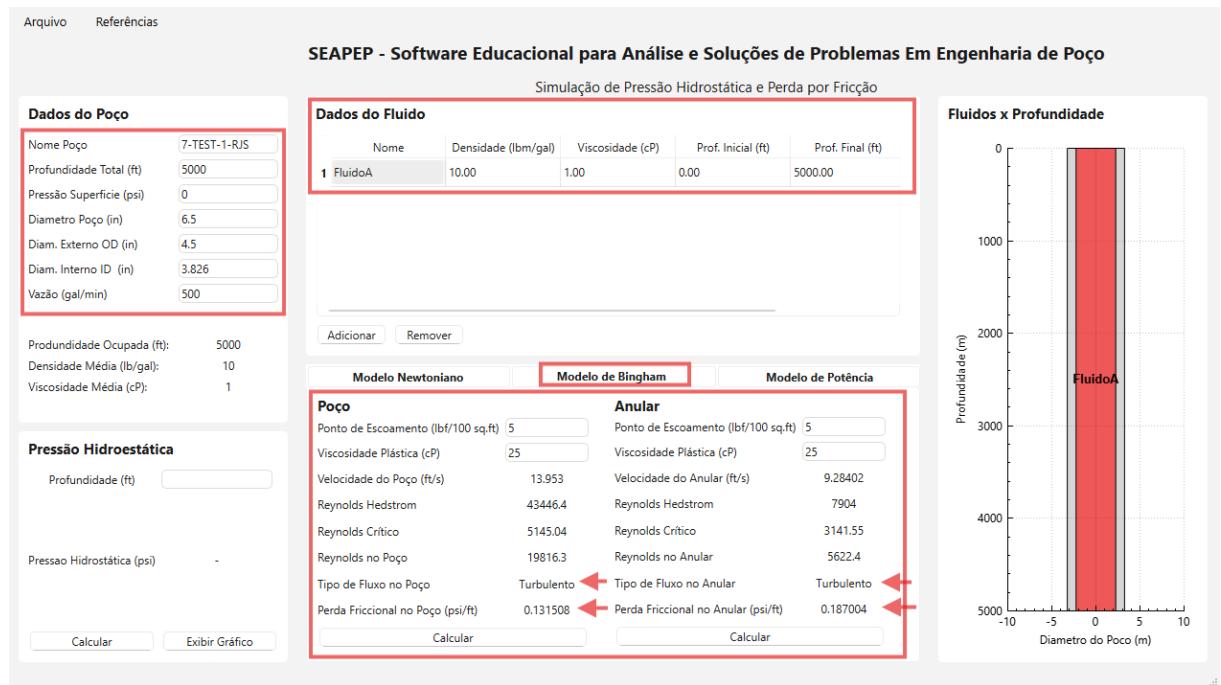
$$N_{re} = \frac{757\rho\bar{v}(d_2 - d_1)}{\mu_p} = \frac{757(10)(9.284)(6.5 - 4.5)}{25} = 5622.39$$

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

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

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

Figura 8.3: Soluções geradas pelo código para modelo de Bingham no poço e anular



Fonte: Produzido pelo autor.

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

Neste subtópico, valida-se o modelo reológico Lei da Potência para cálculo da perda de carga por fricção em fluidos não-newtonianos, aplicando os mesmos parâmetros geométricos do exercício 4.40. Utiliza-se o exercício 4.42 do livro Applied Drilling Engineering como referência, alterando-se apenas as propriedades reológicas do fluido para refletir um comportamento pseudoplástico.

Resolva o Exercício 4.40 para um fluido do tipo potência com densidade de 12 lbm/gal, índice de comportamento de fluxo (n) igual a 0,75 e índice de consistência (K) igual a 200 eq cP. (traduzido, Jr. et al. (1991))

Resposta esperada:

Regime de escoamento no tubo: turbulento

Perda de pressão no tubo: 144 psi/1.000 ft

Regime no anular: turbulento

Perda de pressão no anular: 205 psi/1.000 ft

Cálculos no tubo

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

$$N_{re} = \frac{89100\rho\bar{v}^{2-n}}{k} \left(\frac{0.0416d}{3+1/n} \right)^n = \frac{89100(12)(13.95)^{2-0.75}}{200} \left(\frac{0.0416(3.826)}{3+1/0.75} \right)^{0.75} = 12092.29$$

$$N_{rec} = 3500 (fig.4.33, [APPLIED1991])$$

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

$$f = 0.006 (fig.4.34, [APPLIED1991])$$

$$\frac{dp_f}{dL} = \frac{f\rho\bar{v}^2}{25.8d} = \frac{0.006(12)(13.95)^2}{25.8(3.826)} = 0.1419 \text{ psi/ft}$$

Cálculos no anular

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

$$N_{re} = \frac{109000\rho\bar{v}^{2-n}}{k} \left(\frac{0.0208(d_2 - d_1)}{2+1/n} \right)^n = \frac{109000(12)(9.284)^{2-0.75}}{200} \left(\frac{0.0208(6.5 - 4.5)}{2+1/0.75} \right)^{0.75} = 3957.37$$

$$N_{rec} = 2500 (fig.4.33, [APPLIED1991])$$

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

$$f = 0.008 (fig.4.34, [APPLIED1991])$$

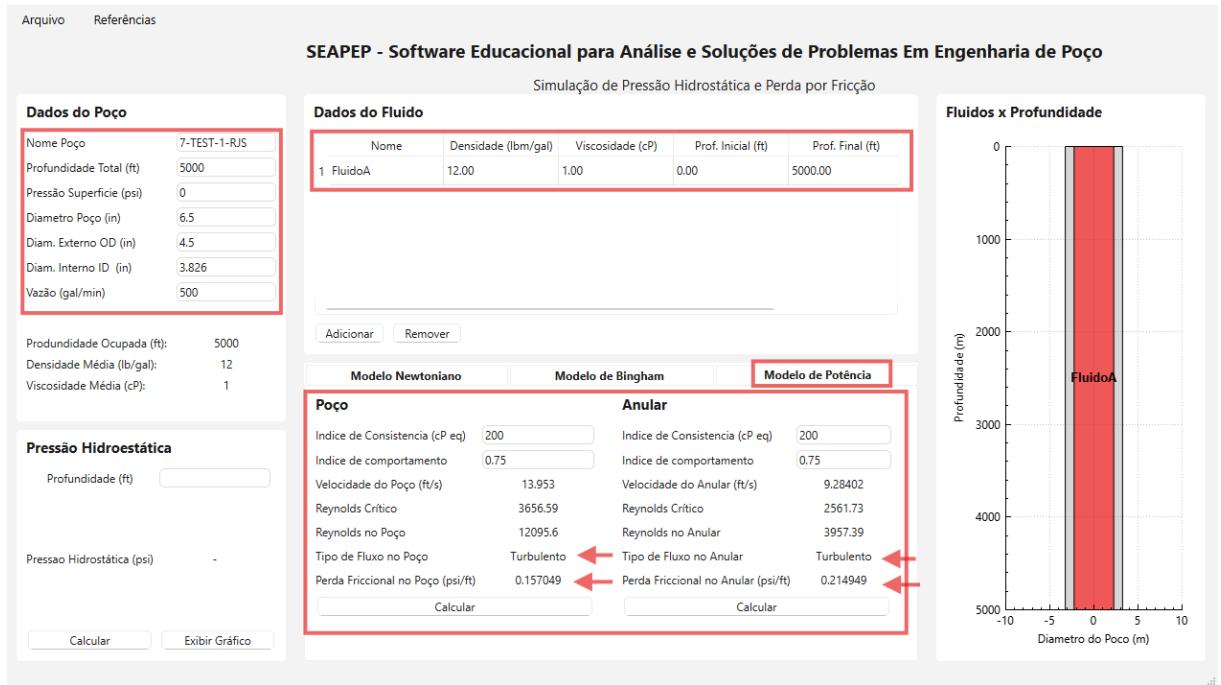
$$\frac{dp_f}{dL} = \frac{f\rho\bar{v}^2}{21.1(d_2 - d_1)} = \frac{0.008(12)(9.284)^2}{21.1(6.5 - 4.5)} = 0.196 \text{ psi/ft}$$

O teste com o modelo da lei da potência apresentou resultados bastante próximos aos esperados na literatura, validando tanto o cálculo do número de Reynolds quanto a perda de pressão por fricção para um fluido real.

Os resultados não são completamente idênticos, pois na resolução analítica é necessário consultar tabelas com escala logarítmica, o que pode introduzir erros de leitura ou interpolação manual. O software, por outro lado, obtém esses valores por meio de aproximações lineares, baseadas nos principais pontos da curva log-log, utilizando um método de regressão linear para melhor ajuste. A ferramenta demonstrou, como ilustrado na

Figura 8.4, capacidade confiável de tratar modelos reológicos mais complexos, indo além dos comportamentos já validados nos modelos anteriores.

Figura 8.4: Soluções geradas pelo código para modelo de lei de Potência no poço e anular



Fonte: Produzido pelo autor.

8.7 Validação da variação do comprimento do tubo devido a força pistão

Neste subtópico, valida-se o módulo do software que calcula a variação do comprimento axial do tubo causada pela força pistão, efeito resultante da diferença de área entre o tubo e o packer em presença de variação de pressão interna e externa. Para validar os modelos de variação axial implementados no software, foi utilizado um cenário adaptado de exercício aplicado em sala de aula.

A completação analisada contém um packer instalado a 8.000 ft de profundidade. O tubo é livre para se mover e possui diâmetro externo de 4,5 polegadas e diâmetro interno de 3,862 polegadas, enquanto o packer apresenta diâmetro externo de 5,0 polegadas. Inicialmente, o poço estava preenchido com fluido de densidade igual a 10 lb/gal. A temperatura na superfície foi mantida constante em 80 °F, enquanto no fundo variou de 200 °F para 220 °F entre a condição inicial e final. O tubo apresenta módulo de elasticidade igual a 10^6 psi, coeficiente de Poisson de 0,3, peso por unidade de comprimento de 17,0 lb/ft e coeficiente de expansão térmica igual a $7,0 \times 10^{-6} \text{ } ^\circ\text{F}^{-1}$.

Resposta esperada: 3ft

$$\Delta L = \frac{\Delta F \cdot L}{E \cdot A_s}$$

Onde :

$$\Delta F = \Delta P_{in} A_{in} - \Delta P_{out} A_{out}$$

$$\Delta P_{in} = [0.052(10)(8000) + 2000] - [0.052(10)(8000)] = 2000 \text{ psi}$$

$$\Delta P_{out} = 0 \text{ psi}$$

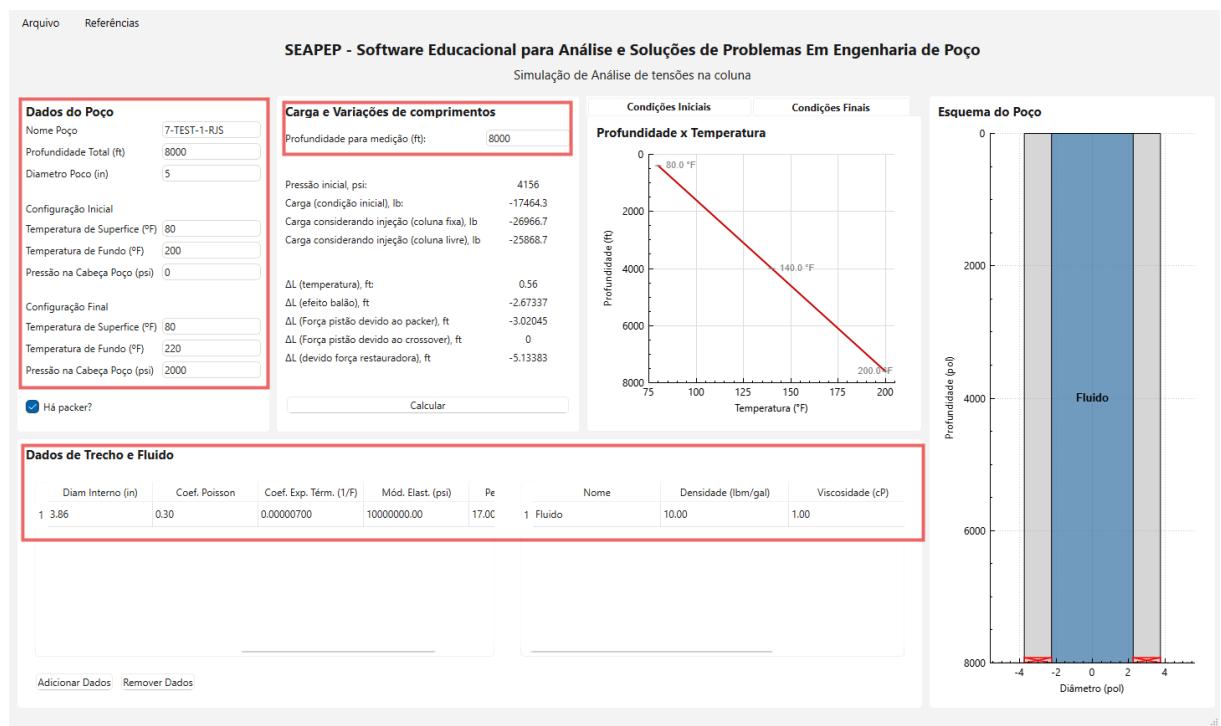
$$A_{in} = \frac{\pi}{4}(d_2^2 - d_1^2) = \frac{\pi}{4}(5^2 - 3.862^2) = 7.92 \text{ in}^2$$

$$\Delta F = 2000(7.92) - 0 \cdot A_{out} = -15840 \text{ lb}$$

$$\Delta L = \frac{\Delta F \cdot L}{E \cdot A_s} = \frac{-15840(8000)}{10 \cdot 10^6 [\frac{\pi}{4}(4.5^2 - 3.862^2)]} == 3.04 \text{ ft}$$

A simulação realizada no software resultou em uma variação de comprimento compatível com o esperado teoricamente. A deformação foi calculada utilizando a equação de carga axial para efeito pistão, considerando o diferencial de área e a variação de pressão induzida pela substituição do fluido.

Figura 8.5: Solução gerada para variação do comprimento todo devido a força pistão



Fonte: Produzido pelo autor.

8.8 Validação da variação do comprimento do tubo devido ao efeito balão

Este subtópico trata da validação do efeito balão, fenômeno associado à deformação radial e longitudinal do tubo sob diferentes pressões interna e externa.

Utilizando o mesmo cenário descrito no item anterior, o efeito balão foi avaliado com base na mudança de pressão devido à substituição do fluido no tubo e no anular. A diferença de pressão provoca a expansão (ou contração) do tubo, afetando seu comprimento axial.

Resposta esperada: -2.6ft

$$\Delta L_b = \frac{2.v.\Delta F.L}{E.A_s}$$

Utilizandos os dados já calculados na validação acima:

$$\Delta F = \Delta P_{in}A_{in} - \Delta P_{out}A_{out}$$

$$\Delta P_{in} = 2000\text{psi}$$

$$\Delta P_{out} = 0\text{psi}$$

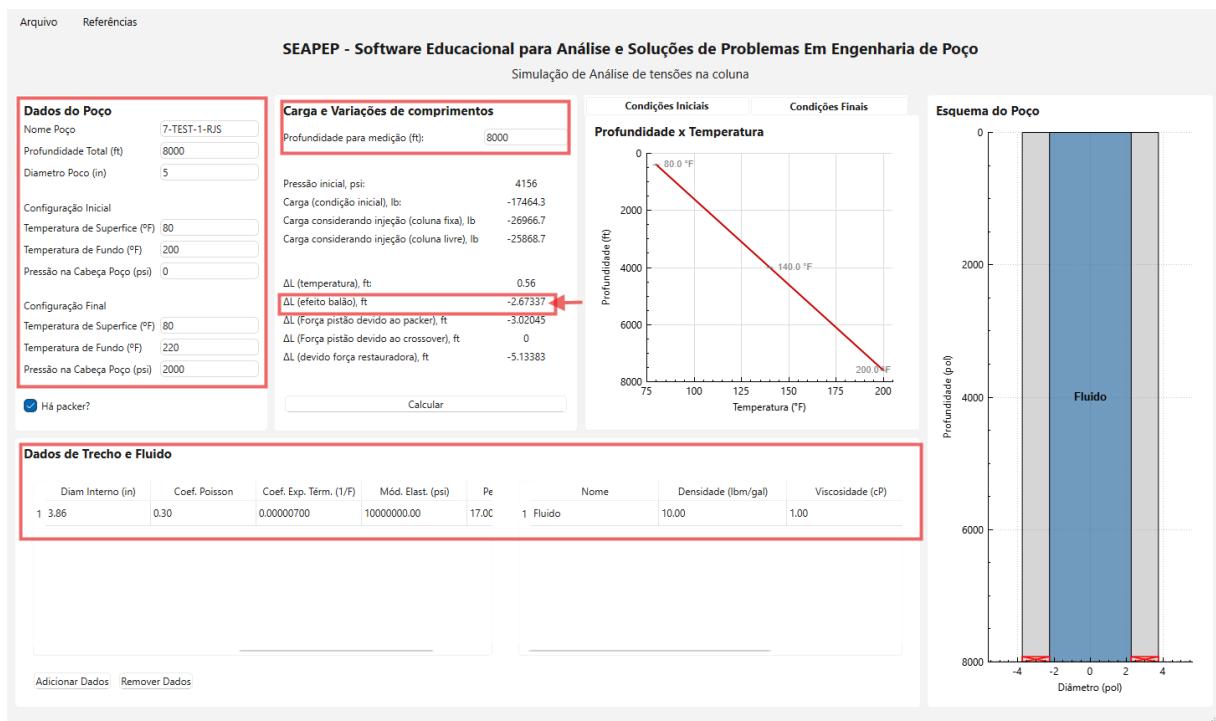
$$A_{in} = \frac{\pi}{4}(d_2^2 - d_1^2) = \frac{\pi}{4}(3.862^2) = 11.71\text{in}^2$$

$$\Delta F = 2000(11.71) - 0.A_{out} = -23428.49\text{lb}$$

$$\Delta L_b = \frac{-2(0.3)(23428.49)(8000)}{10.10^6[\frac{\pi}{4}(4.5^2 - 3.862^2)]} = -2.68\text{ft}$$

O software foi capaz de calcular a deformação com base nas pressões associadas à troca dos fluidos, e os resultados mostraram boa concordância com a equação clássica para efeito balão.

Figura 8.6: Solução gerada para variação do comprimento todo devido ao efeito balão



8.9 Validação da variação do comprimento do tubo devido ao efeito da temperatura

este subtópico, o software é validado quanto à dilatação térmica do tubo em função da variação de temperatura entre a configuração inicial e final do poço.

Utilizando o mesmo cenário descrito no item anterior.

Resposta esperada: 0.5ft

$$\Delta L_t = C_t \cdot L \cdot \Delta T$$

onde:

$$\bar{T}_{inicial} = \frac{80 + 200}{2} = 140^{\circ}F$$

$$\bar{T}_{final} = \frac{80 + 220}{2} = 150^{\circ}F$$

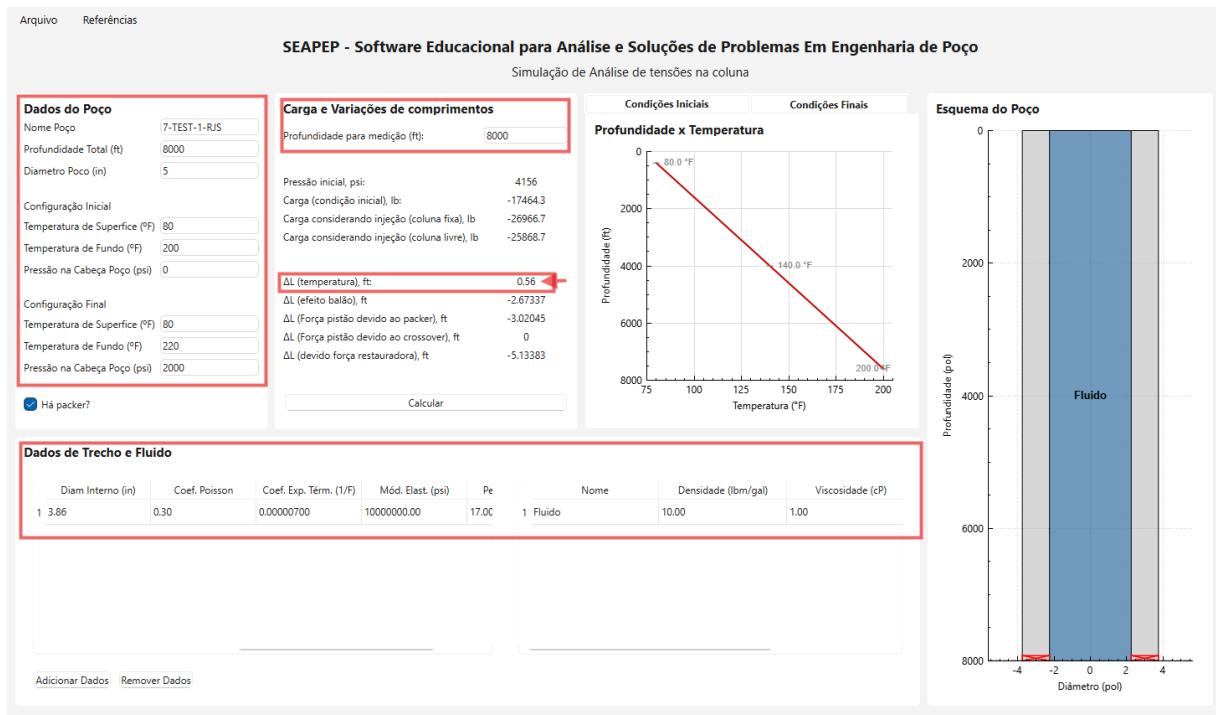
$$\Delta T = 150 - 140 = 10^{\circ}F$$

Logo:

$$\Delta L_t = (7 \cdot 10^{-6})(8000)(10) = 0.56 ft$$

A simulação indicou uma dilatação térmica compatível com os valores obtidos pela fórmula, considerando a interpolação linear da temperatura entre o fundo e a superfície.

Figura 8.7: Solução gerada para variação do comprimento todo devido ao efeito da temperatura



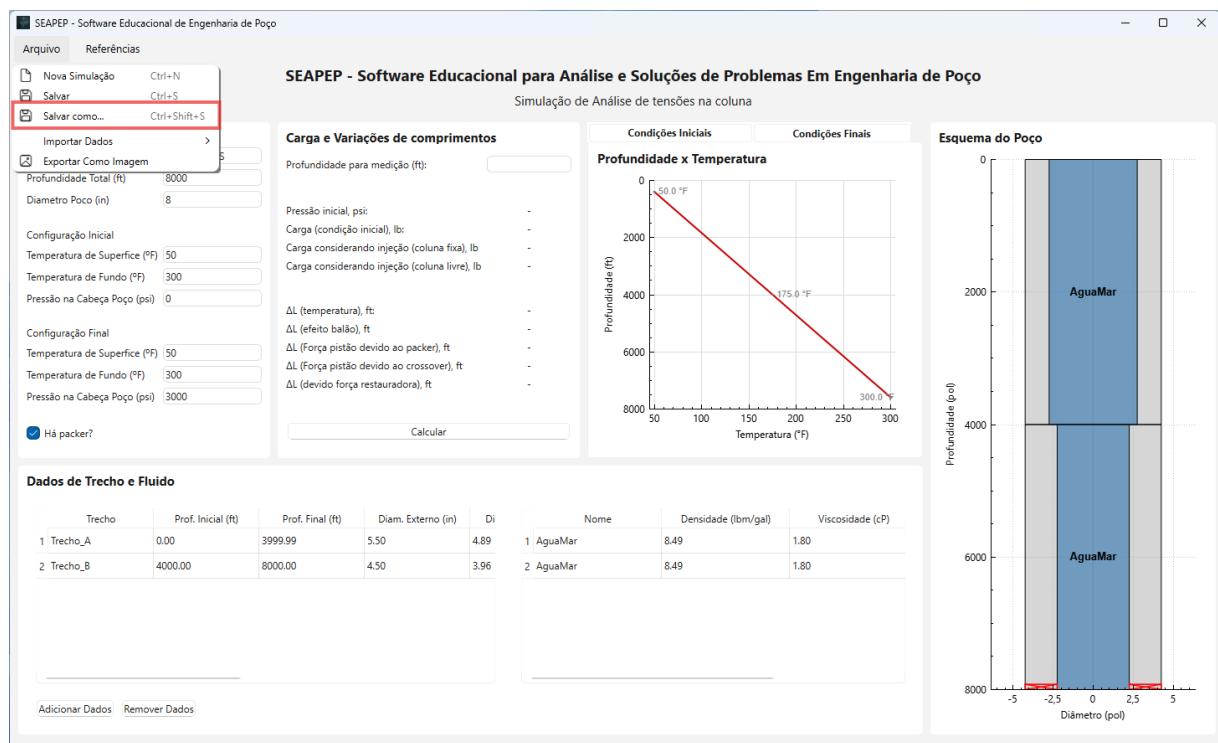
Fonte: Produzido pelo autor.

8.10 Validação da importação e exportação do documento (“salvar como...”)

A funcionalidade de importação e exportação de arquivos foi validada por meio da opção “salvar como...” presente na interface do software, que permite ao usuário armazenar as configurações completas do poço e dos fluidos em arquivos com extensão .dat. Essa funcionalidade garante que os dados possam ser salvos, compartilhados e reabertos posteriormente com total fidelidade.

Durante os testes, foram preenchidos manualmente diferentes conjuntos de dados, incluindo profundidade total do poço, pressão e temperatura nas condições inicial e final, presença de packer, além das propriedades de cada trecho da coluna e das características dos fluidos envolvidos. Após o preenchimento, os dados foram exportados para arquivos .dat.

Figura 8.8: Opção de Salvar



Fonte: Produzido pelo autor.

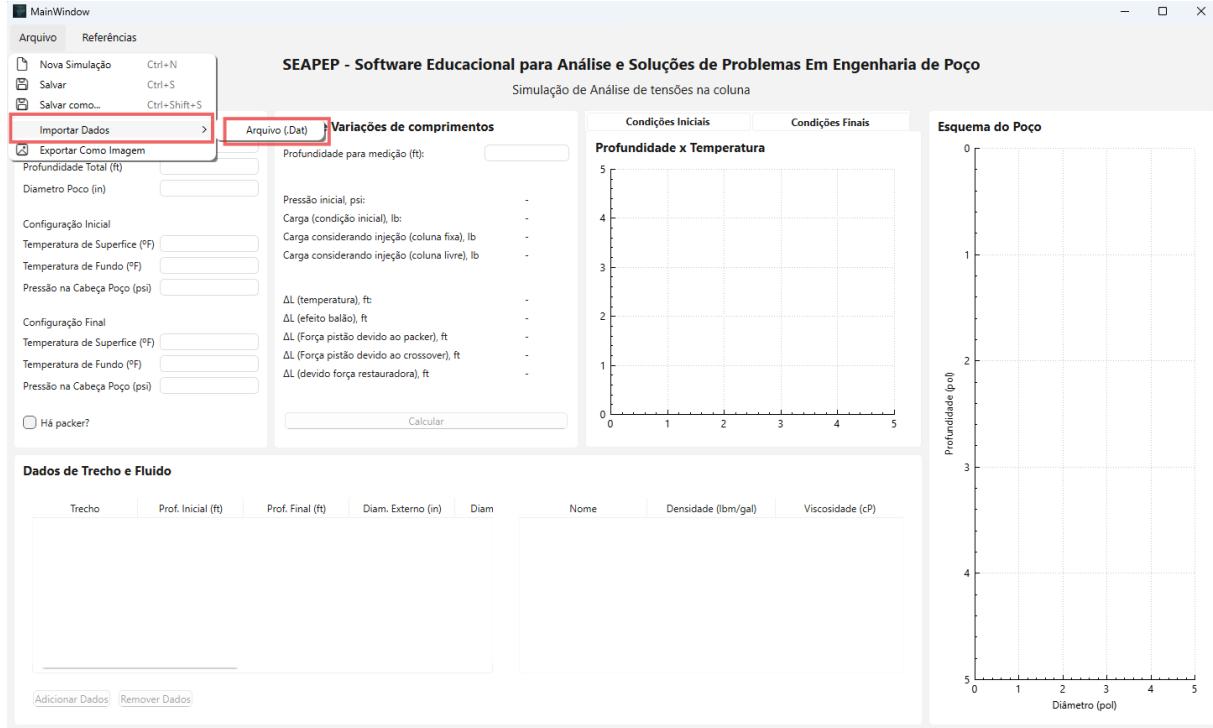
Figura 8.9: Modelo do Arquivo .dat

# Configurações do Poço	Profundidade (ft)	Diametro do Poco (ft)	Pressao Sup. Inicial (psi)	Pressao Sup. Final (psi)	Temp Sup. Inicial (°F)	Temp Fund. Inicial (°F)	Temp Sup. Final (°F)	Temp Fund. Final (°F)		
# Nome Packer (ft) 7-TE5-01-R35 true	8000	8	0	3000	50	300	50	300		
# Nome Trecho	Prof. Inicial (ft)	Prof. Final (ft)	Diam. interno (in)	Coef. Polisson	Coef. Exp. Term. (1/F)	Mod. Elast. (psi)	Peso/unid (lb/ft)	Nome fluido	Densidade (lbm/gal)	Viscosidade (cP)
Trecho_A Trecho_B	0.00 4000.00	3999.99 8000.00	5.50 4.50	0.30 0.30	0.00003000 0.00003000	30000000.00 35000000.00	22.00 15.00	AquaMar AquaMar	8.49 8.49	1.80 1.80

Fonte: Produzido pelo autor.

Posteriormente, os arquivos foram reimportados utilizando a opção de carregamento automático do software. Ao reabrir o arquivo salvo, todas as informações foram recuperadas com exatidão, sem perdas numéricas ou inconsistências nos campos de entrada.

Figura 8.10: Opção de importação de arquivos no software



Fonte: Produzido pelo autor.

Essa validação confirma que a rotina de leitura e escrita está corretamente implementada, permitindo que os usuários arquivem projetos, compartilhem simulações e retomem configurações com segurança. Além disso, essa funcionalidade viabiliza a reproduibilidade dos testes apresentados neste capítulo, já que os arquivos .dat utilizados encontram-se disponíveis nas pastas de documentação do projeto.

8.11 Conclusão

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

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

Além das simulações, foi verificado o correto funcionamento das funcionalidades de importação e exportação de dados por arquivos .dat, garantindo reproduzibilidade, rastreabilidade e usabilidade da ferramenta em ambientes acadêmicos e operacionais.

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

Capítulo 9

Documentação

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

9.1 Documentação do usuário

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

9.2 Documentação do desenvolvedor

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

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

Na Figura 9.1 mostra uma imagem da documentação gerada.

Figura 9.1: Logo e documentação do *software*

SoftwareEducacionalParaAnaliseESolucaoDeProblemasEmEngenhariaDePoco

The screenshot shows a web-based documentation interface for a software project. At the top, there's a navigation bar with links for 'Main Page', 'Classes', and 'Files'. A search bar is also present. Below the navigation, a section titled 'File List' displays a list of files with brief descriptions. The list includes:

- CAuxiliar.cpp
- CAuxiliar.h
- CFluido.cpp
- CFluido.h
- CModeloBingham.cpp
- CModeloBingham.h
- CModeloNewtoniano.cpp
- CModeloNewtoniano.h
- CModeloPotencia.cpp
- CModeloPotencia.h
- CModeloReologico.cpp
- CModeloReologico.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

The screenshot shows the source code for the `CSimuladorPoco` class. The code is presented in a monospaced font with line numbers on the left. The code includes header file includes, a protected section with pointers to other classes, and a public section with constructor/destructor definitions and menu-related functions.

```
1 #ifndef CSIMULADORPOCO_H
2 #define CSIMULADORPOCO_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 public:
22     // Construtor e destrutor
23     CSimuladorPoco() {}
24     ~CSimuladorPoco() {}
25
26     // Menus principais
27     void MenuPrincipal();
28     void MenuConfigurarSimulador();
29     void MenuPressaoHidrostatica();
30     void MenuPerdaDeCarga();
```

Fonte: Produzido pelo autor.

Capítulo 10

Manual do desenvolvedor

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 obrigatórias para rodar o software

Compilador

- g++ (GNU C++ Compiler)
 - Site oficial: Site
 - Instalação no Linux (Fedora/RHEL): SUDO DNF INSTALL GCC-C++
 - Instalação no Ubuntu/Debian: SUDO APT INSTALL G++

Qt Framework

- Qt 5 ou Qt 6 (com Qt Widgets)
 - Inclui: QAPPLICATION, QFILEDIALOG, QMESSAGEBOX, QMAINWINDOW, QDIALOG, QSTRING, QTABLEWIDGETITEM, QMAP, QCOLOR, QDESKTOPSERVICES, QURL, QDEBUG
 - Download: Site
 - Instale preferencialmente via Qt Online Installer ou via apt: SUDO APT INSTALL QTBASE5-DEV QTTOOLS5-DEV-TOOLS

QCustomPlot

- Biblioteca para gráficos customizados em Qt.

- Site oficial: Site
- Instruções de uso:
 - * Baixe o arquivo QCUSTOMPLOT.H e QCUSTOMPLOT.CPP e adicione ao seu projeto Qt.
 - * Inclua normalmente: `#INCLUDE "QCUSTOMPLOT.H"`

10.1.2 Bibliotecas padrão da linguagem C++ (não precisam ser instaladas separadamente)

Essas bibliotecas fazem parte da STL (Standard Template Library) e da linguagem C++ padrão:

- `<iostream>` – entrada/saída padrão
- `<fstream>` – leitura e escrita em arquivos
- `<sstream>` – manipulação de strings como fluxos
- `<iomanip>` – controle de formatação de saída
- `<cmath>` e `<math.h>` – funções matemáticas
- `<algorithm>` – funções de ordenação e busca
- `<vector>` – estrutura de vetor dinâmico
- `<memory>` – ponteiros inteligentes (`std::unique_ptr`)
- `<string>` – manipulação de strings
- `<cstdlib>` – utilitários de C como `std::exit`
- `<numbers>` – constantes matemáticas (disponível em C++20 ou superior)]

10.1.3 Observações importantes

- Certifique-se de que o seu projeto está configurado para usar C++20 (ou no mínimo C++17), especialmente para `<numbers>`.

10.1.4 Execução sem compilador

Se o usuário não deseja instalar dependências, basta:

- Rodar o executável pré-compilado que está disponível nas pastas TEST ou RELEASE.

10.2 Interface gráfica

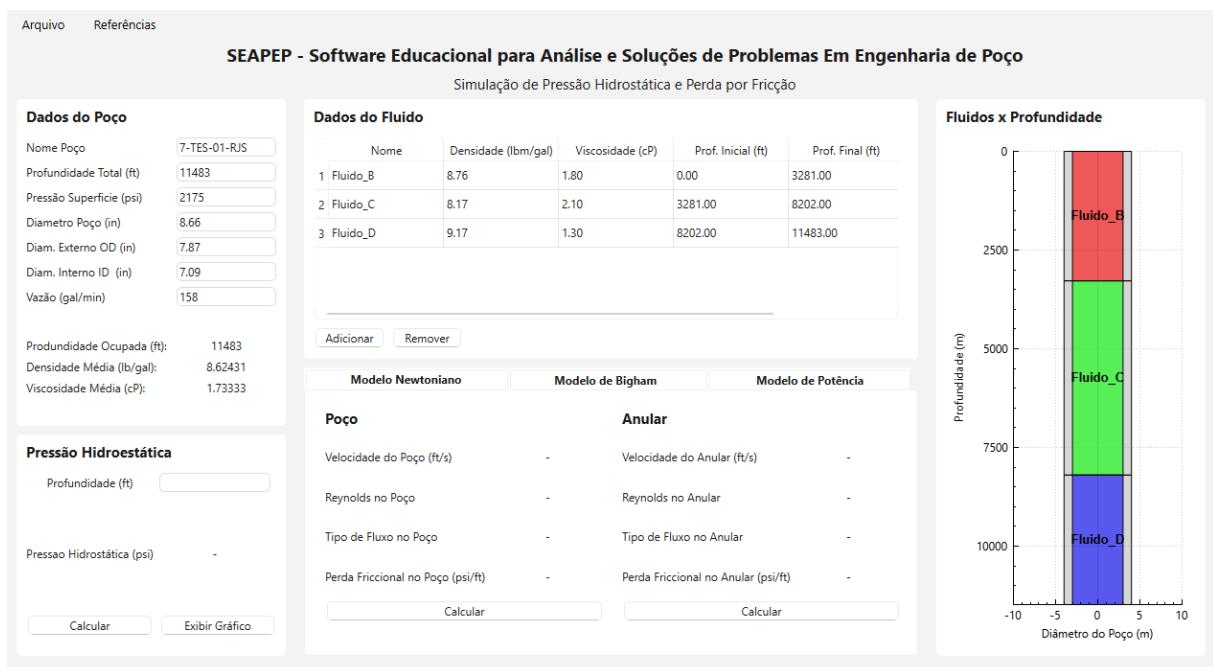
A interface do programa é apresentada nas Figuras: 10.1, 10.2 e 10.3.

Figura 10.1: Versão 1.1, Menu de interface do software



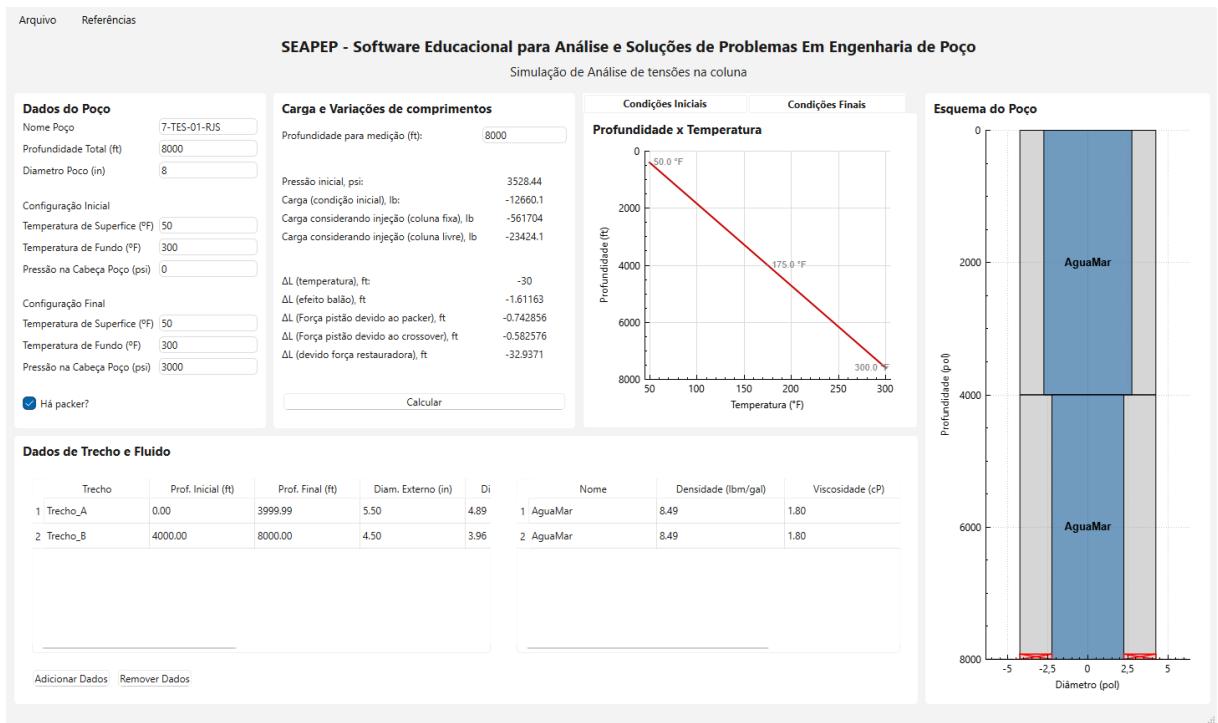
Fonte: Produzido pelo autor.

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.

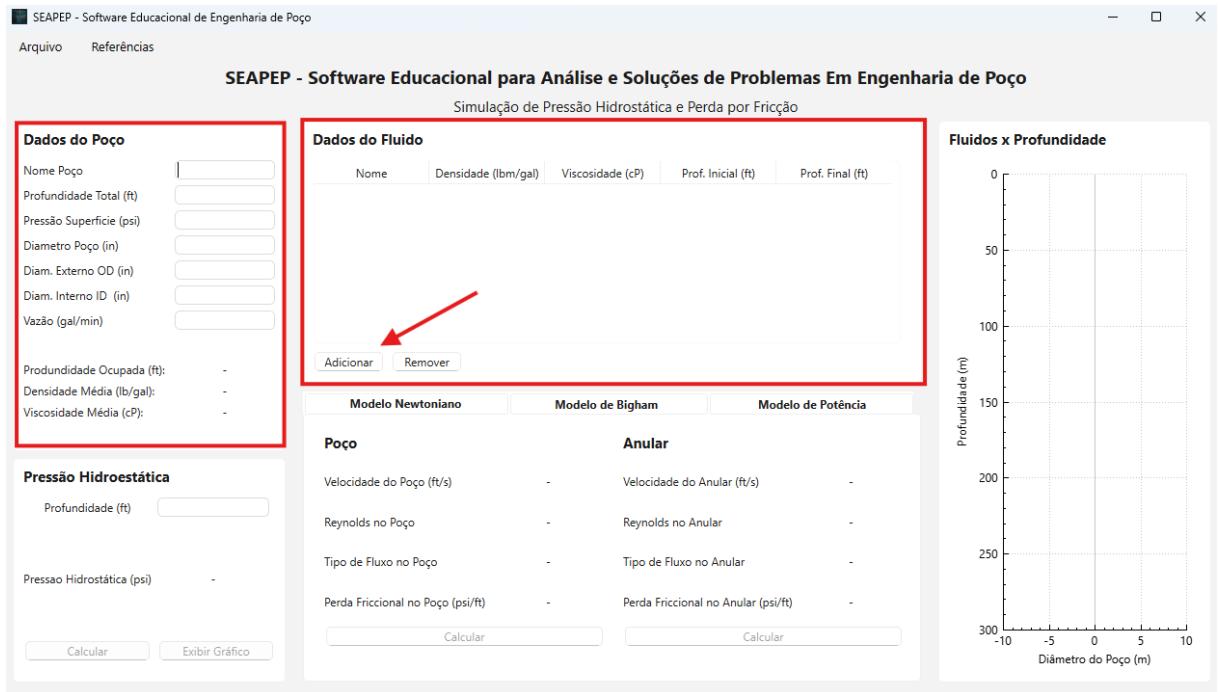
Configuração Manual via Interface Gráfica

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

The screenshot shows a Windows Notepad window with the title bar 'ArquivoPoco.dat'. The menu bar includes 'Arquivo', 'Editar', and 'Exibir'. The status bar at the bottom indicates 'Ln 10, Col 87 | 775 caracteres' and encoding 'UTF-8'. The main content area contains two sections of configuration data:

```
# 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|
```

Fonte: Produzido pelo autor.

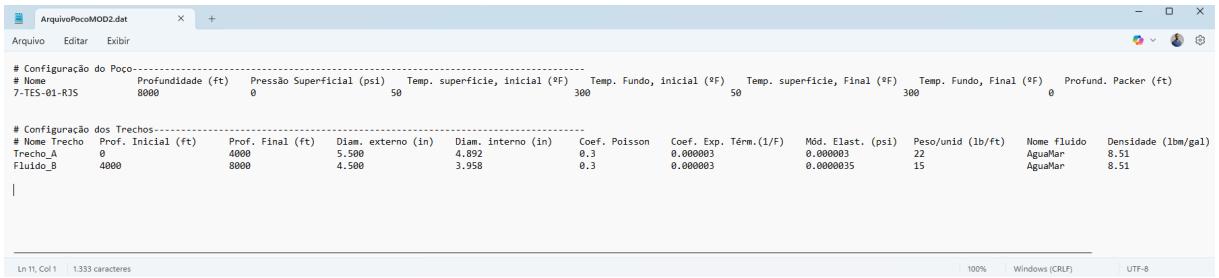
Dessa forma após configurar o poço o usuário poderá explorar as funcionalidade do software descritas na Seção 10.3.

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 with the title "ArquivoPocoMOD2.dat". The menu bar includes "Arquivo", "Editar", and "Exibir". The window contains two sections of configuration data:

```
# Configuração do Poço-----
# 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-R3S      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. Térmt.(1/F)  Mód. 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             Aguamar     8.51
```

At the bottom of the window, status bars show "Ln 11, Col 1 1.333 caracteres", "100% Windows (CRLF)", and "UTF-8".

Fonte: Produzido pelo autor.

Referências Bibliográficas

- BUENO, André Duarte. 2003. *Programação orientada a objeto com c++*. Novatec. 42
- 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, 152, 153, 154, 156, 158
- Mitchell, Robert F., & Miska, Stefam Z. 2011. *Fundamentals of drilling engineering*. Society of Petroleum Engineers. 17, 32, 33, 34, 38, 39, 40