

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
Junho - 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
Junho - 2025

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

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.

Aprovada em 27 de Junho de 2025.

Comissão Examinadora:

Prof. José Ricardo Siqueira (D.Sc) - LAMET/UENF

Prof. Bruno José Vicente (D.Sc) - LAMET/UENF

Prof. André Duarte Bueno (D.Sc) - LENEP/UENF

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

Este trabalho apresenta o desenvolvimento de um software educacional voltado para estudantes de Engenharia de Petróleo e áreas correlatas, com foco na assimilação prática dos principais conceitos da disciplina de Engenharia de Poços. O sistema, concebido com uma interface gráfica intuitiva e recursos interativos, visa auxiliar o aprendizado de tópicos complexos por meio de ferramentas de visualização, análise, simulação e cálculo, abrangendo conteúdos-chave da ementa da disciplina LEP01353 - Engenharia de Poço, ministrada no Laboratório de Engenharia e Exploração de Petróleo (LENEP/CCT/UENF) desde o período letivo de 2024/01. O software foi projetado para ser acessível tanto a discentes vinculados à disciplina quanto a outros interessados, inclusive de instituições externas, reforçando seu caráter inclusivo e educativo. O processo de desenvolvimento foi conduzido com base em metodologias ágeis e no uso de sistemas de controle de versão como Git e GitHub, práticas comuns no ambiente profissional e acadêmico contemporâneo. Durante as fases de testes, o sistema foi aplicado em diversos cenários de simulação, como o estudo de perfis térmicos, cálculo de pressão hidrostática, análise de perda de carga em tubulações e anulares, modelagem reológica de fluidos e avaliação do efeito pistão em completações com e sem packer. Os resultados obtidos demonstraram que o software é capaz de representar com fidelidade os fenômenos envolvidos, facilitando a compreensão dos mecanismos físicos e operacionais típicos da engenharia de poços. Além disso, os testes apontaram boa estabilidade computacional e respostas coerentes com os parâmetros de entrada fornecidos. A proposta da ferramenta educacional mostrou-se eficaz tanto no apoio a aulas práticas e teóricas quanto no estímulo à autonomia dos alunos na exploração de variáveis operacionais, consolidando-se como um recurso didático inovador e de alta relevância no processo de ensino-aprendizagem. Conclui-se, portanto, que o projeto atingiu seus objetivos iniciais, oferecendo uma solução tecnológica viável, didaticamente rica e academicamente alinhada às necessidades da formação em engenharia de petróleo.

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

Abstract

This work presents the development of an educational software aimed at students of Petroleum Engineering and related fields, focusing on the practical understanding of key concepts covered in the Well Engineering course. The system was designed with an intuitive graphical interface and interactive tools to support the learning process through modules of visualization, analysis, simulation, and calculation, addressing essential topics from the syllabus of course LEP01353 – Well Engineering, taught at the Petroleum Engineering and Exploration Laboratory (LENEP/CCT/UENF) since the 2024/01 academic term. The software is intended to be accessible not only to students officially enrolled in the course but also to other interested users, regardless of institutional affiliation, thereby reinforcing its inclusive and educational nature. The development process followed agile methodologies and utilized Git/GitHub for version control, aligning with widely adopted practices in both academia and industry. During testing phases, the software was applied to various simulation scenarios, including thermal profiling, hydrostatic pressure calculations, frictional pressure loss in pipes and annuli, rheological modeling of drilling fluids, and piston effect analysis in well completions with and without packers. The results demonstrated that the system is capable of accurately representing the relevant physical and operational phenomena, facilitating a deeper understanding of well engineering mechanisms. Additionally, simulations confirmed the tool's computational stability and consistency with input parameters. The educational platform proved to be effective in supporting both theoretical and practical instruction, while also encouraging student autonomy in the exploration of operational variables. In conclusion, the project successfully achieved its objectives, offering a viable technological solution that is pedagogically valuable and well-aligned with the academic demands of petroleum engineering education.

Keywords: Educational simulator. Petroleum Engineering education. Simulation. Visualization and analysis. Well Engineering.

Listas de Figuras

2.1	Etapas para o desenvolvimento do software - <i>projeto de engenharia</i>	19
3.1	Diagrama de caso de uso – caso de uso geral	24
3.2	Diagrama de caso de uso específico: “calcular pressão hidrostática”	25
3.3	Diagrama de caso de uso específico - cálculo da perda de pressão devido a perda por fricção no poço e no anular	25
4.1	Diagrama de corpo livre, atuação de forças em um elemento de fluido	32
4.2	Coluna composta por fluidos com diferentes características	34
4.3	Fluxo laminar de fluido Newtoniano	36
4.4	Diagrama de pacotes	45
5.1	Diagrama de classes simplificado	47
5.2	Diagrama de classes	48
5.3	Diagrama de sequência - Módulo reológico	51
5.4	Diagrama de comunicação para caso do cálculo usando modelo reológico . .	52
5.5	Diagrama de máquina de estado do objeto CSIMULADORPERDATUBULACAO	53
5.6	Diagrama de atividades: CObjetoPoco::Carga(double profundidade, bool inicio)	54
6.1	Diagrama de componentes	57
7.1	Versão 1.0, interface do software	59
7.2	Versão 1.1, interface do software	60
7.3	Versão 2.0, interface do software	62
7.4	Versão 2.6, interface do software	63
7.5	Versão 3.0, Menu de interface do software	65
7.6	Versão 3.0, interface do módulo 01 do software	65
7.7	Versão 3.0, interface do módulo 02 software	66
8.1	Solução gerada para calculo da pressão hidrostática	69
8.2	Soluções geradas pelo código para modelo Newtoniano no poço e anular .	71
8.3	Soluções geradas pelo código para modelo de Bingham no poço e anular .	73

8.4	Soluções geradas pelo código para modelo de lei de Potência no poço e anular	76
8.5	Solução gerada para variação do comprimento do tudo devido a força pistão	77
8.6	Solução gerada para variação do comprimento do tudo devido ao efeito balão	79
8.7	Solução gerada para variação do comprimento do tudo devido ao efeito da temperatura	80
8.8	Opção de Salvar	81
8.9	Modelo do Arquivo .dat	81
8.10	Opção de importação de arquivos no software	82
9.1	Logo e documentação do <i>software</i>	85
9.2	Código fonte da classe CSimuladorPoco, no <i>Doxygen</i>	85
A.1	Versão 1.1, Menu de interface do software	91
A.2	Versão 1.1, interface do modulo 01 do software	91
A.3	Versão 1.1, interface do modulo 02 software	92
A.4	Acesso às funcionalidades de configuração do poço e dos fluidos	94
A.5	Exemplo de estrutura de arquivo .dat para importação de dados	95
A.6	Exemplo de estrutura de arquivo .dat para importação de dados	96

Lista de Tabelas

3.1	Características básicas do Software Educacional Para Análise e Solução de Problemas em Engenharia de Poço. Fonte: Elaborado pelo autor (2025) . . .	20
3.2	Caso de uso geral	23

Sumário

1	Introdução	15
1.1	Escopo do Problema	15
1.2	Objetivos	15
2	Metodologia	17
3	Concepção	20
3.1	Nome do Sistema/Produto	20
3.2	Especificação	21
3.3	Requisitos	21
3.3.1	Requisitos Funcionais	21
3.3.2	Requisitos Não Funcionais	22
3.4	Casos de Uso do Software	23
3.4.1	Diagrama de Caso de Uso Geral	23
3.4.2	Diagrama de Caso de Uso Específico	24
4	Elaboração	26
4.1	Análise de Domínio	26
4.2	Formulação Teórica	27
4.2.1	Ementa da Disciplina	27
4.2.2	Termos e Unidades	30
4.2.3	Pressão Hidrostática	31
4.2.4	Pressão Hidrostática em Colunas Com Mais de Um Tipo de Fluido	33
4.2.5	Modelos Reológicos de Fluidos de Perfuração	35
4.2.6	Perda de Pressão Friccional em um Tubo de Perfuração	38
4.2.7	Perda de Pressão Friccional em um Anular	39
4.2.8	Variações de Carga Axial e Deslocamento em Colunas de Poço . . .	41
4.3	Identificação de Pacotes – Assuntos	44
4.4	Diagrama de Pacotes – Assuntos	45
5	AOO – Análise Orientada a Objeto	46
5.1	Diagramas de Classes	46

5.2	Diagrama de Sequência – Eventos e Mensagens	50
5.3	Diagrama de Comunicação – Colaboração	52
5.4	Diagrama de Máquina de Estado	52
5.5	Diagrama de Atividades	53
6	Projeto	55
6.1	Projeto do sistema	55
6.2	Diagrama de componentes	56
7	Ciclos de planejamento/detalhamento	58
7.1	Versão 1.0 – Interação via Terminal e Geração de Gráficos com Gnuplot	58
7.2	Versão 1.1 – Otimização de Entrada, Validação e Armazenamento Automático de Dados	59
7.3	Versão 2.0 – Interface Gráfica, Amigável e Intuitiva Utilizando o Framework Qt	61
7.4	Versão 2.6 – Consolidação Visual, Barra de Tarefas e Automação de Processos	62
7.5	Versão 3.0 – Navegação por Módulos e Simulação Mecânica de Variação de Comprimento (ΔL)	64
8	Resultados	67
8.1	Metodologia de Validação do Software	67
8.2	Casos de Teste	68
8.3	Validação da Pressão Hidrostática	68
8.4	Validação da Perda de Fricção pelo Modelo Newtoniano no Poço e Anular	69
8.5	Validação da Perda de Fricção pelo Modelo <i>Binghan</i> no Poço e Anular	71
8.6	Validação da Perda de Fricção pelo Modelo Potência no Poço e Anular	74
8.7	Validação da Variação do Comprimento do Tubo Devido ao Packer	76
8.8	Validação da Variação do Comprimento do Tubo Devido ao Efeito Balão	78
8.9	Validação da Variação do Comprimento do Tubo Devido ao Efeito da Temperatura	79
8.10	Validação da Importação e Exportação do Documento (“salvar como...”)	80
8.11	Conclusão dos Testes	82
9	Documentação	84
9.1	Documentação do usuário	84
9.2	Documentação do desenvolvedor	84
10	Conclusão	86
10.1	Sugestões para trabalhos futuros	86
	Referências Bibliográficas	88

A Manual do desenvolvedor	89
A.1 Instalação	89
A.1.1 Dependências obrigatórias para rodar o software	89
A.1.2 Bibliotecas padrão da linguagem C++ (não precisam ser instaladas separadamente)	90
A.1.3 Observações importantes	90
A.1.4 Execução sem compilador	90
A.2 Interface gráfica	91
A.3 Funcionalidades	92
A.3.1 Módulo 1 – Hidráulica de Perfuração	93
B Ciclos Construção - Implementação	97
B.1 Código-Fonte (modelo)	97
.0.1 Código Qt (interface)	157

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, os códigos desenvolvidos e testes do software (este documento).

Capítulo 2

Metodologia

O desenvolvimento deste projeto está diretamente associado ao conteúdo programático de três disciplinas do curso de Engenharia de Petróleo da Universidade Estadual do Norte Fluminense (UENF), oferecidas pelo Laboratório de Engenharia e Exploração de Petróleo (LENEP): **Introdução ao Projeto de Engenharia, Programação Orientada a Objetos com C++ e Projeto de Software Aplicado à Engenharia.**

Na disciplina **LEP01348 – Introdução ao Projeto de Engenharia**, foram apresentados os fundamentos da metodologia utilizada neste trabalho. Essa disciplina fornece a base teórica para o planejamento e organização de projetos de engenharia, abordando desde a definição de escopo até a estruturação em ciclos de desenvolvimento e validação incremental. A metodologia aplicada neste TCC é diretamente derivada dos princípios explorados nessa disciplina.

A disciplina **LEP01447 – Programação Orientada a Objetos com C++** oferece os conceitos fundamentais da linguagem C++, com foco em abstração, encapsulamento, herança, polimorfismo e estrutura modular de software. Embora não seja voltada à construção prática de projetos completos, essa disciplina é essencial para compreender a lógica por trás da organização do código, da modelagem de classes e do comportamento dos objetos utilizados no software desenvolvido.

Por fim, em **LEP01449 – Projeto de Software Aplicado à Engenharia**, os conhecimentos adquiridos nas disciplinas anteriores são integrados na construção de um produto de software voltado à solução de problemas reais da área de engenharia. Nessa etapa, a metodologia estruturada é aplicada em conjunto com os conceitos de orientação a objetos, possibilitando o desenvolvimento de uma solução funcional, validada por meio de testes e simulações.

Essas três disciplinas fornecem a base conceitual e prática necessária para o desenvolvimento completo do projeto, desde sua concepção até a entrega do produto final.

A metodologia é descrita e disponível em

<https://github.com/ldsc/LDSC-ProjetoEngenharia-0-Metodologia-Instrucoes-Etapas>.

O desenvolvimento do software seguiu um processo incremental e iterativo, dividido

em três grandes fases: **o ciclo de concepção e análise, o ciclo de planejamento detalhado e o ciclo de construção**. A estrutura metodológica adotada baseia-se nos princípios do desenvolvimento orientado a características e foi fortemente influenciada por práticas ágeis, similares ao framework Scrum, com reuniões de acompanhamento e entregas parciais contínuas.

Na etapa de **gestão**, organizou-se uma rotina de reuniões periódicas, presenciais e remotas, via Google Meet, com o objetivo de alinhar as decisões técnicas e garantir a rastreabilidade do progresso. A comunicação da equipe foi mantida por meio do aplicativo Telegram, assegurando agilidade e registro das discussões. O controle de versões foi realizado com o uso da plataforma GitHub, permitindo a organização modular do código e a recuperação de versões anteriores quando necessário.

O **ciclo de concepção e análise** teve como objetivo a construção de um modelo abrangente do sistema e a definição de uma lista de características do produto. Essa etapa envolveu atividades como a concepção inicial, a elaboração de protótipos, a documentação preliminar, os testes lógicos e a modelagem do sistema com o auxílio de diagramas UML (casos de uso, classes, sequência, atividades, componentes, entre outros). Como resultado, obteve-se uma **visão do produto** e uma **lista estruturada de características funcionais**, que serviram como base para o planejamento das etapas seguintes.

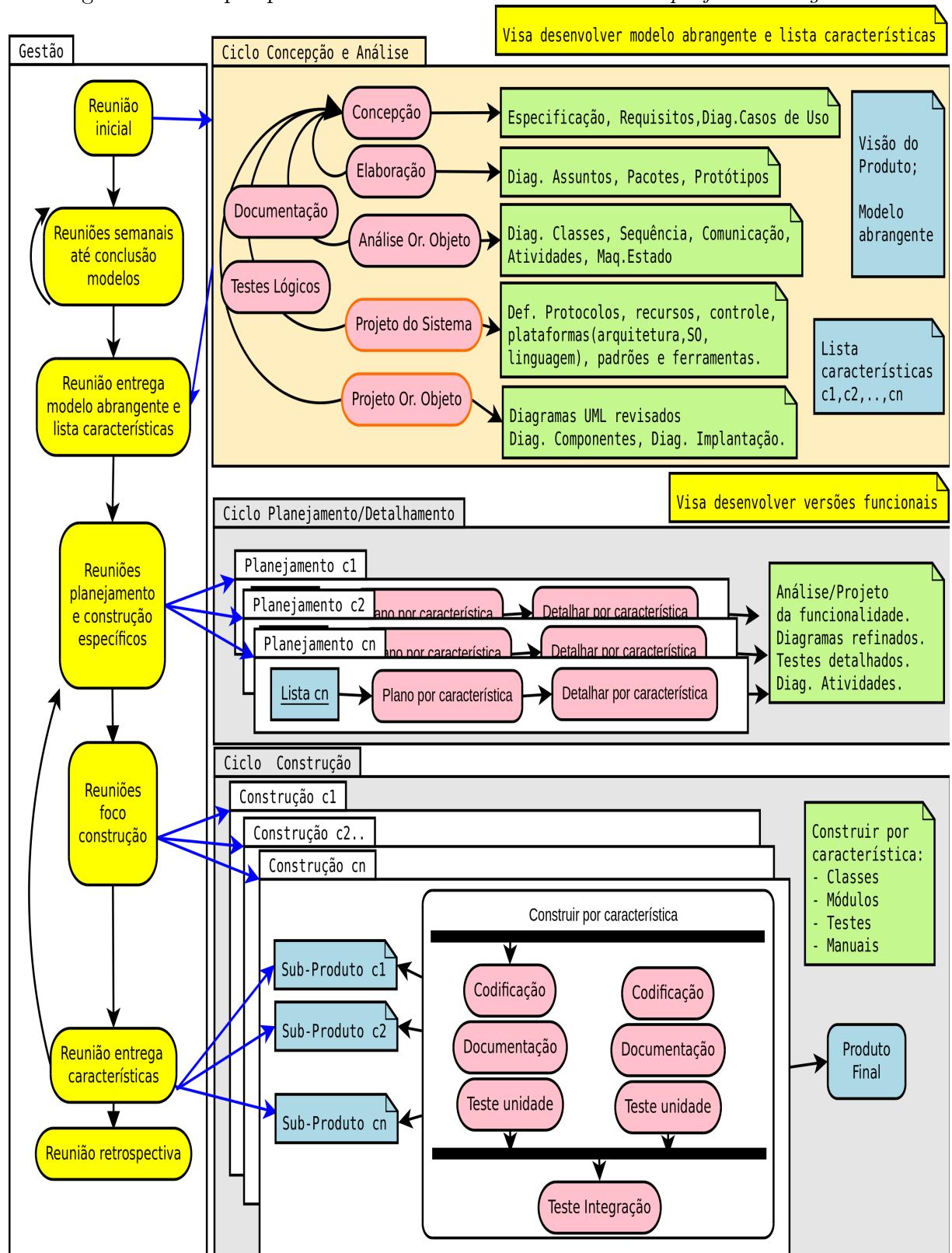
Na fase seguinte, o **ciclo de planejamento e detalhamento**, cada característica identificada foi tratada individualmente. Foram definidos os planos de desenvolvimento específicos, bem como a análise funcional detalhada, o projeto de cada módulo, os testes esperados e os diagramas de apoio. Essa abordagem permitiu uma organização incremental, em que cada funcionalidade do sistema foi planejada de forma independente, possibilitando entregas parciais coerentes e contínuas.

Por fim, no **ciclo de construção**, cada característica foi implementada de forma isolada, passando pelas etapas de codificação, documentação, testes de unidade e integração. Cada entrega parcial gerou um **subproduto validado**. Ao final da construção de todas as características planejadas, consolidou-se o **produto final**, que integra todas as funcionalidades de forma coesa e funcional.

Conforme ilustrado na 2.1, todo esse processo é representado em um diagrama que sintetiza as fases do projeto, seus ciclos e entregas, bem como a estrutura de reuniões, entradas e saídas de cada etapa. O diagrama permite visualizar a progressão do desenvolvimento desde a concepção até a integração final do sistema.

O uso da linguagem **C++ com o framework Qt** permitiu o desenvolvimento de interfaces gráficas intuitivas e responsivas, ao mesmo tempo em que a modularização do código facilitou a manutenção e expansão futura do software. A metodologia adotada garantiu não apenas a organização técnica do projeto, mas também a rastreabilidade das decisões e a validação contínua das entregas.

Figura 2.1: 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 3

Concepção

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

3.1 Nome do Sistema/Produto

O sistema desenvolvido, denominado Software Educacional para Análise e Solução de Problemas em Engenharia de Poço. A base tecnológica permitiu a criação de uma interface gráfica robusta, que facilita a navegação e oferece recursos visuais para apoio ao ensino.

A 3.1, apresentada a seguir, resume as principais características do sistema, incluindo seu nome, missão e os componentes que o integram.

Tabela 3.1: 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.

3.2 Especificação

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 *crossovers*.

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.

Além disso ele 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).

3.3 Requisitos

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

3.3.1 Requisitos Funcionais

Apresenta-se a seguir os requisitos funcionais.

RF-01	O sistema 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 pode carregar dados de propriedade para a simulação.
RF-03	O usuário tem a liberdade para alterar as propriedades reológicas do poço/fluido.
RF-04	Permite a exportação de simulações.
RF-05	Permite cenários de simulação baseado em diferentes modelos teóricos.
RF-06	O usuário pode comparar os resultados da simulação em diferentes modelos reológicos.
RF-07	O usuário tem a liberdade para adicionar ou retirar simplificações das premissas do modelo.

RF-08	O usuário pode visualizar seus resultados em um gráfico. O gráfico poderá ser salvo como imagem.
RF-09	O sistema permite a análise dos deslocamentos axiais (ΔL) da coluna de completação em diferentes condições operacionais.
RF-10	O sistema contempla variações térmicas, efeito balão, atuação de pistão, <i>packer</i> e <i>crossover</i> nas análises de carga.
RF-121	O sistema permite 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-12	O sistema salva automaticamente os dados da simulação em arquivos nomeados conforme o poço selecionado, mantendo um histórico completo das ações do usuário
RF-13	O sistema realiza a verificação automática de entradas inválidas, garantindo estabilidade e prevenindo falhas durante a execução.
RF-14	O sistema oferece atalhos de teclado, como Ctrl+N para nova simulação e Ctrl+S para salvar, otimizando a interação com o usuário.
RF-15	O sistema disponibiliza uma seção de ajuda com instruções de uso e descrição dos modelos implementados.

3.3.2 Requisitos Não Funcionais

RNF-01	Suas primeiras versões suportam os sistemas operacionais GNU/Linux e <i>Windows</i> .
RNF-02	A linguagem predominante utilizada no desenvolvimento é C++.
RNF-03	A geração dos gráficos foram realizadas por meio da biblioteca QCustomPlot.
RNF-04	O sistema permite a exportação dos resultados em arquivos de texto e no formato .dat.
RNF-05	A interface gráfica foi desenvolvida com o Qt Framework, oferecendo usabilidade intuitiva.

RNF-06	O sistema tem uma organização modular do código, facilitando manutenção e futuras expansões.
RNF-07	Os arquivos de entrada e saída seguem padrões consistentes e documentados para garantir interoperabilidade.
RNF-08	Os arquivos usam a codificação de caracteres UTF-8

3.4 Casos de Uso do Software

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

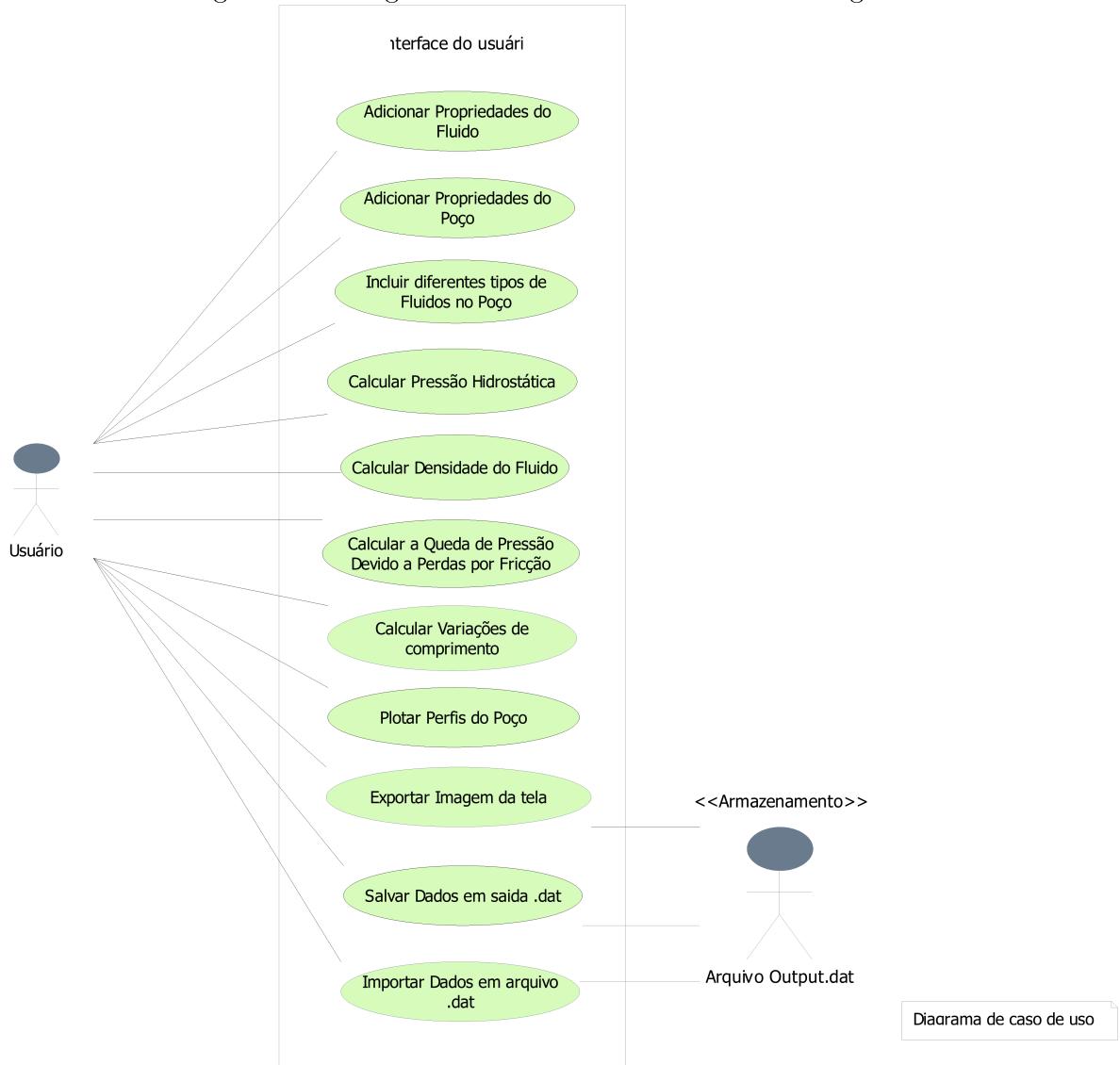
3.4.1 Diagrama de Caso de Uso Geral

As condições do caso de uso geral são apresentadas na Tabela 3.2. Este cenário é representado graficamente pelo diagrama de caso de uso geral da Figura 3.1. 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 3.2: Caso de uso geral

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

Figura 3.1: Diagrama de caso de uso – caso de uso geral

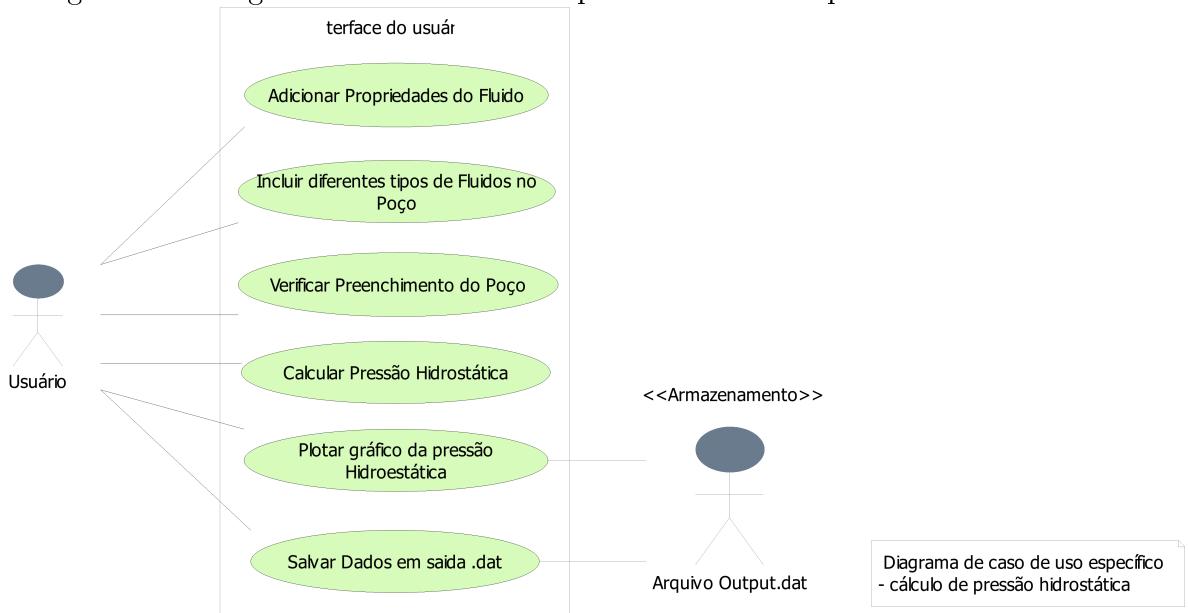


Fonte: Produzido pelo autor.

3.4.2 Diagrama de Caso de Uso Específico

O caso de uso específico da Figura 3.2 mostra o cenário onde o usuário deseja “Calcular a pressão hidrostática” no poço.

Figura 3.2: Diagrama de caso de uso específico: “calcular pressão hidrostática”



Fonte: Produzido pelo autor.

O caso de uso específico da Figura 3.3 mostra o cenário onde o usuário deseja calcular a perda de pressão devido a perda por fricção no poço e no anular.

Figura 3.3: 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 4

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.

Nota sobre a fonte do conteúdo: Este capítulo foi inteiramente desenvolvido a partir das referências bibliográficas Applied Drilling Engineering (Jr. *et al.* (1991), 1986) e Fundamentals of Drilling Engineering (Mitchell & Miska (2011), 2011). Todo o embasamento teórico, as formulações matemáticas, os algoritmos descritos e demais conteúdos apresentados foram extraídos dessas obras, com adaptações de linguagem e organização para fins acadêmicos, conforme exigências deste trabalho.

4.1 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. A integração com a API Qt permite criar uma interface intuitiva e interativa, essencial para o uso didático e técnico do simulador.

4.2 Formulação Teórica

4.2.1 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.

4.2.2 Termos e Unidades

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

- Z é a profundidade [ft];
- dZ é a variação de profundidade [ft];
- d é o diâmetro interno do revestimento ID [in];
- d_1 é o diâmetro externo do revestimento OD [in];
- d_2 é o diâmetro do poço [in];
- p_0 é a constante de integração igual a pressão na superfície [psi];
- p é a pressão [psi];
- dp é a variação de pressão [psi];
- ΔP_{in} Variação de pressão na parte interna do poço [psi];
- ΔP_{out} Variação de pressão no anular [psi];
- τ é a tensão de cisalhamento exercida sobre o fluido [psi];
- $\dot{\gamma}$ é a taxa de cisalhamento [$1/s$];
- τ_y é a tensão de escoamento ou o ponto de escoamento [$lbf/100.sq.ft$];
- τ_w é a tensão de cisalhamento na parede [lbf/ft^2];
- ρ é a densidade do fluido [lbm/gal];
- T é a temperatura absoluta [$^{\circ}R$];
- μ é a viscosidade aparente [cP];
- μ_p é a viscosidade plástica [cP];
- g é a gravidade [ft/s^2];
- v velocidade [ft/s];
- q é a vazão do poço [gal/min];
- z é o fator de desvio de gás;
- n é o expoente da lei de potência ou o índice de comportamento do fluxo;
- R é constante universal dos gases [$psi.ft^3/lb - mol.^{\circ}R$];

- M é o peso molecular do gás [$lb/lb - mol$];
- K é o índice de consistência do fluido [cP];
- f é o fator de fricção;
- E Módulo de elasticidade [psi];
- N_{re} é o número de Reynolds;
- N_{rec} é o número de Reynods crítico;
- N_{He} é o número de Hedstrom;
- $\frac{dp_f}{dL}$ é a perda de pressão por fricção [psi/ft];
- L Comprimento do tubo [ft];
- ΔL Variação de comprimento do tubo [ft];
- ΔL_b Variação de comprimento do tubo devido efeito balão [ft];
- ΔL_{packer} Variação de comprimento do tubo devido força pistão [ft];
- $\Delta L_{crossover}$ Variação de comprimento do tubo devido ao crossover [ft];

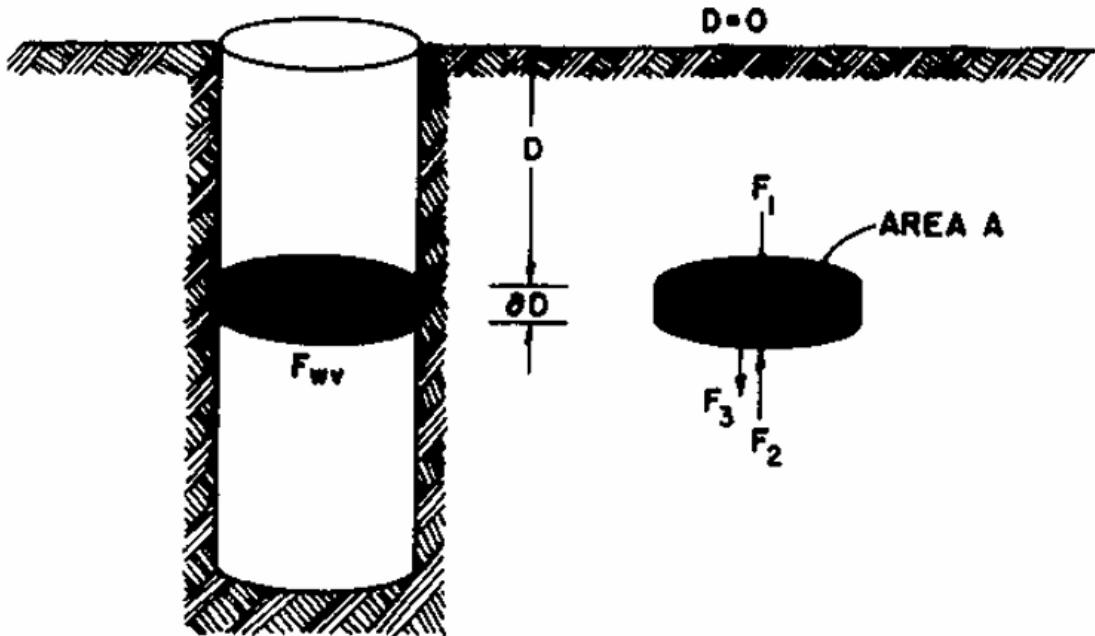
4.2.3 Pressão Hidrostática

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 um dos tipos de problemas: os estáticos, que envolvem o cálculo da pressão hidrostática (Jr. *et al.* (1991)).

A pressão hidrostática corresponde à variação da pressão em função da profundidade ao longo de uma coluna de fluido, sendo comumente calculada em condições de poço estático. Sua dedução pode ser realizada a partir da análise do diagrama de corpo livre apresentado na Figura 4.1.

Figura 4.1: Diagrama de corpo livre, atuação de forças em um elemento de fluido



Fonte Jr. et al. (1991)

No diagrama da Figura 4.1, observa-se a atuação de três forças principais sobre um elemento cilíndrico de fluido com área transversal A e altura dZ , situado a uma profundidade z . A força F_1 representa a pressão exercida pelo fluido sobre a face superior do elemento, atuando para baixo, enquanto F_2 é a força de pressão exercida pela camada inferior do fluido, atuando para cima. Já a força F_w corresponde ao peso do elemento de fluido, atuando também para baixo. A diferença entre as forças F_1 e F_2 , em equilíbrio com o peso do fluido, permite deduzir a equação da variação da pressão com a profundidade, conforme mostrado na Equação 4.1.

$$\frac{dp}{dz} = 0.05195\rho \quad (4.1)$$

onde 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].

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 (4.1), obtém-se a equação hidrostática para fluidos incompressíveis:

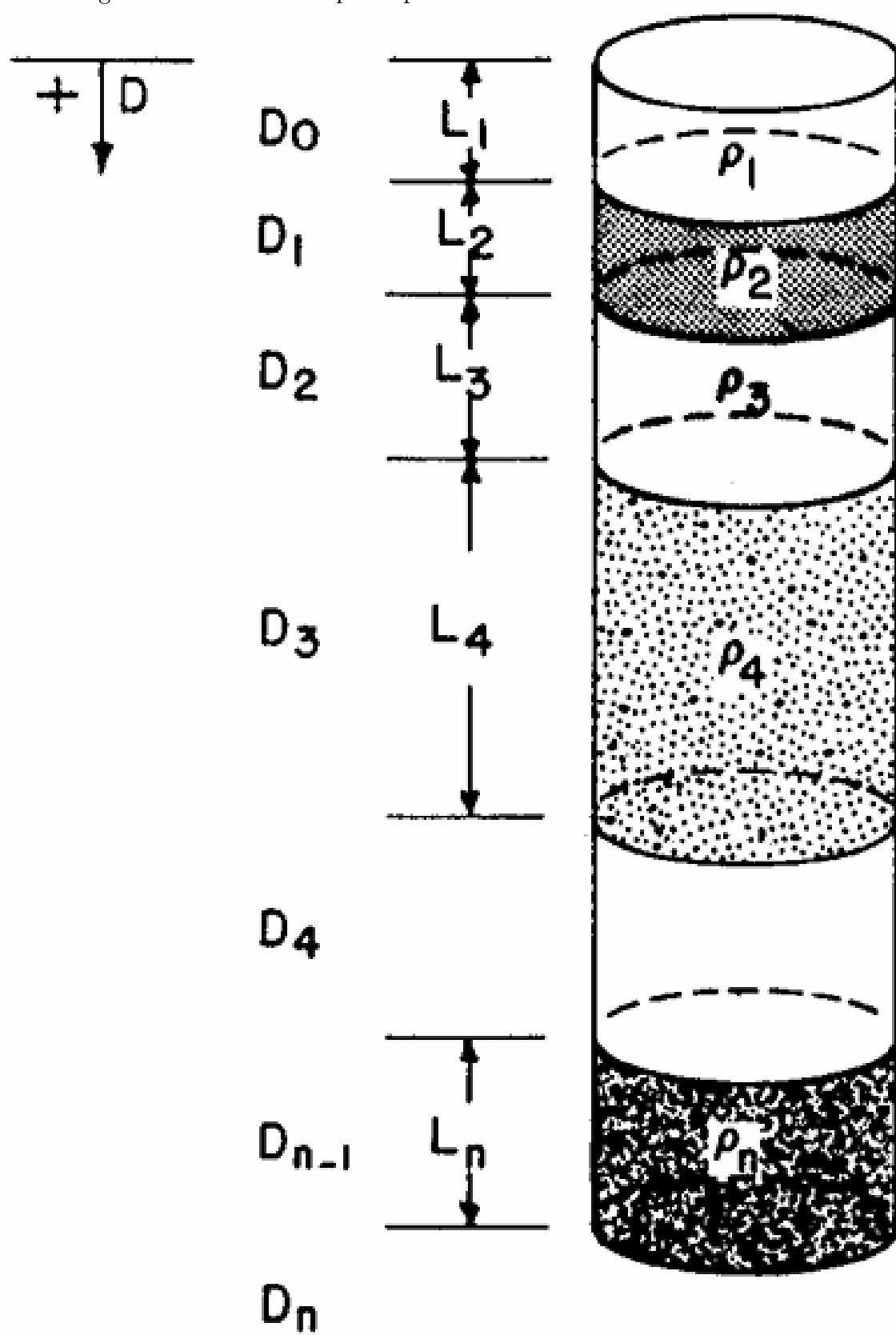
$$p = 0.05195\rho Z + p_0 \quad (4.2)$$

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 *kick* 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)).

4.2.4 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 4.2.

Figura 4.2: 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:

$$p = p_0 + g \sum p_i (Z_i - Z_{i-1}) + g \rho_n (Z_i - Z_{i-1}) \quad (4.3)$$

Onde g é a gravidade [ft/s^2].

4.2.5 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.

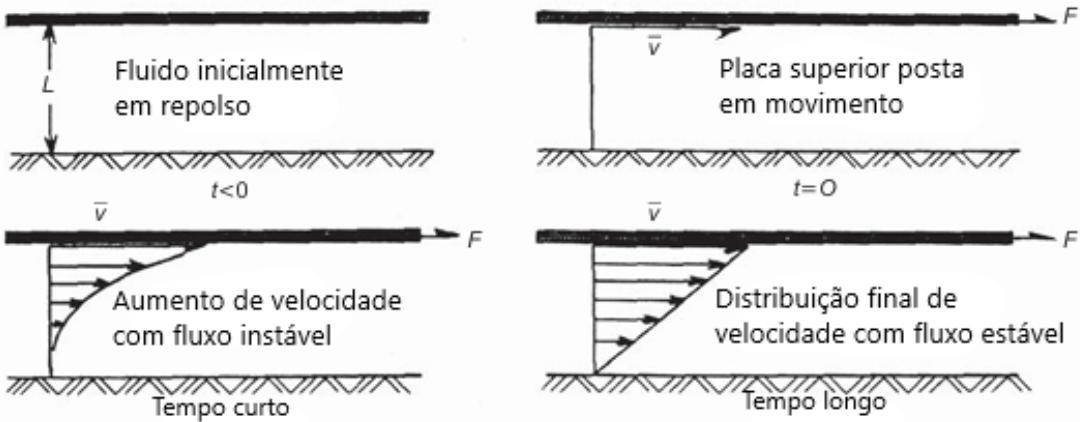
Na maioria dos casos, as propriedades elásticas dos fluidos de perfuração e seus efeitos durante o escoamento no poço são desprezíveis, sendo consideradas apenas as forças viscosas nos cálculos. Entretanto, com o avanço tecnológico, lamas cada vez mais complexas vêm sendo desenvolvidas, e, portanto, os testes devem considerar também as propriedades elásticas associadas à deformação do fluido durante o escoamento (Mitchell & Miska (2011)).

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 (Jr. *et al.* (1991)). 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 4.3.

Figura 4.3: 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 [$lb f$] constante é necessária para manter a placa superior em movimento , a magnitude dessa força pode ser determinada por:

$$\frac{F}{A} = \mu \frac{\nu}{L} \quad (4.4)$$

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:

$$\tau = \frac{F}{A} \quad (4.5)$$

Em reologia, os termos taxa de cisalhamento e taxa de deformação são frequentemente utilizados como sinônimos, ambos representados por $\dot{\gamma}$ [1/s]. Esse parâmetro descreve o gradiente da velocidade entre camadas adjacentes de fluido em movimento relativo:

$$\dot{\gamma} = \frac{d\nu}{dL} \approx \frac{\nu}{L} \quad (4.6)$$

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:

$$\tau = \mu \dot{\gamma} \quad (4.7)$$

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 (4.7) 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:

$$\tau = \tau_y + \mu_p \dot{\gamma} \quad (4.8)$$

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:

$$\tau = K \dot{\gamma}^n \quad (4.9)$$

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.

4.2.6 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:

$$N_{re} = \frac{928\rho\bar{v}d}{\mu} \quad (4.10)$$

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:

$$\bar{v} = \frac{q}{2.448d^2} \quad (4.11)$$

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:

$$\frac{dp_f}{dL} = \frac{\mu\bar{v}}{1500d} \quad (4.12)$$

Para o fluxo turbulento:

$$\frac{dp_f}{dL} = \frac{f\rho\bar{v}^2}{25.8d} \quad (4.13)$$

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 (4.11). O número de Reynolds pode ser obtido a partir da seguinte equação:

$$N_{re} = \frac{928\rho\bar{v}d}{\mu_p} \quad (4.14)$$

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:

$$N_{rec} = \frac{1 - \frac{4}{3} \left(\frac{\tau_y}{\tau_w} \right) + \frac{1}{3} \left(\frac{\tau_y}{\tau_w} \right)^4}{8 \left(\frac{\tau_y}{\tau_w} \right)} N_{He} \quad (4.15)$$

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:

$$N_{He} = \frac{37100 \rho \tau_y d^2}{\mu_p^2} \quad (4.16)$$

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

Para fluxo laminar:

$$\frac{dp_f}{dL} = \frac{\mu_p \bar{v}}{1500 d^2} + \frac{\tau_y}{225 d} \quad (4.17)$$

Para o fluxo turbulento podemos usar a Equação (4.13).

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:

$$N_{re} = \frac{89100 \rho \bar{v}^{2-n}}{K} \left(\frac{0.0416 d}{3 + \frac{1}{n}} \right)^n \quad (4.18)$$

A velocidade média pode ser obtida pela Equação (4.11). 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:

$$\frac{dp_f}{dL} = \frac{K \bar{v}^n \left(\frac{3 + \frac{1}{n}}{0.0416} \right)^n}{144000 d^{1+n}} \quad (4.19)$$

Para o fluxo turbulento podemos usar a Equação (4.13).

4.2.7 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:

$$N_{re} = \frac{757\rho\bar{v}(d_2 - d_1)}{\mu} \quad (4.20)$$

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:

$$\bar{v} = \frac{q}{2.448(d_2^2 - d_1^2)} \quad (4.21)$$

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:

$$\frac{dp_f}{dL} = \frac{\mu\bar{v}}{1000(d_2 - d_1)^2} \quad (4.22)$$

Para o fluxo turbulento:

$$\frac{dp_f}{dL} = \frac{f\rho\bar{v}^2}{21.1(d_2 - d_1)} \quad (4.23)$$

Fluidos plásticos de Bingham

Para um fluido que se comporta como plástico de Bingham a velocidade média pode ser obtida pela Equações (4.21). O número de Reynolds pode ser obtido a partir da seguinte equação:

$$N_{re} = \frac{757\rho\bar{v}(d_2 - d_1)}{\mu_p} \quad (4.24)$$

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 (4.15), mas o número de Hedstrom deve ser calculado pela seguinte equação:

$$N_{He} = \frac{24700\rho\tau_y(d_2 - d_1)^2}{\mu_p^2} \quad (4.25)$$

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

Para fluxo laminar:

$$\frac{dp_f}{dL} = \frac{\mu\bar{v}}{1000(d_2 - d_1)^2} + \frac{\tau_y}{200(d_2 - d_1)} \quad (4.26)$$

Para o fluxo turbulento podemos usar a Equação (4.23).

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:

$$N_{re} = \frac{109000\rho\bar{v}^{2-n}}{K} \left(\frac{0.0208(d_2 - d_1)}{2 + \frac{1}{n}} \right)^n \quad (4.27)$$

A velocidade média pode ser obtida pela Equação (4.21). 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:

$$\frac{dp_f}{dL} = \frac{K\bar{v}^n \left(\frac{2+\frac{1}{n}}{0.0208} \right)^n}{144000(d_2 - d_1)^{1+n}} \quad (4.28)$$

Para o fluxo turbulento podemos usar a Equação (4.23).

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

4.2.8 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 de uma coluna de produção causa dilatação ou contração térmica, influenciando diretamente o seu comprimento axial. Esse efeito é particularmente importante em poços profundos, onde o gradiente geotérmico e as operações de circulação de fluido podem resultar em variações térmicas significativas. O alongamento ou encurtamento da coluna por temperatura é um fenômeno reversível, assumindo-se comportamento elástico do material.

$$\Delta L_t = \int_0^L \alpha_T \Delta T(s) ds \quad (4.29)$$

$$\Delta L_t = \alpha_T \Delta T \int_0^L ds \quad (4.30)$$

$$\Delta L_t = \alpha_T L \Delta T \quad (4.31)$$

Onde α_T é o coeficiente de expansão térmica linear, L o comprimento da coluna e ΔT a variação térmica ao longo da profundidade considerada. Essa formulação é derivada da equação 7.57e apresentada por Mitchell & Miska (2011) (2011, p. 436).

Efeito Balão (Ballooning Effect)

O efeito balão ocorre quando há uma diferença entre a pressão interna e externa atuante sobre a parede do tubo. Essa diferença provoca uma deformação radial, que se traduz em uma variação de comprimento axial da coluna. O tubo se comporta como um cilindro elástico: ao expandir radialmente por pressão interna, ocorre também um leve alongamento longitudinal, mesmo sob regime de equilíbrio estático. Este fenômeno é tratado na equação 7.57d por Mitchell & Miska (2011), como parte da variação total de comprimento da coluna.

$$\Delta L_b = \int_0^L \frac{2v[\Delta P_i A_i - \Delta P_e A_e]}{EA} ds \quad (4.32)$$

$$\Delta L_b = \frac{2v[\Delta P_{in} A_{in} - \Delta P_{out} A_{out}]}{E(A_{out} - A_{in})} \int_0^L ds \quad (4.33)$$

$$\Delta L_b = \frac{2v[\Delta P_{in} A_{in} - \Delta P_{out} A_{out}]L}{E(A_{out} - A_{in})} \quad (4.34)$$

Onde v é o coeficiente de Poisson, E é o módulo de elasticidade, L o comprimento da coluna, P_{in} pressão interna, P_{out} pressão externa, A_{in} área interna e A_{out} área externa.

Variação Axial Devido à Força de Pistão (ΔF)

A força de pistão é um fenômeno mecânico relevante no comportamento de colunas de revestimento em poços de petróleo, surgindo em situações onde há descontinuidades geométricas ou barreiras à livre movimentação axial, como crossovers e packers. Este efeito resulta da aplicação de pressões desbalanceadas sobre áreas distintas ao longo da coluna, gerando uma força líquida que provoca deslocamento axial. Esse comportamento é especialmente crítico em cenários de altas pressões diferenciais, podendo contribuir para falhas estruturais se não for devidamente considerado no projeto. A formulação matemática para esse efeito encontra-se descrita por Mitchell & Miska (2011), em sua Equação 7.51 (p. 426).

A equação que representa o efeito pistão total sobre a coluna, considerando tanto a pressão interna quanto a externa atuando em áreas diferentes, é expressa por:

$$\Delta F = \Delta P_{in} A_{in} + \Delta P_{out} A_{out} \quad (4.35)$$

Essa equação é geral e pode ser aplicada a qualquer ponto da coluna onde haja mudança de geometria ou contenção axial.

A força de pistão produzida atua axialmente e gera uma deformação elástica ao longo da coluna, proporcional à sua rigidez axial. O deslocamento resultante pode ser determinado pela Lei de Hooke para tração/compressão axial:

$$\Delta L = \frac{(\Delta F)L}{EA_s} \quad (4.36)$$

Essa expressão permite estimar o alongamento ou encurtamento da coluna em decorrência de efeitos locais de pressão.

Força de Pistão Gerada por Packer

Quando um packer é instalado em uma coluna, ele atua como um ponto fixo que impede a movimentação axial da tubulação. Isso gera uma condição assimétrica, onde a pressão do fluido acima e abaixo do packer pode ser distinta. Essa diferença de pressão exerce uma força líquida axial sobre a coluna, caracterizando o chamado efeito pistão.

A força resultante é dada por:

$$\Delta F_{packer} = (P_{acima} - P_{abaixo})A_{packer} \quad (4.37)$$

Onde A_{packer} é a área efetiva no qual a diferença de pressão atua. Essa força provoca um deslocamento axial calculado por:

$$\Delta L_{packer} = \frac{(\Delta F_{packer})L}{EA_s} \quad (4.38)$$

Essa deformação pode impactar a integridade da coluna, principalmente em completações profundas ou com altos diferenciais de pressão.

Força de Pistão em Crossovers Em pontos da coluna onde há mudanças geométricas, como transições entre seções com diferentes diâmetros ou espessuras, ocorre o que se denomina crossover. Nessas regiões, as áreas internas e externas variam, o que implica em diferenças de força quando submetidas à mesma pressão.

A força líquida causada por esse desbalanceamento é expressa como:

$$\Delta F_{crossover} = P_{in}(A_{in+} - A_{in-}) + P_{out}(A_{out+} - A_{out-}) \quad (4.39)$$

onde A_{in+} / A_{in-} são as áreas internas acima e abaixo do ponto de transição e A_{out+} / A_{out-} são as áreas externas imediatamente acima e abaixo.

A variação de comprimento causada por essa força é dada por:

$$\Delta L_{crossover} = \frac{(\Delta F_{crossover})L}{EA_s} \quad (4.40)$$

Essa formulação é essencial para prever deslocamentos localizados que ocorrem em colunas compostas por múltiplos trechos com diferentes especificações geométricas ou mecânicas.

Efeitos de Injeção: Coluna Livre vs. Coluna Fixa Durante operações de injeção, como na pressurização do poço por circulação de fluidos, a variação da pressão interna influencia diretamente o comportamento axial da coluna de revestimento. Essa influência depende da condição de restrição mecânica da coluna:

- **Coluna livre:** quando a extremidade inferior da coluna está desacoplada ou apenas parcialmente apoiada, permite-se sua expansão ou contração axial. Nesse cenário, parte da energia associada à variação de pressão é absorvida na forma de deformação longitudinal, reduzindo a magnitude da carga axial transmitida.
- **Coluna fixa:** quando a extremidade inferior está rigidamente ancorada ou cimentada, a coluna não pode se mover livremente. Assim, qualquer variação de pressão resulta diretamente em um acréscimo ou alívio de carga axial, sem o amortecimento promovido pelo deslocamento axial.

Essa distinção entre colunas fixas e livres é essencial para a previsão do comportamento estrutural durante operações pressurizadas (Mitchell & Miska (2011)).

4.3 Identificação de Pacotes – Assuntos

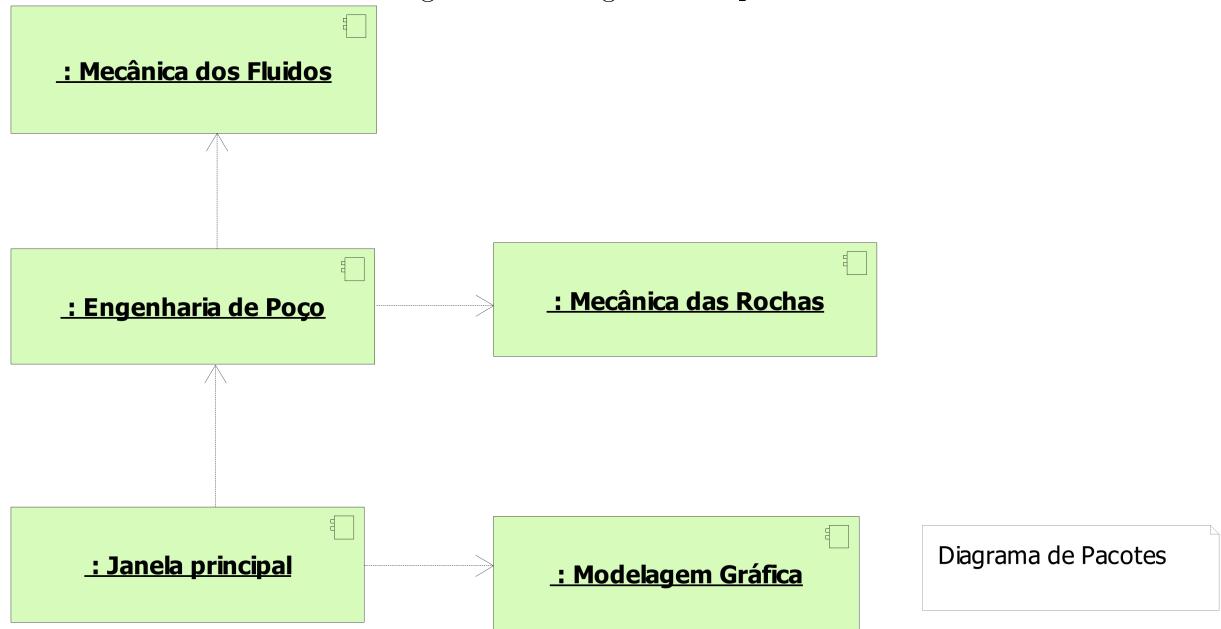
- Pacote engenharia de poço:
 - O pacote engenharia de poço é responsável por relacionar os pacotes mecânicos dos fluidos, mecânica das rochas e equações analíticas de forma a tornar possível e coerente os resultados obtidos pela simulação.
- Pacote mecânica dos fluidos:
 - É o pacote que relaciona todas 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 compreende 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 modelagem gráfica:
 - Esse é o pacote responsável por montar os gráficos que são obtidos a partir dos resultados da simulação.

4.4 Diagrama de Pacotes – Assuntos

O diagrama de pacotes é apresentado na Figura 4.4.

Figura 4.4: Diagrama de pacotes



Capítulo 5

AOO – Análise Orientada a Objeto

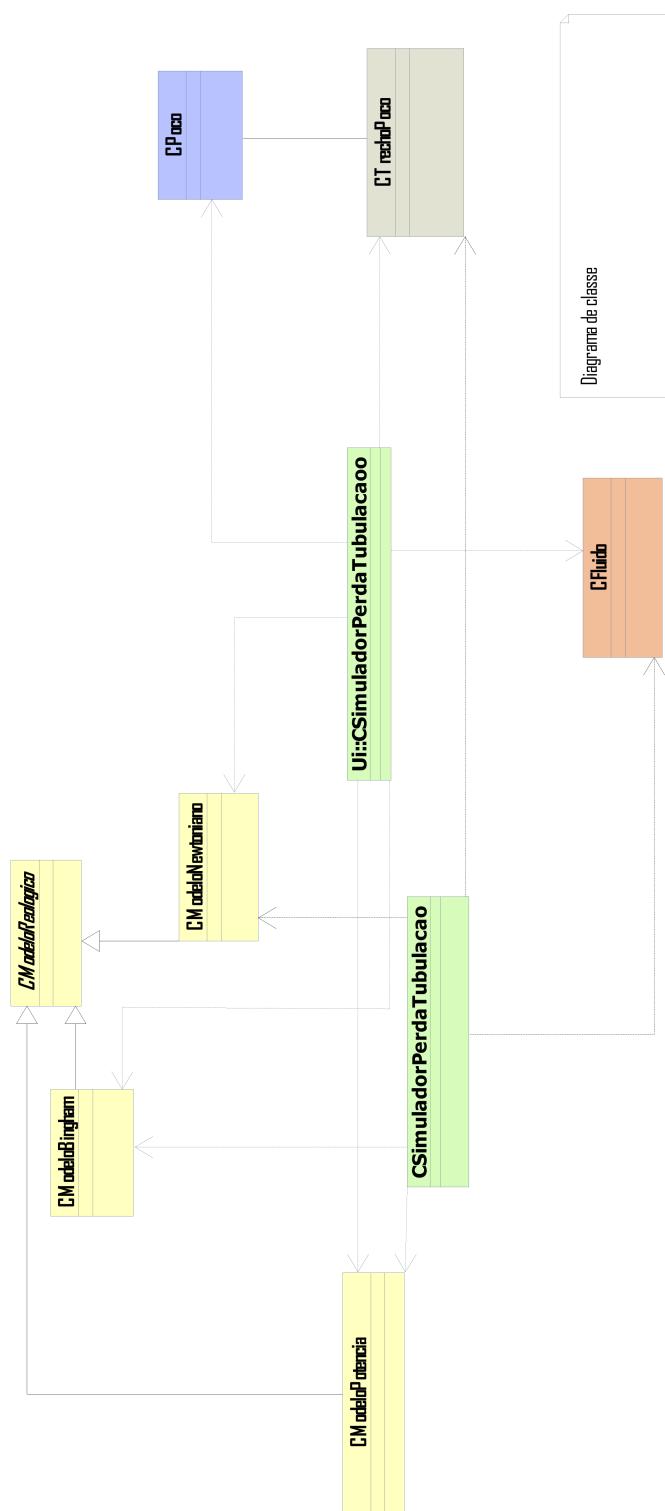
Neste capítulo apresentam-se as classes desenvolvidas no projeto, suas respectivas relações, atributos e métodos. Apresenta-se também um breve conceito de cada classe. Todos os diagramas foram elaborados seguindo a estrutura da UML - *Unified Modeling Language* (Linguagem de Modelagem Unificada), com 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)).

5.1 Diagramas de Classes

A Figura 5.1 apresenta o diagrama de classes simplificado, que representa graficamente a estrutura estática do sistema por meio das principais classes e suas relações. Esse modelo tem como objetivo fornecer uma visão geral da arquitetura do software, destacando apenas os elementos estruturais essenciais, sem detalhar atributos e métodos.

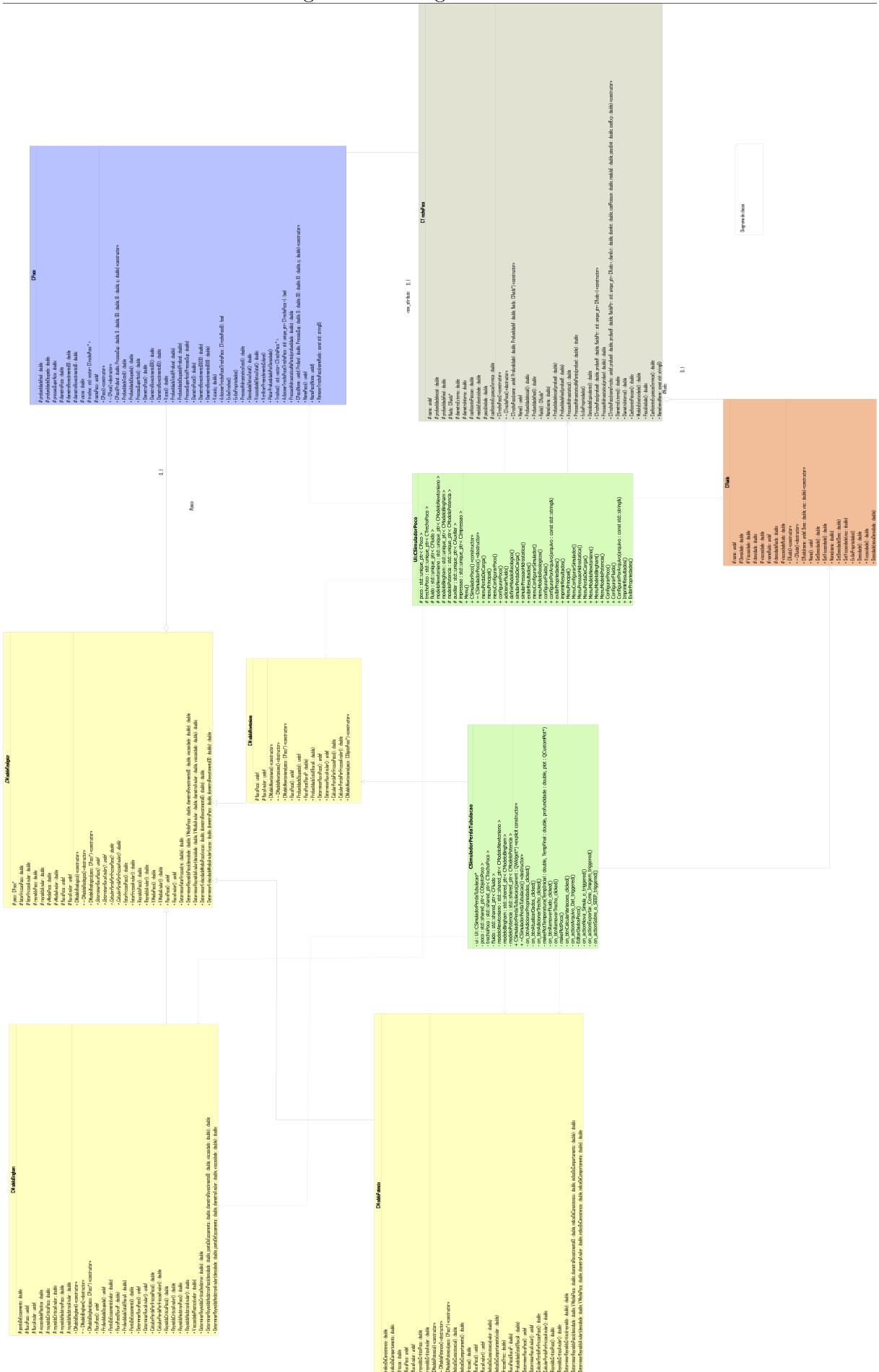
Na sequência, a Figura 5.2 exibe o diagrama de classes completo, conforme a notação UML, incluindo todos os atributos, métodos, heranças e associações entre as classes. Essa representação detalhada é fundamental para o entendimento pleno da arquitetura interna do sistema, permitindo uma análise precisa da organização e das interdependências entre seus componentes.

Figura 5.1: Diagrama de classes simplificado



Fonte: Produzido pelo autor.

Figura 5.2: Diagrama de classes



Fonte: Produzido pelo autor

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 5.2, resume as relações entre as principais entidades do sistema, seus atributos, métodos e heranças, seguindo a notação da Linguagem de Modelagem Unificada (UML)

5.2 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.

Diagrama de Sequência

O diagrama de comunicação do módulo reológico na Figura 5.3 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 5.3: Diagrama de sequência - Módulo reológico



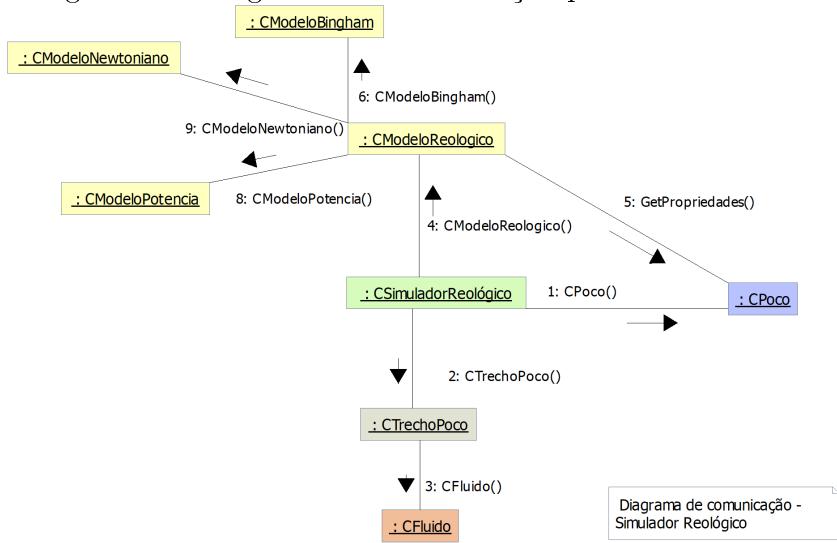
Fonte: Produzido pelo autor.

5.3 Diagrama de Comunicação – Colaboração

O diagrama de comunicação na figura 5.4 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 reológico. 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 5.4: Diagrama de comunicação para caso do cálculo usando modelo reológico



Fonte: Produzido pelo autor.

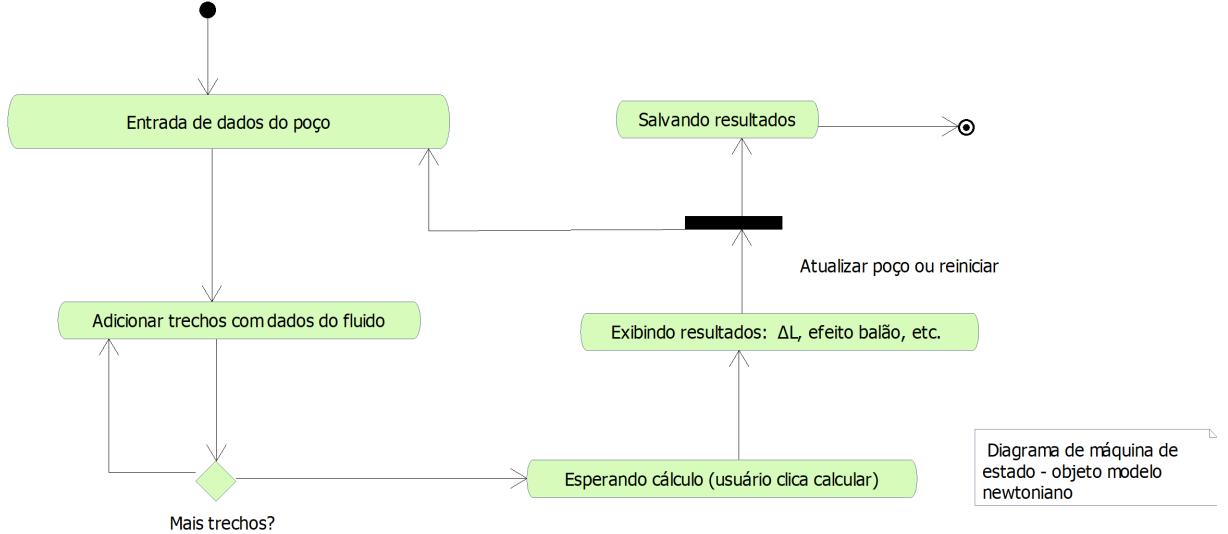
5.4 Diagrama de Máquina de Estado

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 na figura 5.5 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 5.5: Diagrama de máquina de estado do objeto CSIMULADORPERDATUBULACAO



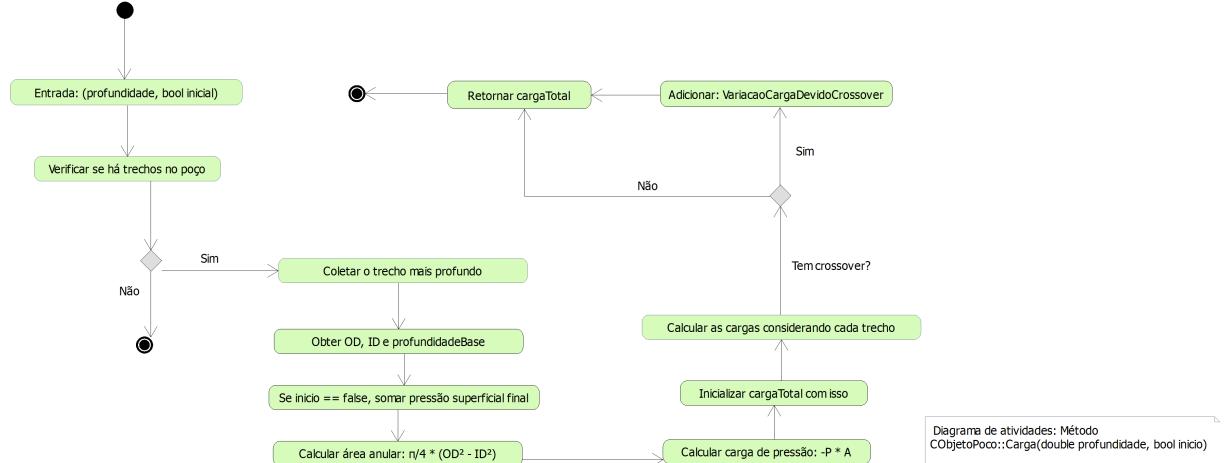
Fonte: Produzido pelo autor.

5.5 Diagrama de Atividades

O diagrama de atividades apresentado na figura 5.6 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 5.6: Diagrama de atividades: CObjetoPoco::Carga(double profundidade, bool inicio)



Fonte: Produzido pelo autor.

Capítulo 6

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.

6.1 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.

6.2 Diagrama de componentes

O diagrama de componentes apresentado na Figura 6.1 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++.

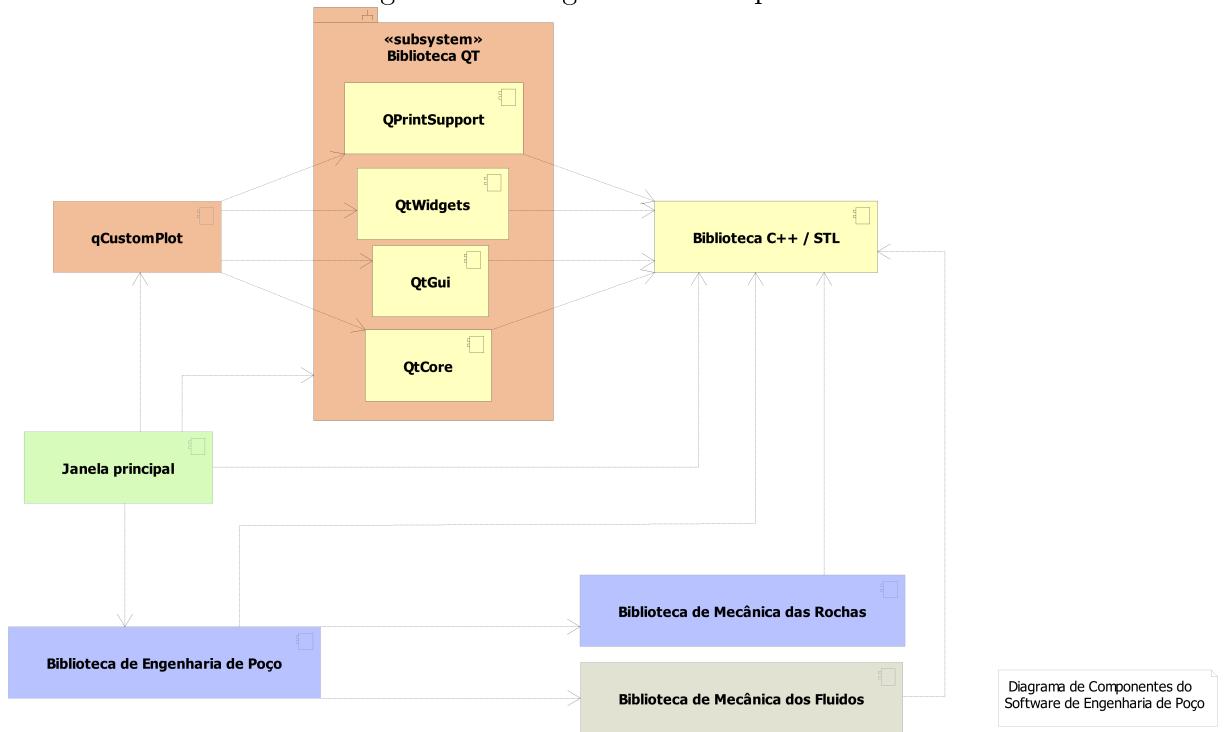
A interface gráfica do sistema é centralizada na Janela Principal, que depende diretamente da biblioteca qCustomPlot para renderização de gráficos técnicos, e se comunica com os módulos do QT. A partir dessa interface, são acionadas funcionalidades específicas implementadas em três bibliotecas técnicas 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 6.1: Diagrama de componentes



Fonte: Produzido pelo autor.

Capítulo 7

Ciclos de planejamento/detalhamento

Apresenta-se neste capítulo as versões do software desenvolvido.

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

Versão: 1.0 (15/12/2024)

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

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

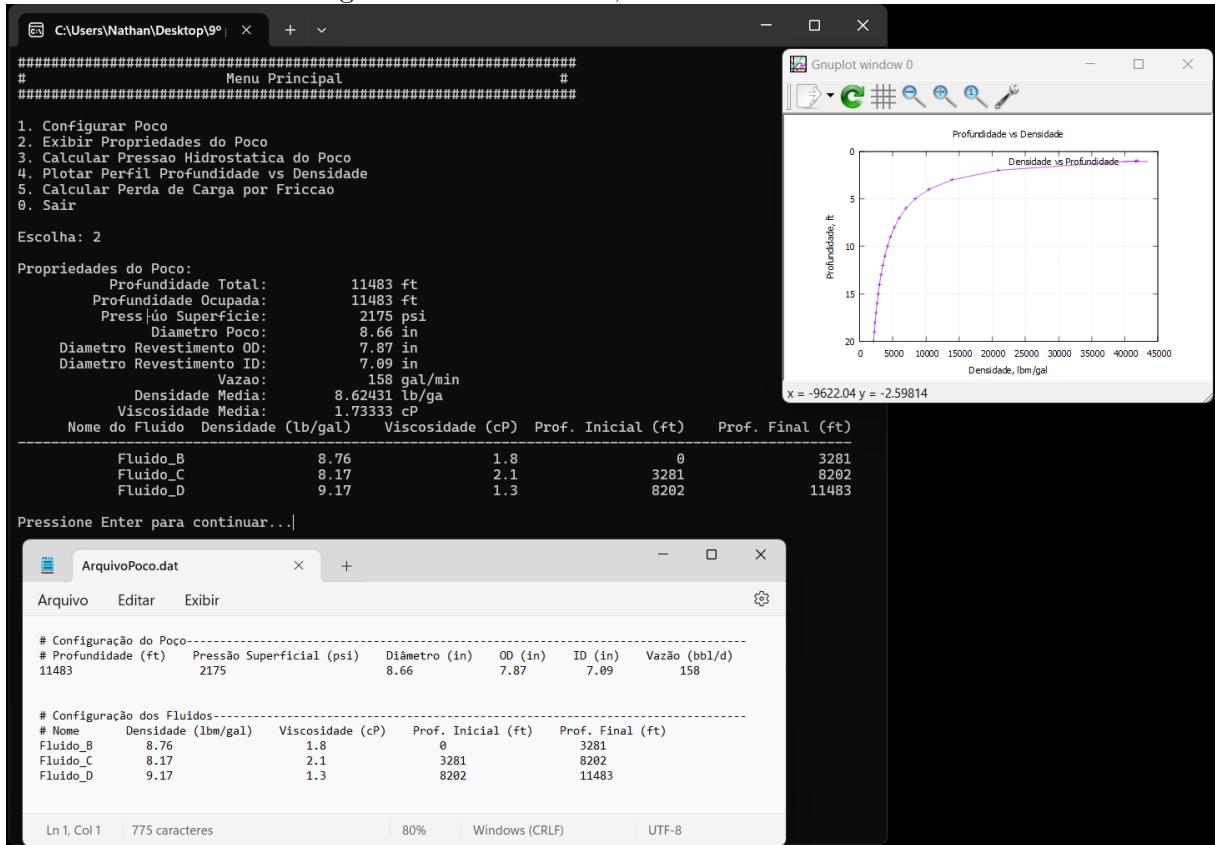
Registro da versão 1.0: versão v1.0 disponível no GitHub

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 7.1.

Figura 7.1: Versão 1.0, interface do software



Fonte: Produzido pelo autor.

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

Versão: 1.1 (19/12/2024)

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

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

Registro da versão 1.1: versão v1.1 disponível no GitHub

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.

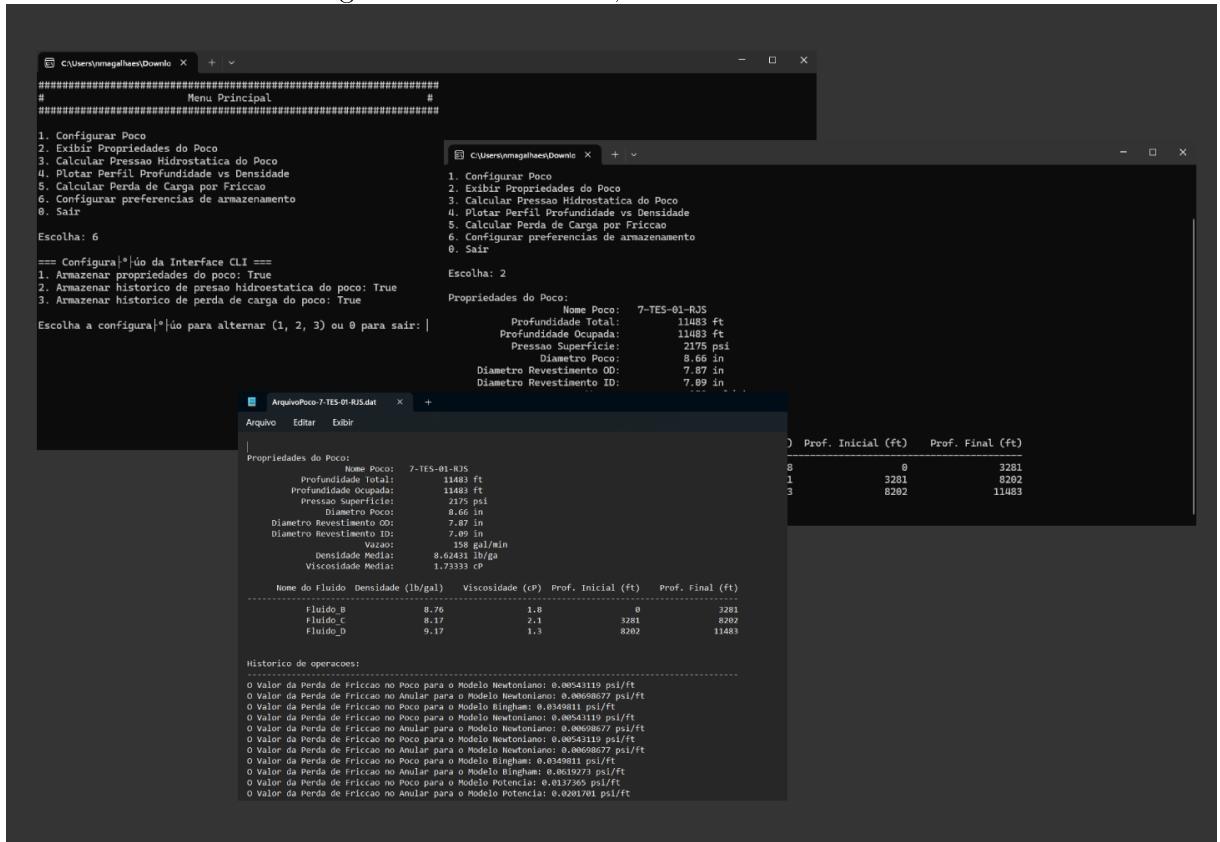
Principais aprimoramentos implementados:

- 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 7.2 exemplifica algumas dessas melhorias implementadas na versão 0.2, evidenciando a evolução da interface textual e das funcionalidades associadas à gestão dos dados.

Figura 7.2: Versão 1.1, interface do software



Fonte: Produzido pelo autor.

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

Versão: 2.0 (25/02/2025)

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

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

Registro da versão 2.0: versão v2.0 disponível no GitHub

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.

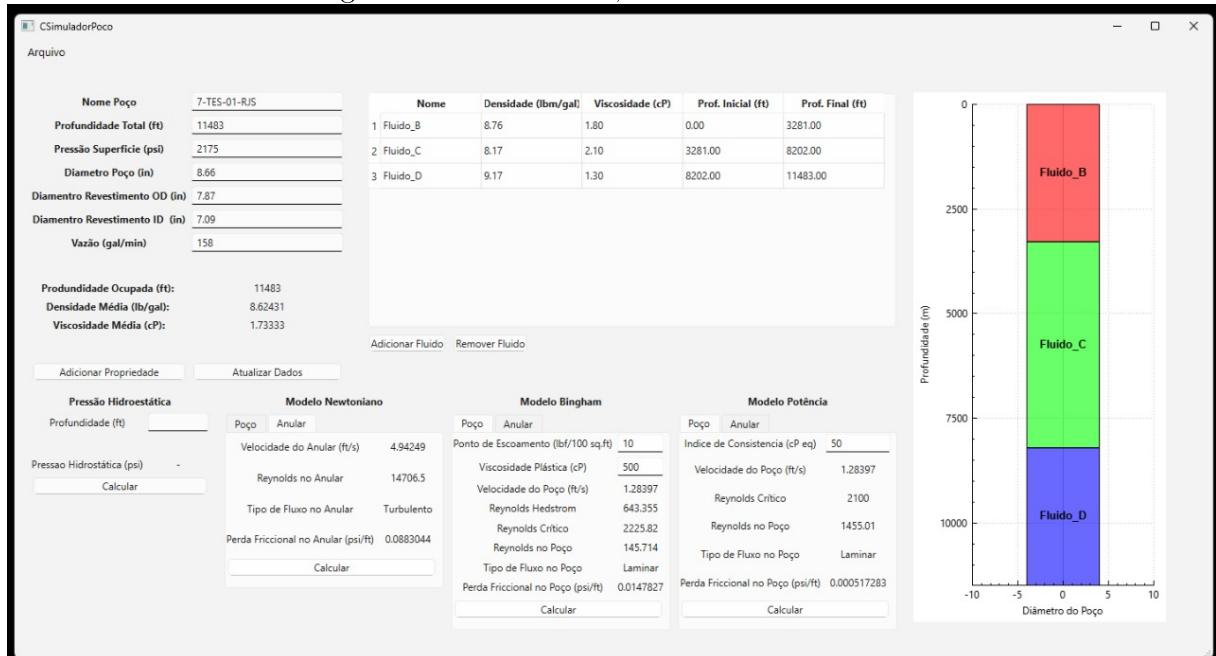
Com a introdução da interface gráfica, o usuário passou a contar com as seguintes facilidades:

- 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 7.3.

Figura 7.3: Versão 2.0, interface do software



Fonte: Produzido pelo autor.

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

Versão: 2.6 (16/04/2025)

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

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

Registro da versão 2.6: versão v2.6 disponível no GitHub

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;

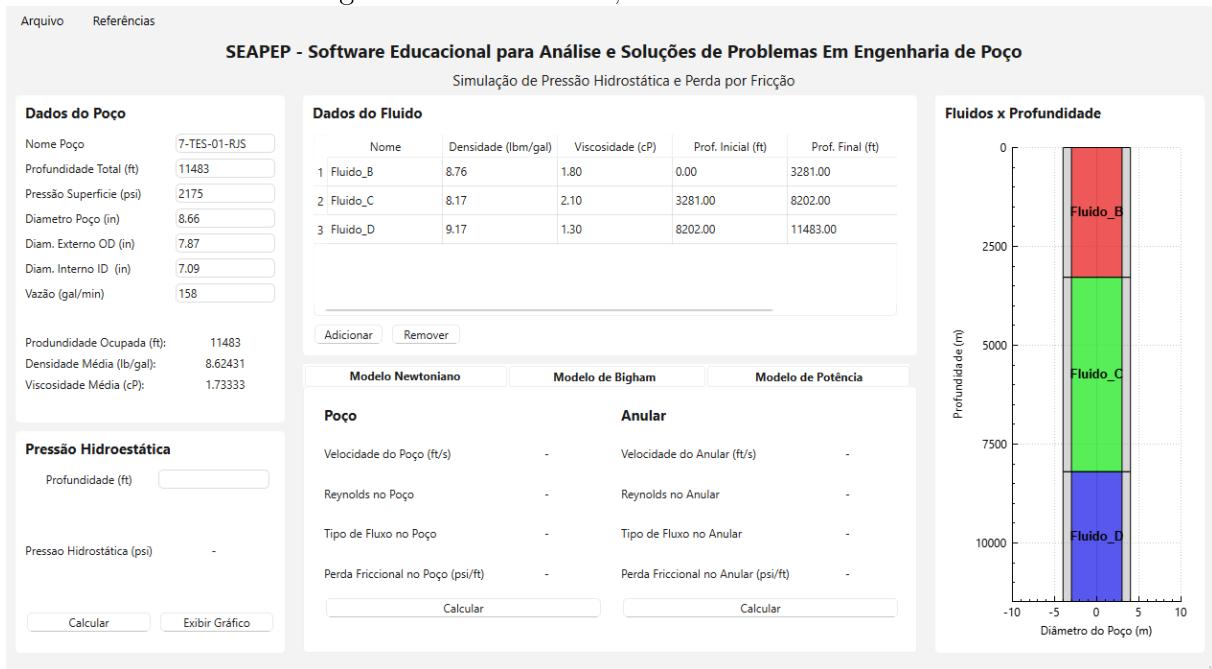
- **Referências:** acesso ao manual do usuário e à documentação dos modelos reológicos implementados; Permite acessar todos os manuais em pdf.
- **Ajuda:** Acesso ao manual do usuário (como instalar e usar o software); Acesso ao manual técnico científico (modelos reológicos implementados); Sobre o software;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

Veja Figura 7.4.

Figura 7.4: Versão 2.6, interface do software



Fonte: Produzido pelo autor.

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

Versão: 3.0 (28/05/2025)

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

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

Registro da versão 3.0: versão v3.0 disponível no GitHub

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 (Figura 7.5).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 (Figura 7.6) 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 (Figura 7.7), 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.

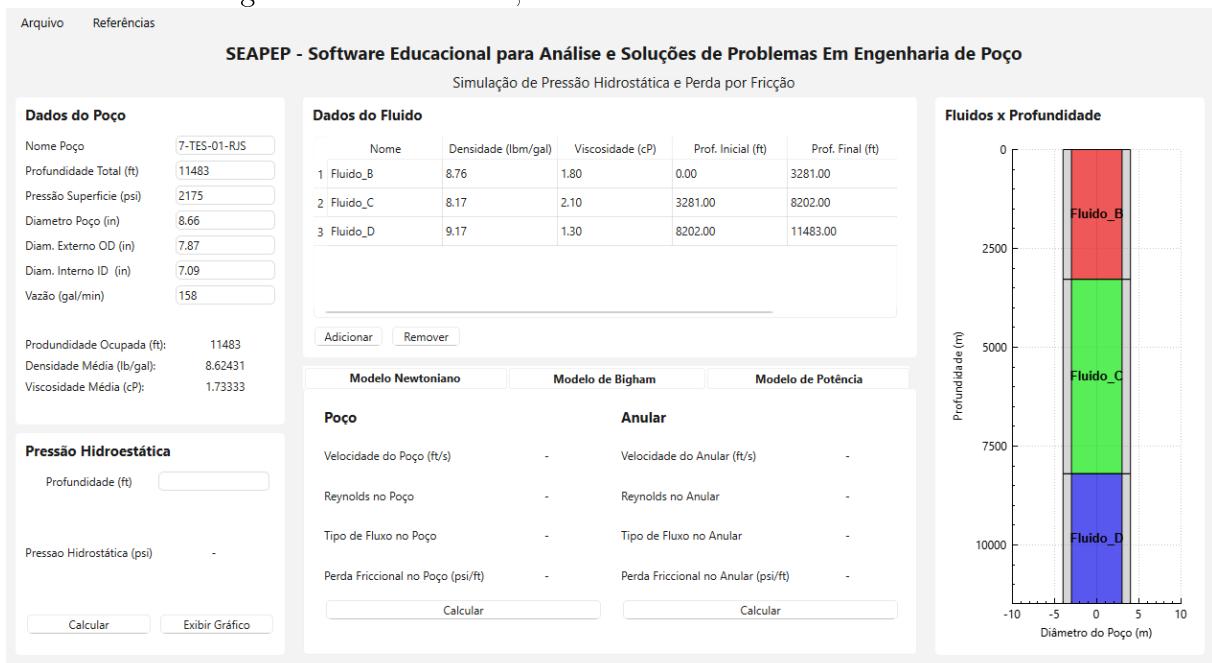
Com essa versão, o sistema passa a incorporar, além da análise hidráulica, uma abordagem mecânica de simulação, ampliando significativamente seu escopo didático e técnico na área de Engenharia de Poço.

Figura 7.5: Versão 3.0, Menu de interface do software



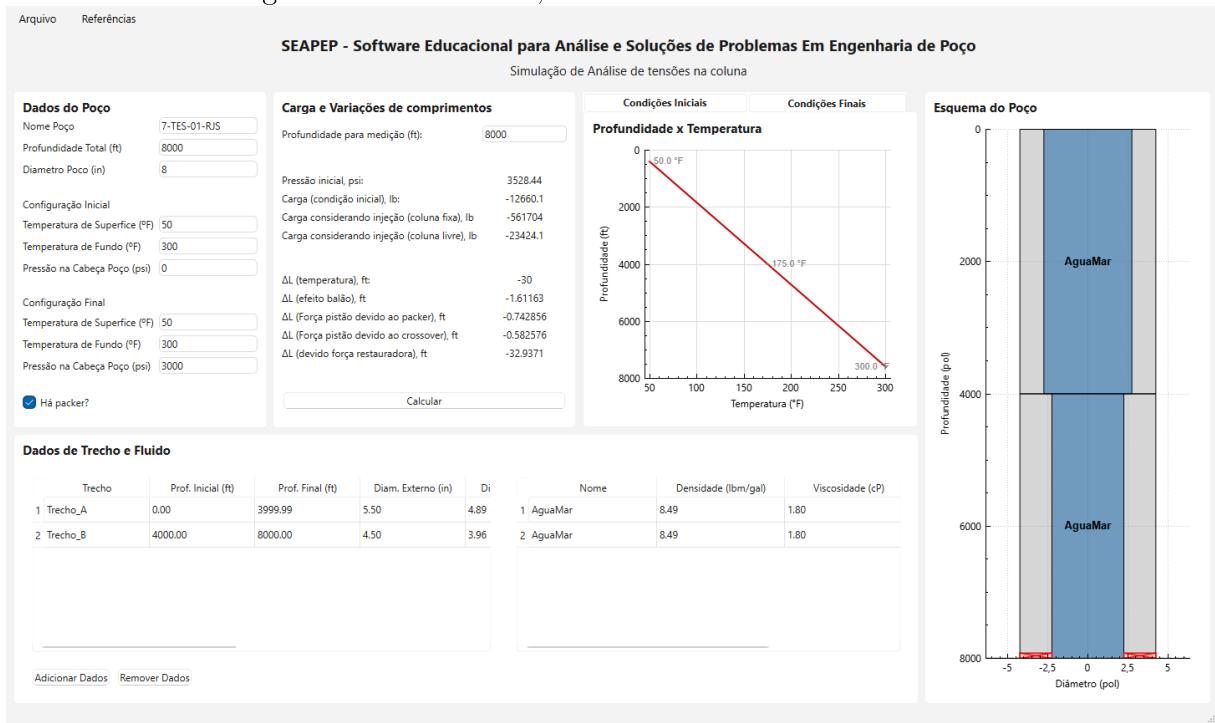
Fonte: Produzido pelo autor.

Figura 7.6: Versão 3.0, interface do módulo 01 do software



Fonte: Produzido pelo autor.

Figura 7.7: Versão 3.0, interface do módulo 02 software



Fonte: Produzido pelo autor.

Capítulo 8

Resultados

Neste capítulo, apresentam-se a validação e os resultados do software. Aplicam-se os conceitos descritos no capítulo de Metodologia.

Inicialmente, o software é 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, utilizam-se exemplos do livro *Applied Drilling Engineering* (Jr. *et al.* (1991)), além dos exercícios aplicados durante a disciplina de Engenharia de Poço no período letivo 2024/01.

Todos os exemplos utilizados neste capítulo encontram-se 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.

Evita-se sobrecarregar este capítulo com métodos repetidos, como diversos testes de pressão hidrostática. Embora durante a fase de validação apliquem-se múltiplos exemplos, aqui apresentam-se apenas casos representativos, com o objetivo de evitar redundâncias e tornar o conteúdo mais direto e comprehensível. Neste capítulo, são apresentados a validação e os resultados do software. Serão aplicados os conceitos descritos no capítulo de Metodologia.

8.1 Metodologia de Validação do Software

A comparação dos cálculos foi 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.

Consideramos 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 Hidrostática

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

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

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

- **Resposta esperada: 136 psig**

$$P_H = 0.052\rho L - P_{formação}$$

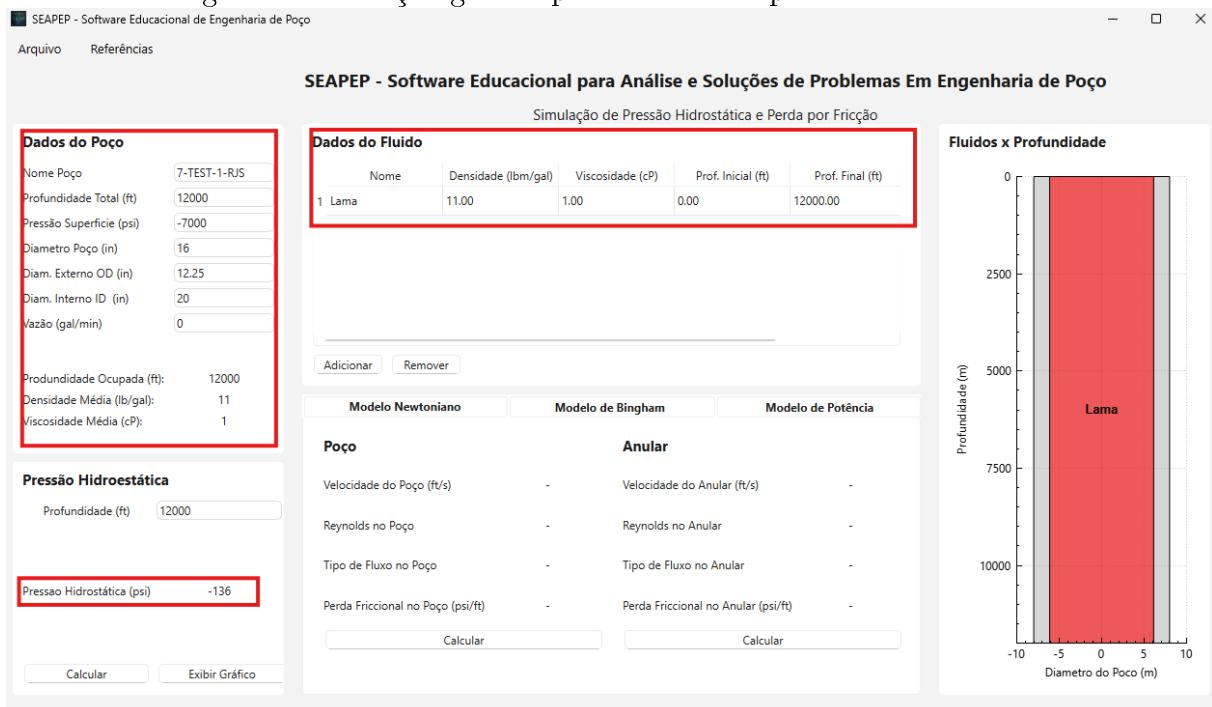
Onde: $\rho=11$ lbm/gal (densidade do fluido); $L=12.000$ ft (profundidade total); $P_{formação}=7000$ psi;

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

$$P_H = -136psig$$

A partir da Figura 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.

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, foi validado o módulo de cálculo de perda de carga por fricção para fluidos Newtonianos, aplicando os conceitos diretamente no tubo de perfuração 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);

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

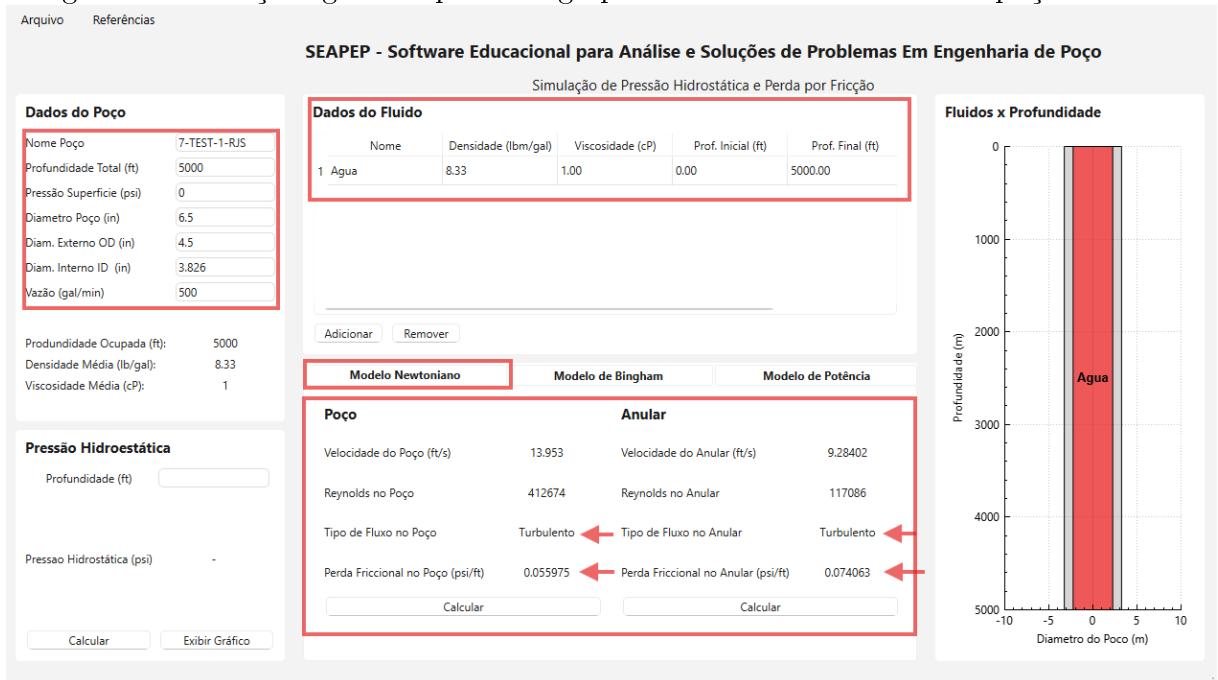
Onde: ρ (densidade); μ (viscosidade); N_{re} (Número de Reynolds);

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

Onde: dp_f (Perda Friccional);

O mesmo cenário é simulado no software (Figura 8.2), utilizando-se os mesmos parâmetros de entrada. A simulação apresenta boa concordância com os resultados de referência. Para o trecho no tubo, o erro percentual em relação ao valor esperado é de 8,99%, enquanto para o anular o desvio é de apenas 1,51%. Esses resultados validam a consistência do modelo Newtoniano implementado, tanto na identificação do regime de escoamento quanto no cálculo das perdas de carga por fricção.

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
- Regime de escoamento: turbulento
- Perda de pressão no anular: 185 psi/1.000 ft

Cálculos no tubo

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

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

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

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

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

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

Cálculos no anular

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

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

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

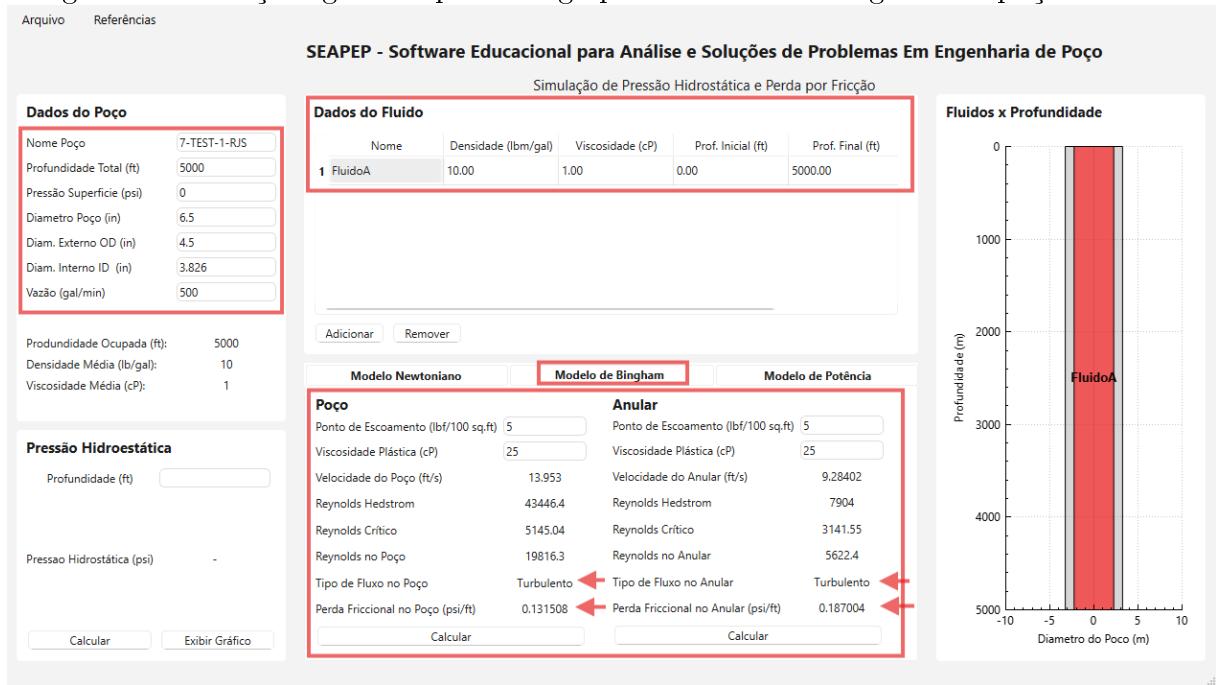
$$N_{re} = \frac{757\rho\bar{v}(d_2 - d_1)}{\mu_p} = \frac{757(10)(9.284)(6.5 - 4.5)}{25} = 5622.39$$

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

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

O teste com fluido Bingham apresenta 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 (Figura 8.3) demonstra capacidade confiável de tratar modelos reológicos mais complexos, indo além do comportamento Newtoniano. Os erros percentuais observados são inferiores a 1,1%, o que reforça a precisão do modelo implementado.

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

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

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

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

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

$$f = 0.006 (\text{fig.4.34, [APPLIED1991]})$$

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

Cálculos no anular

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

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

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

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

$$f = 0.008(\text{fig.4.34, [APPLIED1991]})$$

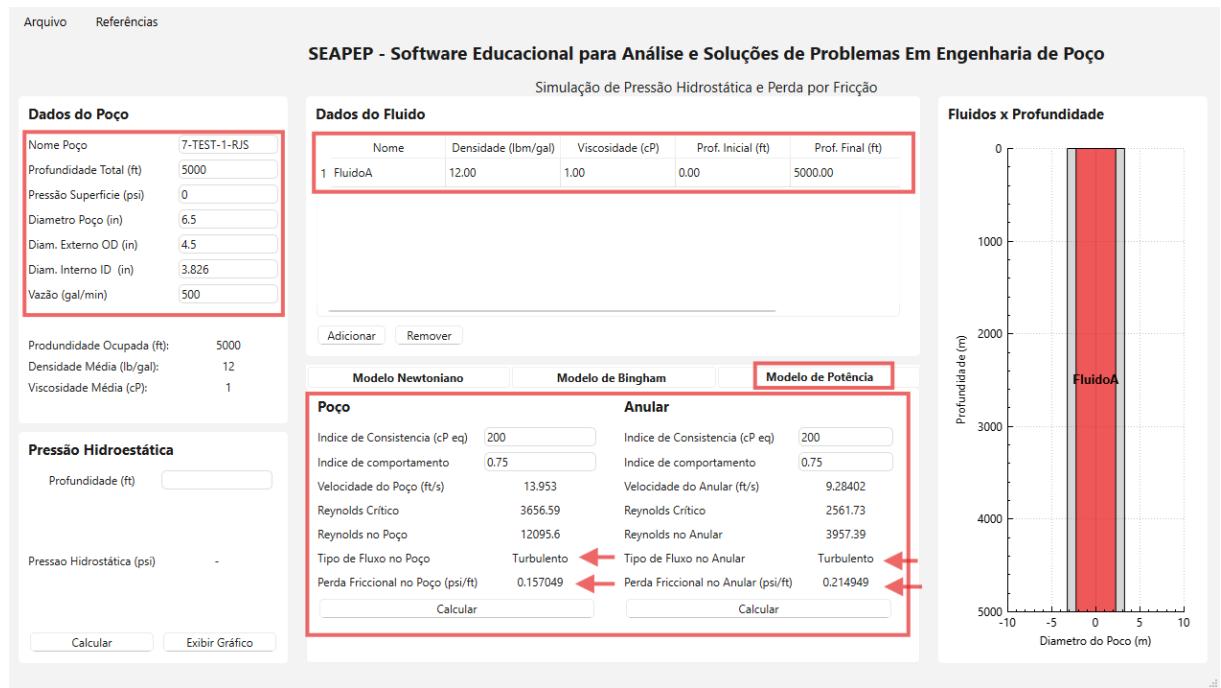
$$\frac{dp_f}{dL} = \frac{f\rho\bar{v}^2}{21.1(d_2 - d_1)} = \frac{0.008(12)(9.284)^2}{21.1(6.5 - 4.5)} = 0.196 \text{ psi/ft}$$

O teste com o modelo da lei da potência apresenta 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.

As pequenas divergências observadas devem-se ao fato de que a resolução analítica exige consulta a tabelas em escala logarítmica, o que pode introduzir erros de leitura ou interpolação. O software, por sua vez, utiliza aproximações lineares com base nos principais pontos da curva log-log, empregando regressão linear para melhor ajuste.

Como ilustrado na Figura 8.4, os erros percentuais observados variam entre 2% e 9%, o que demonstra uma boa aderência do modelo, mesmo em comparação com métodos analíticos mais tradicionais. A ferramenta confirma sua capacidade de tratar modelos reológicos não lineares com eficiência e precisão.

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 ao Packer

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 :

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

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

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

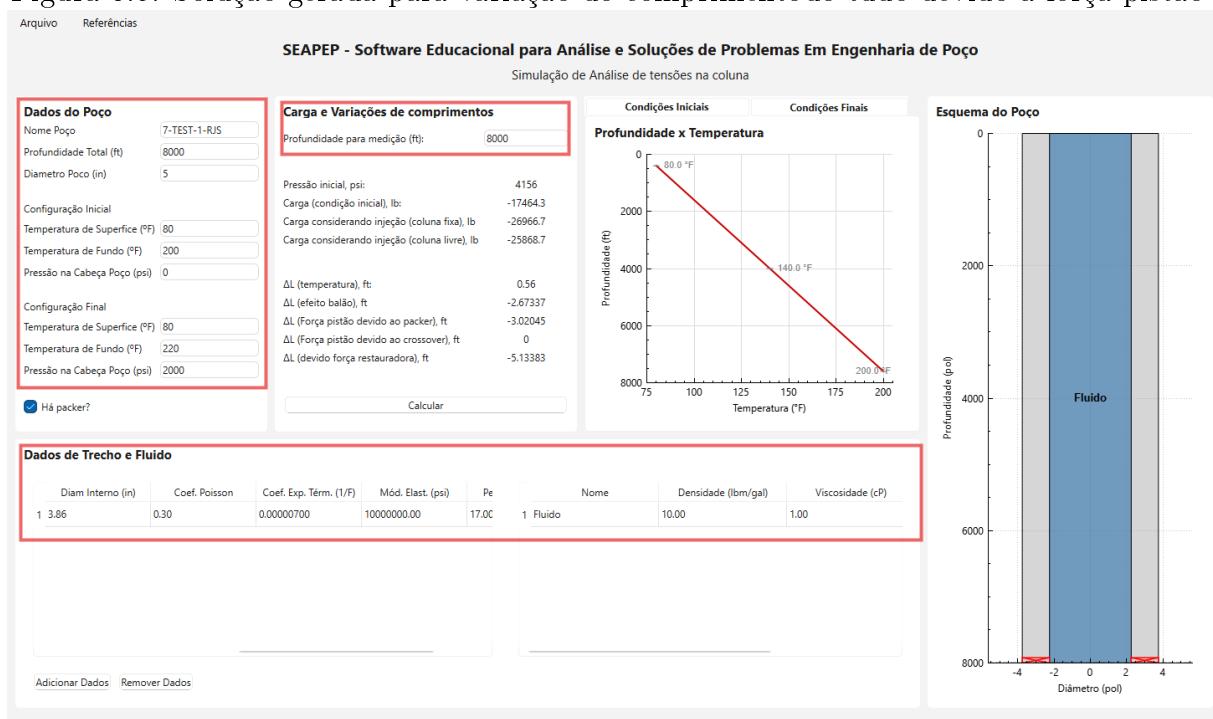
$$A_{in} = \frac{\pi}{4}(d_2^2 - d_1^2) = \frac{\pi}{4}(5^2 - 3.862^2) = 7.92 \text{ in}^2$$

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

$$\Delta L = \frac{\Delta F \cdot L}{E \cdot A_s} = \frac{-15840(8000)}{10.10^6 \left[\frac{\pi}{4} (4.5^2 - 3.862^2) \right]} = 3.04 \text{ ft}$$

A simulação realizada no software (Figura 8.5) resulta em uma variação de comprimento compatível com o valor esperado teoricamente. A deformação é calculada com base na equação de carga axial aplicada ao efeito pistão, considerando o diferencial de área e a variação de pressão induzida pela substituição do fluido. O erro percentual em relação ao valor de referência é inferior a 0,7%, evidenciando a precisão do modelo implementado.

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:

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

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

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

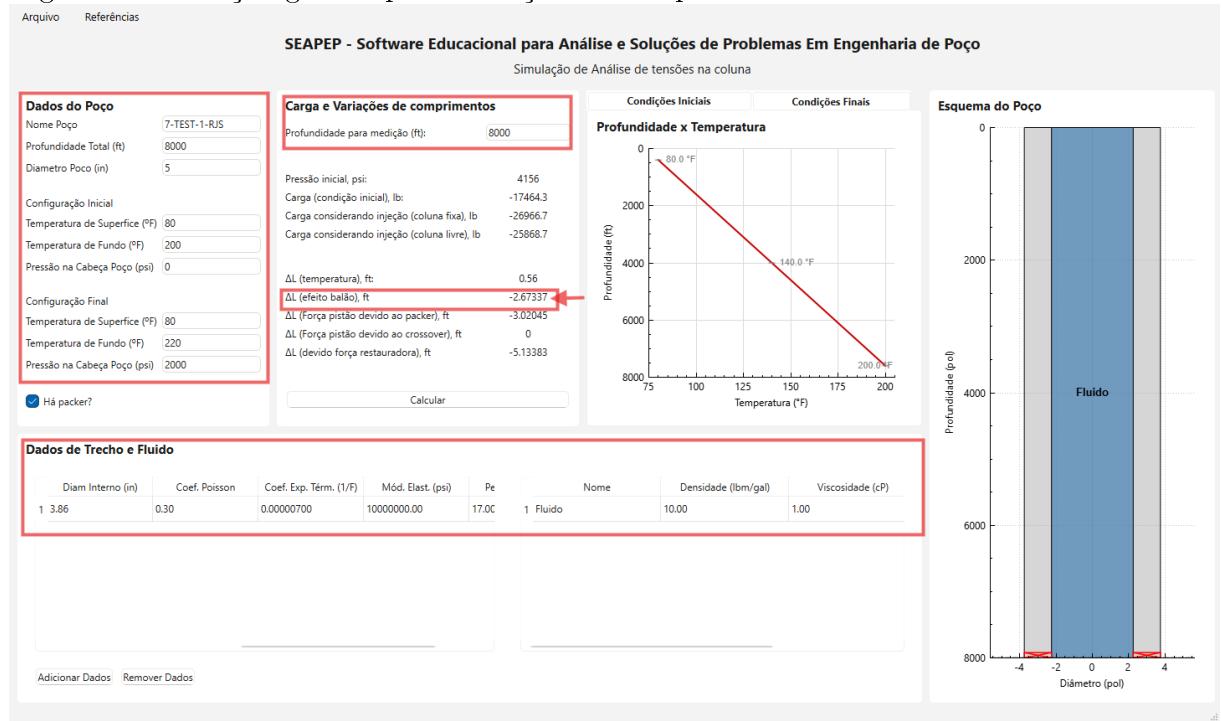
$$A_{in} = \frac{\pi}{4}(d_2^2 - d_1^2) = \frac{\pi}{4}(3.862^2) = 11.71\text{in}^2$$

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

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

O software (Figura 8.6) é capaz de calcular a deformação associada ao efeito balão com base nas pressões envolvidas na troca de fluidos. Os resultados mostram boa concordância com a equação clássica, apresentando erros percentuais inferiores a 3,1%, o que valida a consistência do modelo proposto para esse fenômeno.

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



Fonte: Produzido pelo autor.

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:

$$T_{inicial} = \frac{80 + 200}{2} = 140^{\circ}F$$

$$T_{final} = \frac{80 + 220}{2} = 150^{\circ}F$$

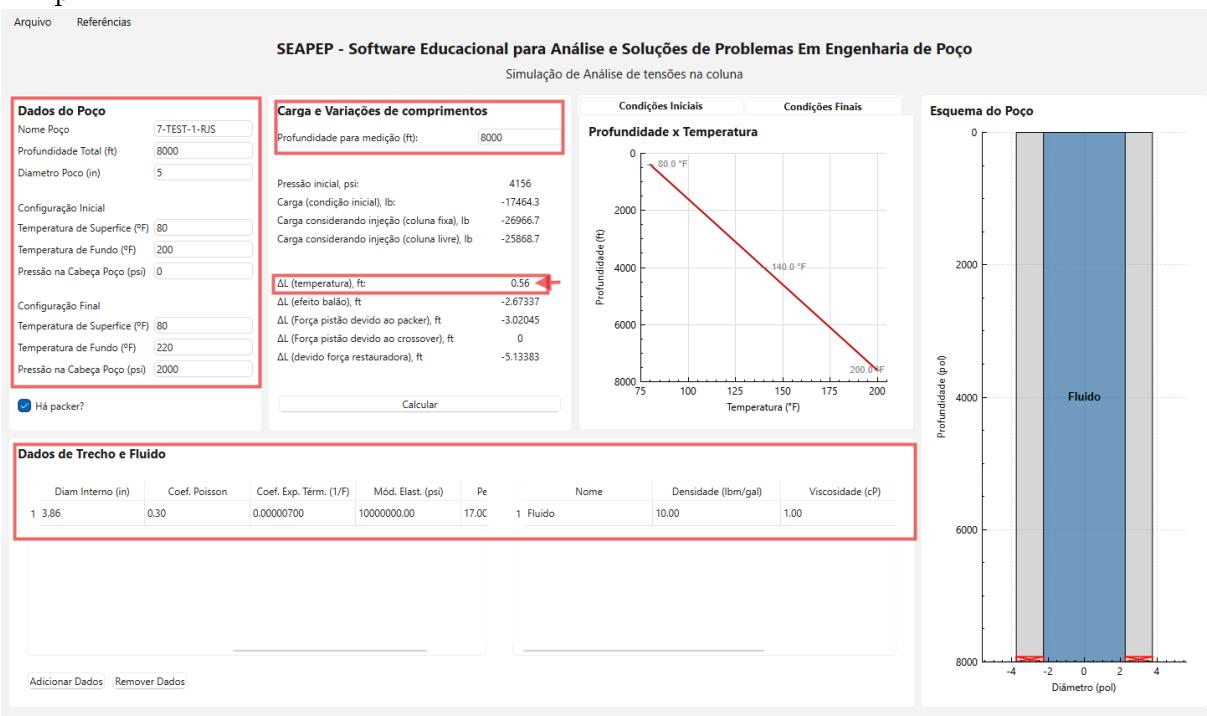
$$\Delta T = 150 - 140 = 10^{\circ}F$$

Logo:

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

A simulação (Figura 8.7) 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.

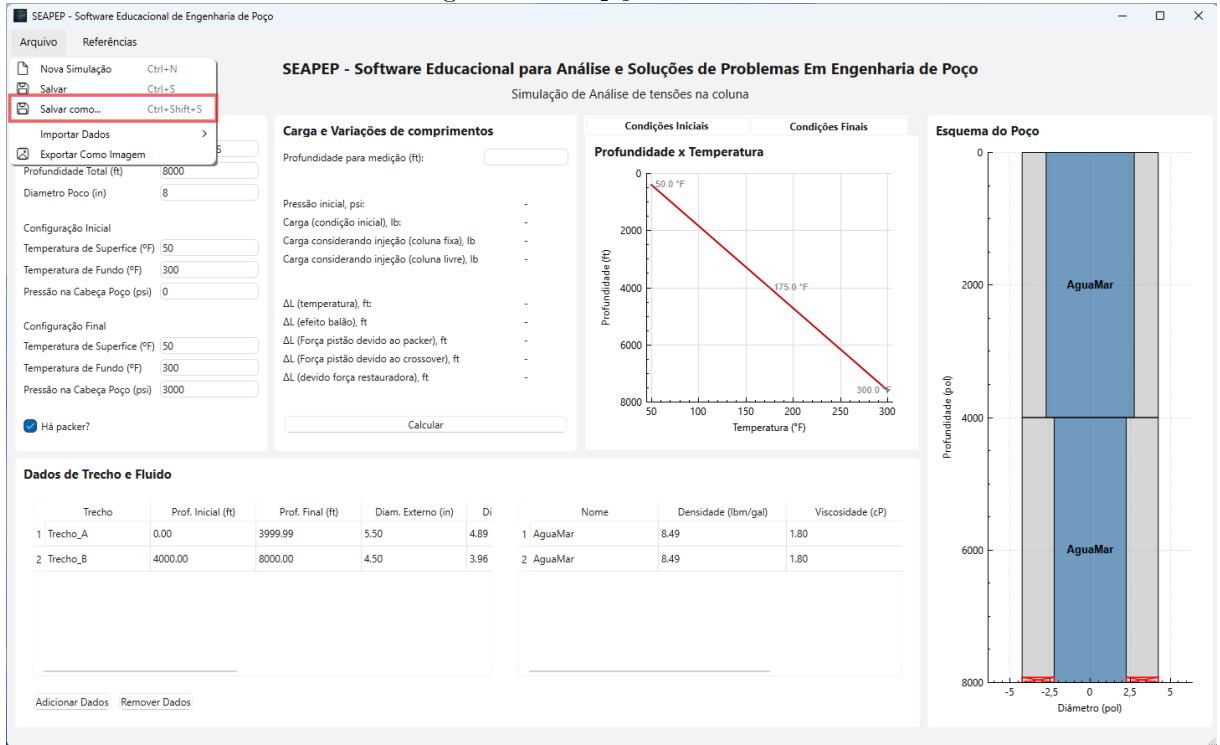
8.10 Validação da Importação e Exportação do Documento (“salvar como...”)

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.

Veja a Figura 8.8 e 8.9.

Figura 8.8: Opção de Salvar



Fonte: Produzido pelo autor.

Figura 8.9: Modelo do Arquivo .dat

# Configuração do Poço	Profundidade (ft)	Diametro do Poço (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) 7-715-01-R35 true	8000	8	0	3800	50	300	50	300			
# Configuração dos Fluidos	Prof. Inicial (ft)	Prof. Final (ft)	Diam. externo (in)	Diam. interno (in)	Coef. Poisson	Coef. Exp. Term. (1/R)	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	4.89 3.96	0.30 0.30	8.8000000000E-001 8.8000000000E-001	38000000.00 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.

Veja a Figura 8.10.

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 dos Testes

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 reproduibilidade, ras-treabilidade e usabilidade da ferramenta em ambientes acadêmicos e operacionais.

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

Capítulo 9

Documentação

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

9.1 Documentação do usuário

O Manual do Usuário é apresentado no Apêndice A - 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*:
github.com/ldsc/ProjetoEngenharia-SoftwareEducacionalParaAnaliseESolucaoDeProblemasEmEngenhariaDePoco
- A documentação foi gerada utilizando o software *Doxygen*:
 - <https://www.doxygen.nl/>

Na Figura 9.1 mostra uma imagem da documentação gerada.

Figura 9.1: Logo e documentação do *software*
SoftwareEducacionalParaAnaliseESolucaoDeProblemasEmEngenhariaDePoco

Main Page Classes Files Search

File List

Here is a list of all files with brief descriptions:

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

Fonte: Produzido pelo autor.

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

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

Main Page Classes Files Search

CSimuladorPoco.h

Go to the documentation of this file.

```

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

```

Fonte: Produzido pelo autor.

Capítulo 10

Conclusão

Ao final deste trabalho, constata-se que os objetivos propostos foram plenamente atingidos. O software educacional desenvolvido cumpre sua função como ferramenta de apoio ao ensino da disciplina de Engenharia de Poço, permitindo simular com fidelidade os principais fenômenos operacionais incluindo o escoamento de fluidos em diferentes regimes, perdas de carga, variações térmicas e deformações mecânicas da coluna.

Os testes realizados demonstraram que os resultados obtidos pelo programa apresentam boa concordância com referências consagradas da literatura técnica, com erros percentuais baixos e dentro de margens aceitáveis para aplicações didáticas. Além disso, a implementação de múltiplos modelos reológicos e efeitos físicos complexos permitiu explorar cenários mais realistas, enriquecendo a compreensão dos conceitos envolvidos.

Mais do que validar cálculos, o projeto possibilitou integrar teoria e prática em uma plataforma acessível, contribuindo para a formação de futuros engenheiros com uma visão mais aplicada da engenharia de poço. Espera-se que o material desenvolvido possa ser utilizado em por alunos fora de sala de aula, em atividades práticas e como base para o desenvolvimento de novas funcionalidades.

Por fim, este trabalho deixa como legado não apenas um software funcional, mas também uma base sólida para futuras pesquisas e aprimoramentos na área, seja por meio da ampliação de funcionalidades, da modernização da interface ou da aplicação em contextos profissionais.

10.1 Sugestões para trabalhos futuros

Com base nos resultados alcançados e nas possibilidades de aprimoramento identificadas ao longo do desenvolvimento deste trabalho, propõem-se as seguintes sugestões para projetos futuros:

- **Expansão da base física do modelo:** inclusão de efeitos transientes, como variações dinâmicas de pressão e temperatura ao longo do tempo, ampliando orealismo das simulações.

- **Simulação de poços não verticais:** implementação de modelos voltados a poços horizontais, desviados ou multisegmentados, com representação espacial mais complexa.
- **Aprimoramento da interface gráfica:** modernização visual, com inclusão de ícones interativos, navegação mais intuitiva e adaptação responsiva para diferentes resoluções de tela.
- **Conversão automática de unidades:** criação de um sistema interno para permitir que o usuário escolha o sistema de unidades (SI, imperial etc.), com conversões automáticas nas entradas e saídas.
- **Geração automática de relatórios:** integração de uma funcionalidade para exportar relatórios personalizados em PDF ou outros formatos, com tabelas, gráficos e interpretações dos resultados.
- **Internacionalização da ferramenta:** adaptação do software para diferentes idiomas, ampliando seu uso para contextos acadêmicos e industriais internacionais.
- **Aplicação de inteligência artificial:** uso de algoritmos para sugerir parâmetros iniciais, prever resultados esperados ou detectar incoerências nos dados de entrada.

Essas propostas visam transformar o protótipo atual em uma plataforma ainda mais robusta, versátil e alinhada às necessidades reais do setor petrolífero e do ensino técnico-científico.

Referências Bibliográficas

- BUENO, André Duarte. 2003. *Programação orientada a objeto com c++*. Novatec. 46
- Jr., Adam T. Bourgoyne, Millheim, Keith E., Chenevert, Martim E., & Jr., F. S. Young. 1991. *Applied drilling engineering*. Society of Petroleum Engineers. 21, 26, 31, 32, 33, 34, 35, 38, 41, 67, 68, 70, 72, 74
- Mitchell, Robert F., & Miska, Stefam Z. 2011. *Fundamentals of drilling engineering*. Society of Petroleum Engineers. 21, 26, 35, 36, 37, 41, 42, 44

Apêndice A

Manual do desenvolvedor

A.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.

A.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"`

A.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)]

A.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>`.

A.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.

A.2 Interface gráfica

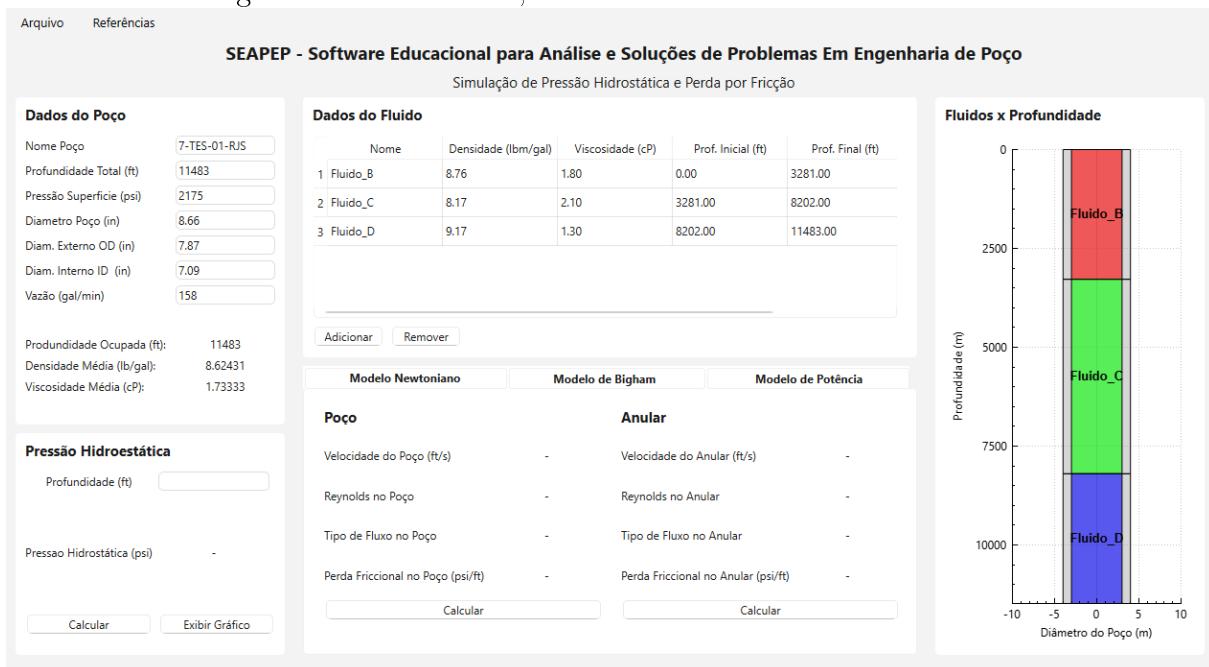
A interface do programa é apresentada nas Figuras: A.1, A.2 e A.3.

Figura A.1: Versão 1.1, Menu de interface do software



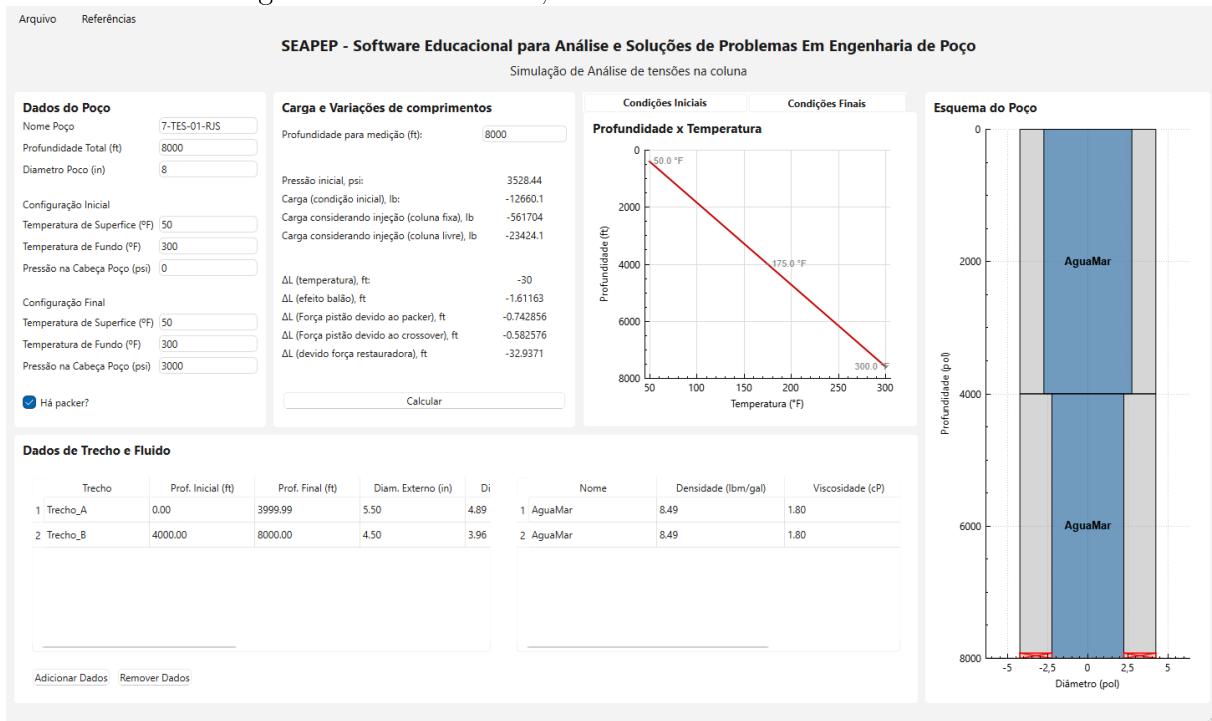
Fonte: Produzido pelo autor.

Figura A.2: Versão 1.1, interface do modulo 01 do software



Fonte: Produzido pelo autor.

Figura A.3: Versão 1.1, interface do modulo 02 software



Fonte: Produzido pelo autor.

A Figura 1 (A.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 (A.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 (A.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.

A.3 Funcionalidades

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.

A.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.

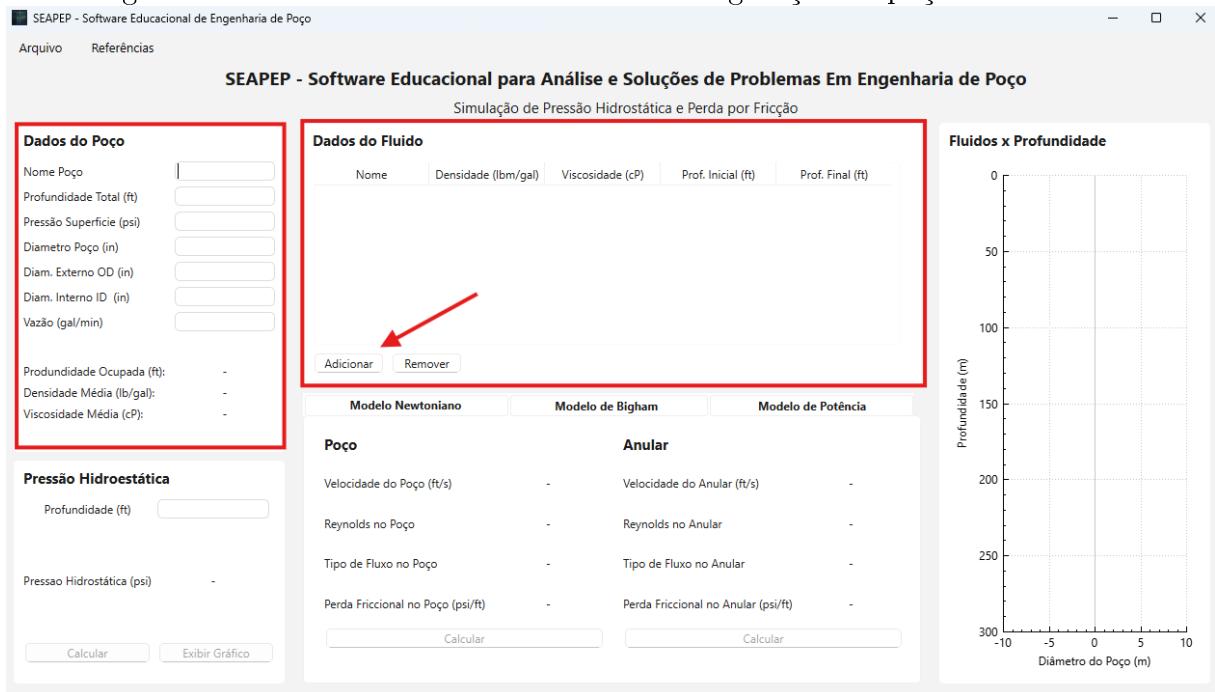
Configuração Manual via Interface Gráfica

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 A.4 ilustra o caminho no menu da interface gráfica para acessar essas funcionalidades.

Figura A.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 figura A.5. apresenta um exemplo de arquivo .dat estruturado corretamente e compatível com o sistema.

Figura A.5: Exemplo de estrutura de arquivo .dat para importação de dados

```

ArquivoPoco.dat
Arquivo Editar Exibir

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

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


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

```

Fonte: Produzido pelo autor.

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

Veja a FiguraA.6.

Figura A.6: Exemplo de estrutura de arquivo .dat para importação de dados

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

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

Ln 11, Col 1 | 1.333 caracteres
```

Fonte: Produzido pelo autor.

Apêndice B

Ciclos Construção - Implementação

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

B.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 1 o arquivo com código da função *main*.
frame

Listing 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 de Processo");
21    w.show();
22
23    return app.exec();
24 }
```

Fonte: produzido pelo autor.

Apresenta-se na listagem 2 o arquivo de cabeçalho da classe CSimuladorReologico. frame

Listing 2: Arquivo de implementação da classe CSimuladorReologico

```
1 #ifndef CSIMULADORREOLOGICO_H
2 #define CSIMULADORREOLOGICO_H
3
4 #include <QMainWindow>
5 #include <QTableWidgetItem>
6
7 // inclui os arquivos com os modelos fisicos e estrutura do poco
8 #include "CObjetoPoco.h"
9 #include "CTrechoTubulacao.h"
10 #include "CModeloNewtoniano.h"
11 #include "CModeloBingham.h"
12 #include "CModeloPotencia.h"
13
14 namespace Ui {
15 class CSimuladorReologico ;
16 }
17
18 // classe principal da interface grafica do simulador reológico
19 // herda de QMainWindow pois usa os recursos visuais do Qt
20 class CSimuladorReologico : public QMainWindow
21 {
22     Q_OBJECT
23
24 public:
25     // construtor e destrutor da interface
26     CSimuladorReologico(QWidget *parent = nullptr);
27     ~CSimuladorReologico();
28     QString nomeArquivo = "";
29     QString caminhoArquivo = "";
30
31 private slots:
32     // atualiza os dados na interface quando ha alteracao
33     void AtualizarDados();
34
35     // botoes da interface para adicionar e remover fluido
36     void on_btnAdicionarFluido_clicked();
37     void on_btnRemoverFluido_clicked();
38
39     // botao que calcula a pressao hidrostatica com base nos dados do poco e do fluido
40     void on_btnCalcularPressaoHidroestatica_clicked();
41
42     // botoes que executam os calculos usando os diferentes modelos reológicos
43     void on_btnCalcularModeloNewtonianoAnular_clicked();
44     void on_btnCalcularModeloNewtonianoPoco_clicked();
45     void on_btnCalcularModeloBighamPoco_clicked();
46     void on_btnCalcularModeloBighamAnular_clicked();
47     void on_btnCalcularModeloPotenciaPoco_clicked();
48     void on_btnCalcularModeloPotenciaAnular_clicked();
49
50     // funcao que gera o grafico do poco (perfil visual da configuracao)
51     void makePlotPoco();
52
53     // edita uma linha da tabela de fluidos ou trechos do poco
```

```

54     void EditarLinhaTabela(int row);
55
56     // edita os dados gerais do poco como nome, profundidade etc
57     void EditarDadosPoco();
58
59     // opcoes do menu superior da interface (arquivo, sobre, exportar etc)
60     void on_actionExportar_como_Imagen_triggered();
61     void on_actionSobre_o_Programa_triggered();
62     void on_actionSalvar_Como_triggered();
63     void SalvarArquivo(bool salvarComo);
64     void on_actionArquivo_dat_triggered();
65     void on_actionSalvar_triggered();
66     void on_actionNova_Simula_o_triggered();
67
68     // exibe o grafico da pressao hidroestatica ao longo da profundidade
69     void on_btnExibirGraficoPressaoHidroestatica_clicked();
70
71
72     //getters
73     QString NomeArquivo() { return nomeArquivo; }
74     QString CaminhoArquivo() { return caminhoArquivo; }
75
76
77     //setters
78     void NomeArquivo(QString nome) { nomeArquivo = nome; }
79     void CaminhoArquivo(QString caminho) { caminhoArquivo = caminho; }
80
81
82
83     void on_actionAjuda_triggered();
84
85     void on_actionSobre_os_Modelos_Reologicos_triggered();
86
87 private:
88     Ui::CSimuladorReologico *ui; // interface grafica do Qt
89
90     // ponteiros para os objetos principais usados nas simulacoes
91     std::shared_ptr<COBJETOPOCO> poco; // dados do poco inteiro
92     std::shared_ptr<CTRECHOPOCO> trechoPoco; // trecho especifico do
93         poco
94     std::shared_ptr<CFLUIDO> fluido; // fluido que circula no
95         poco
96
97     // ponteiros para os modelos reologicos que vao ser usados nos calculos
98     std::shared_ptr<CMODELONEWTONIANO> modeloNewtoniano = nullptr;
99     std::shared_ptr<CMODELOBINGHAM> modeloBingham = nullptr;
100    std::shared_ptr<CMODELOPOTENCIA> modeloPotencia = nullptr;
101};

101#endif // CSIMULADORREOLOGICO_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 3 a implementação da classe CSimuladorReologico.
frame

Listing 3: Arquivo de implementação da classe CSimuladorReologico

```

1 #include "CSimuladorReologico.h"
2 #include "ui_CSIMULADORREOLOGICO.h"

```

```

3 #include "CJanelaAdicionarFluido.h"
4 #include "CJanelaGraficoPressaoHidroestatica.h"
5 #include "CJanelaSobreSoftware.h"
6
7 #include <QDesktopServices>
8 #include <QUrl>
9 #include <QPushButton>
10 #include <iostream>
11 #include <fstream>
12 #include <sstream>
13 #include <QApplication>
14 #include <QFileDialog>
15 #include <QString>
16 #include <QMessageBox>
17 #include <QMap>
18 #include <QColor>
19 #include <QDesktopServices>
20 #include <QUrl>
21
22 CSimuladorReologico::CSimuladorReologico(QWidget *parent)
23     : QMainWindow(parent)
24     , ui(new Ui::CSimuladorReologico)
25 {
26     ui->setupUi(this);
27
28     makePlotPoco(); // gera o grafico inicial do poco (mesmo vazio)
29
30     // ativa edicao de celulas da tabela com clique duplo ou tecla Enter
31     ui->tblFluidos->setEditTriggers(QAbstractItemView::DoubleClicked | QAbstractItemView
32                                         ::EditKeyPressed);
33
34     // conecta a edicao de celulas na tabela a uma funcao que atualiza os dados do fluido
35     QObject::connect(ui->tblFluidos, &QTableWidget::cellChanged, this, &
36                       CSimuladorReologico::EditarLinhaTabela);
37
38     // conecta a alteracao dos campos de entrada para atualizar o objeto do poco
39     QObject::connect(ui->editNomePoco, &QLineEdit::editingFinished, this, &
40                       CSimuladorReologico::EditarDadosPoco);
41     QObject::connect(ui->editProfundidadeTotal, &QLineEdit::editingFinished, this, &
42                       CSimuladorReologico::EditarDadosPoco);
43     QObject::connect(ui->editPressaoSuperficie, &QLineEdit::editingFinished, this, &
44                       CSimuladorReologico::EditarDadosPoco);
45     QObject::connect(ui->editDiametroPoco, &QLineEdit::editingFinished, this, &
46                       CSimuladorReologico::EditarDadosPoco);
47     QObject::connect(ui->editDiametroOD, &QLineEdit::editingFinished, this, &
48                       CSimuladorReologico::EditarDadosPoco);
49     QObject::connect(ui->editDiametroID, &QLineEdit::editingFinished, this, &
50                       CSimuladorReologico::EditarDadosPoco);
51     QObject::connect(ui->editVazao, &QLineEdit::editingFinished, this, &
52                       CSimuladorReologico::EditarDadosPoco);
53
54     // desativa todos os botoes de calculo inicialmente (s ativa quando preencher dados
55     // do poco)
56     ui->btnCalcularPressaoHidroestatica->setEnabled(false);
57     ui->btnCalcularModeloNewtonianoPoco->setEnabled(false);
58     ui->btnCalcularModeloNewtonianoAnular->setEnabled(false);
59     ui->btnCalcularModeloBighamPoco->setEnabled(false);
60     ui->btnCalcularModeloBighamAnular->setEnabled(false);
61     ui->btnCalcularModeloPotenciaPoco->setEnabled(false);

```

```

53     ui->btnCalcularModeloPotenciaAnular->setEnabled(false);
54     ui->btnExibirGraficoPressaoHidroestatica->setEnabled(false);
55
56     // centraliza a janela no meio da tela do computador
57     QScreen *screen = QGuiApplication::primaryScreen();
58     QRect screenGeometry = screen->geometry();
59     int x = (screenGeometry.width() - this->width()) / 2;
60     int y = (screenGeometry.height() - this->height()) / 2;
61     this->move(x, y);
62 }
63
64 CSimuladorReologico::~CSimuladorReologico()
65 {
66     delete ui;
67 }
68
69
70 void CSimuladorReologico::EditarDadosPoco() {
71     // pega os valores digitados nos campos da interface
72     QString nome = ui->editNomePoco->text();
73
74     // cria variaveis booleanas pra saber se a conversao pra double deu certo
75     bool ok1, ok2, ok3, ok4, ok5, ok6;
76
77     // converte os textos digitados em numeros reais (double)
78     double profund = ui->editProfundidadeTotal->text().toDouble(&ok1);
79     double pressao = ui->editPressaoSuperficie->text().toDouble(&ok2);
80     double diametro = ui->editDiametroPoco->text().toDouble(&ok3);
81     double OD = ui->editDiametroOD->text().toDouble(&ok4);
82     double ID = ui->editDiametroID->text().toDouble(&ok5);
83     double vazao = ui->editVazao->text().toDouble(&ok6);
84
85     // verifica se todos os valores sao validos e o nome nao ta vazio
86     if (!nome.isEmpty() && ok1 && ok2 && ok3 && ok4 && ok5 && ok6) {
87
88         // se ainda nao tiver criado o objeto do poco, entao cria agora
89         if (!poco) {
90             poco = std::make_unique<COBJETOPOCO>(
91                 COBJETOPOCO::CriarParaModulo01(nome.toStdString(), profund, pressao,
92                     diametro, OD, ID, vazao)
93             );
94
95             // ativa os botoes de calculo, ja que agora temos um poco valido
96             ui->btnCalcularPressaoHidroestatica->setEnabled(true);
97             ui->btnCalcularModeloNewtonianoPoco->setEnabled(true);
98             ui->btnCalcularModeloNewtonianoAnular->setEnabled(true);
99             ui->btnCalcularModeloBighamPoco->setEnabled(true);
100            ui->btnCalcularModeloBighamAnular->setEnabled(true);
101            ui->btnCalcularModeloPotenciaPoco->setEnabled(true);
102            ui->btnCalcularModeloPotenciaAnular->setEnabled(true);
103            ui->btnExibirGraficoPressaoHidroestatica->setEnabled(true);
104
105            ui->statusbar->showMessage("Poco criado com sucesso!");
106        } else {
107            // se o poco ja existe, entao atualiza os valores com os novos que o usuario
108            // digitou
109            poco->NomePoco(nome.toStdString());
110            poco->ProfundidadeTotal(profund);
111            poco->PressaoSuperficie(pressao);
112            poco->DiametroPoco(diametro);

```

```

111         poco->DiametroRevestimentoOD();
112         poco->DiametroRevestimentoID();
113         poco->Vazao(vazao);
114
115         ui->statusbar->showMessage("Dados do poco atualizados com sucesso!");
116     }
117
118     // atualiza a interface e os calculos apos alterar os dados
119     AtualizarDados();
120 }
121 }
122
123 void CSimuladorReologico::EditarLinhaTabela(int row)
124 {
125     // valida se o indice da linha e valido
126     if (row < 0 || row >= poco->Trechos().size()) return;
127
128     // pega o trecho e o fluido da linha correspondente
129     CTrechoPoco* trecho = poco->Trechos().at(row);
130     CFluido* fluido = trecho->Fluido();
131
132     if (!fluido) return;
133
134     // atualiza os dados do fluido com base na tabela de fluidos
135     fluido->Nome(ui->tblFluidos->item(row, 0)->text().toString());
136     fluido->Densidade(ui->tblFluidos->item(row, 1)->text().toDouble());
137     fluido->Viscosidade(ui->tblFluidos->item(row, 2)->text().toDouble());
138
139     // atualiza profundidades com base na tabela de trechos (onde elas estao de verdade)
140     trecho->ProfundidadeInicial(ui->tblFluidos->item(row, 3)->text().toDouble()); //
141         coluna 1 = Profund. inicial
142     trecho->ProfundidadeFinal(ui->tblFluidos->item(row, 4)->text().toDouble()); // //
143         coluna 2 = Profund. final
144
145     // atualiza valores calculados
146     AtualizarDados();
147
148 }
149
150
151 void CSimuladorReologico::AtualizarDados()
152 {
153     // desativa os sinais da tabela temporariamente pra evitar chamadas de funcoes
154     // durante a atualizacao
155     ui->tblFluidos->blockSignals(true);
156
157     // verifica se o objeto poco ja foi criado (ou seja, se ja tem dados carregados)
158     if (poco) {
159         // atualiza os campos de entrada com os valores do objeto poco
160         ui->editNomePoco->setText(QString::fromStdString(poco->NomePoco()));
161         ui->editProfundidadeTotal->setText(QString::number(poco->ProfundidadeTotal()));
162         ui->lbnProfundidadeOcupada->setText(QString::number(poco->ProfundidadeOcupada()));
163         ;
164         ui->editPressaoSuperficie->setText(QString::number(poco->PressaoSuperficie()));
165         ui->editDiametroPoco->setText(QString::number(poco->DiametroPoco()));
166         ui->editDiametroOD->setText(QString::number(poco->DiametroRevestimentoOD()));
167         ui->editDiametroID->setText(QString::number(poco->DiametroRevestimentoID()));
168         ui->editVazao->setText(QString::number(poco->Vazao()));

```

```

167     ui->lbnDensidadeMedia ->setText(QString::number(poco->DensidadeEfetivaTotal()));
168     ui->lbnViscosidadeMedia ->setText(QString::number(poco->ViscosidadeEfetivaTotal()))
169         );
170
171         // atualiza o numero de linhas da tabela de fluidos de acordo com os trechos
172         cadastrados no poco
173         ui->tblFluidos ->setRowCount(static_cast<int>(poco->Trechos().size()));
174
175         // percorre os trechos do poco e preenche cada linha da tabela com os dados do
176         fluido
177         int row = 0;
178         for (const auto& trecho : poco->Trechos()) {
179             ui->tblFluidos->setItem(row, 0, new QTableWidgetItem(QString::fromStdString(
180                 trecho->Fluido()->Nome())));
181             ui->tblFluidos->setItem(row, 1, new QTableWidgetItem(QString::number(trecho->
182                 Fluido()->Densidade(), 'f', 2)));
183             ui->tblFluidos->setItem(row, 2, new QTableWidgetItem(QString::number(trecho->
184                 Fluido()->Viscosidade(), 'f', 2)));
185             ui->tblFluidos->setItem(row, 3, new QTableWidgetItem(QString::number(trecho->
186                 ProfundidadeInicial(), 'f', 2)));
187             ui->tblFluidos->setItem(row, 4, new QTableWidgetItem(QString::number(trecho->
188                 ProfundidadeFinal(), 'f', 2)));
189             ++row;
190         }
191
192         // atualiza o grafico visual do poco com base nos trechos e profundidades
193         makePlotPoco();
194     }
195
196     // reativa os sinais da tabela agora que a atualizacao terminou
197     ui->tblFluidos->blockSignals(false);
198 }
199
200 void CSimuladorReologico::on_btnAdicionarFluido_clicked()
201 {
202     // se o poco ainda nao foi criado, nao permite adicionar fluido
203     if (!poco) {
204         QMessageBox::warning(this, "Erro", "As propriedades do poco precisam estar
205             preenchidas!");
206     }
207     else {
208         // abre a janela onde o usuario preenche os dados do novo fluido
209         CJanelaAdicionarFluido JanelaFluido;
210         JanelaFluido.exec(); // abre em modo bloqueante (espera fechar)
211
212         // so continua se todos os campos foram preenchidos
213         if (JanelaFluido.NomeFluido() != "" &&
214             JanelaFluido.Densidade() != "" &&
215             JanelaFluido.Viscosidade() != "" &&
216             JanelaFluido.ProfundidadeInicial() != "" &&
217             JanelaFluido.ProfundidadeFinal() != "") {
218
219             // adiciona uma nova linha na tabela da interface
220             int numLinhas = ui->tblFluidos->rowCount();
221             ui->tblFluidos->insertRow(numLinhas);
222             ui->tblFluidos->setItem(numLinhas, 0, new QTableWidgetItem(JanelaFluido.
223                 NomeFluido()));
224             ui->tblFluidos->setItem(numLinhas, 1, new QTableWidgetItem(JanelaFluido.
225                 Densidade()));
226             ui->tblFluidos->setItem(numLinhas, 2, new QTableWidgetItem(JanelaFluido.

```

```

        Viscosidade()));
216    ui->tblFluidos->setItem(numLinhas, 3, new QTableWidgetItem(JanelaFluido.
217                                ProfundidadeInicial()));
218    ui->tblFluidos->setItem(numLinhas, 4, new QTableWidgetItem(JanelaFluido.
219                                ProfundidadeFinal()));
220
221    // converte os valores da janela pra tipos numericos
222    std::string nome = JanelaFluido.NomeFluido().toStdString();
223    double densidade = JanelaFluido.Densidade().toDouble();
224    double viscosidade = JanelaFluido.Viscosidade().toDouble();
225    double profundInicial = JanelaFluido.ProfundidadeInicial().toDouble();
226    double profundFinal = JanelaFluido.ProfundidadeFinal().toDouble();
227
228    // cria o novo fluido e trecho com base nas informacoes
229    auto fluido = std::make_unique<CFluido>(nome, densidade, viscosidade);
230    auto trechoPoco = std::make_unique<CTrechoPoco>(profundInicial, profundFinal,
231                                                std::move(fluido));
232    poco->AdicionarTrechoPoco(std::move(trechoPoco)); // adiciona ao poco
233
234    AtualizarDados(); // atualiza a interface com o novo trecho
235    ui->statusbar->showMessage("Fluido adicionado com sucesso!");
236}
237
238 void CSimuladorReologico::on_btnRemoverFluido_clicked()
239 {
240     int linhaSelecionada = ui->tblFluidos->currentRow();
241
242     // verifica se alguma linha foi selecionada
243     if (linhaSelecionada >= 0) {
244         QTableWidgetItem* item = ui->tblFluidos->item(linhaSelecionada, 0); // nome do
245         fluido
246
247         if (item) {
248             // confirma se o usuario quer mesmo remover o fluido
249             QMessageBox::StandardButton resposta = QMessageBox::question(
250                 this,
251                 "",
252                 "Tem certeza que deseja remover o fluido?",
253                 QMessageBox::Yes | QMessageBox::No
254             );
255
256             // se o usuario confirmar, remove o trecho do objeto poco e da tabela
257             if (resposta == QMessageBox::Yes) {
258                 QString nomeFluido = item->text();
259                 ui->tblFluidos->removeRow(linhaSelecionada);
260                 poco->RemoverFluidoPoco(nomeFluido.toStdString());
261                 AtualizarDados();
262                 ui->statusbar->showMessage("Fluido removido com sucesso!");
263             }
264         } else {
265             // caso nenhuma linha esteja selecionada
266             QMessageBox::warning(this, "Erro", "Selecione uma linha para deletar.");
267         }
268     }
269 }
270
271 void CSimuladorReologico::on_btnCalcularPressaoHidroestatica_clicked()
272 {

```

```

271     // pega a profundidade digitada pelo usuario
272     QString profundidadeStr = ui->editProfundidadePressaoHidroestatica->text();
273     double profundidade = profundidadeStr.toDouble();
274
275     // calcula a pressao usando o objeto poco e mostra o valor no label
276     ui->lbnPressaoHidroestatica->setText(
277         QString::number(poco->PressaoHidroestaticaNoPonto(profundidade))
278     );
279 }
280
281 void CSimuladorReologico::on_btnCalcularModeloNewtonianoPoco_clicked()
282 {
283     // cria o modelo passando o objeto poco como base
284     modeloNewtoniano = std::make_shared<CModeloNewtoniano>(poco.get());
285
286     // atualiza os campos da interface com os resultados do modelo
287     ui->lbnVelocidadePocoNewtoniano->setText(QString::number(modeloNewtoniano->VMediaPoco
288         ()));
289     ui->lbnReynoldsPocoNewtoniano->setText(QString::number(modeloNewtoniano->ReynoldsPoco
290         ()));
291     ui->lbnTipoFluxoPocoNewtoniano->setText(QString::fromStdString(modeloNewtoniano->
292         FluxoPoco()));
293     ui->lbnPerdaFriccionalPocoNewtoniano->setText(QString::number(modeloNewtoniano->
294         CalcularPerdaPorFriccaoPoco(), 'f', 6));
295 }
296
297 void CSimuladorReologico::on_btnCalcularModeloNewtonianoAnular_clicked()
298 {
299     modeloNewtoniano = std::make_shared<CModeloNewtoniano>(poco.get());
300
301     ui->lbnVelocidadeAnularNewtoniano->setText(QString::number(modeloNewtoniano->
302         VMediaAnular()));
303     ui->lbnReynoldsAnularNewtoniano->setText(QString::number(modeloNewtoniano->
304         ReynoldsAnular()));
305     ui->lbnTipoFluxoAnularNewtoniano->setText(QString::fromStdString(modeloNewtoniano->
306         FluxoAnular()));
307     ui->lbnPerdaFriccionalAnularNewtoniano->setText(QString::number(modeloNewtoniano->
308         CalcularPerdaPorFriccaoAnular(), 'f', 6));
309 }
310
311 void CSimuladorReologico::on_btnCalcularModeloBinghamPoco_clicked()
312 {
313     // verifica se os campos estao vazios e avisa o usuario se faltar algum dado
314     if (ui->editPontoEscoamentoPoco->text().isEmpty() && ui->editViscosidadePlasticaPoco
315         ->text().isEmpty()) {
316         QMessageBox::warning(nullptr, "Aviso", "Preencha o Ponto de Escoamento e a
317             Viscosidade Plastica.");
318     } else if (ui->editPontoEscoamentoPoco->text().isEmpty()) {
319         QMessageBox::warning(nullptr, "Aviso", "Preencha o Ponto de Escoamento.");
320     } else if (ui->editViscosidadePlasticaPoco->text().isEmpty()) {
321         QMessageBox::warning(nullptr, "Aviso", "Preencha a Viscosidade Plastica.");
322     } else {
323         // se os dois valores estiverem preenchidos, converte para double
324         double pontoEscoamento = ui->editPontoEscoamentoPoco->text().toDouble();
325         double viscosidadePlastica = ui->editViscosidadePlasticaPoco->text().toDouble();
326
327         // cria o modelo reológico de Bingham com os dados do poco e os parametros
328         fornecidos
329     }
330 }

```

```

320     modeloBingham = std::make_shared<CModeloBingham>(poco.get(), viscosidadePlastica,
321     pontoEscoamento);
322
323     // atualiza a interface com os resultados dos calculos no espaco do tubo
324     ui->lbnVelocidadePocoBigham->setText(QString::number(modeloBingham->VMediaPoco()))
325     );
326     ui->lbnReynoldsPocoBigham->setText(QString::number(modeloBingham->ReynoldsPoco()))
327     );
328     ui->lbnReynoldsHedstromPocoBigham->setText(QString::number(modeloBingham->
329     ReynoldsHedstromPoco()));
330     ui->lbnReynoldsCriticoPocoBigham->setText(QString::number(modeloBingham->
331     ReynoldsCriticoPoco()));
332     ui->lbnTipoFluxoPocoBigham->setText(QString::fromStdString(modeloBingham->
333     FluxoPoco()));
334     ui->lbnPerdaFriccionalPocoBigham->setText(QString::number(modeloBingham->
335     CalcularPerdaPorFriccaoPoco(), 'f', 6));
336 }
337 }
338
339 void CSimuladorReologico::on_btnCalcularModeloBighamAnular_clicked()
340 {
341     // checa se os campos de entrada estao vazios e avisa o usuario
342     if (ui->editPontoEscoamentoAnular->text().isEmpty() && ui->
343         editViscosidadePlasticaAnular->text().isEmpty()) {
344         QMessageBox::warning(nullptr, "Aviso", "Preencha o Ponto de Escoamento e a
345         Viscosidade Plastica.");
346     } else if (ui->editPontoEscoamentoAnular->text().isEmpty()) {
347         QMessageBox::warning(nullptr, "Aviso", "Preencha o Ponto de Escoamento.");
348     } else if (ui->editViscosidadePlasticaAnular->text().isEmpty()) {
349         QMessageBox::warning(nullptr, "Aviso", "Preencha a Viscosidade Plastica.");
350     } else {
351         // converte os valores digitados para numeros reais
352         double pontoEscoamento = ui->editPontoEscoamentoAnular->text().toDouble();
353         double viscosidadePlastica = ui->editViscosidadePlasticaAnular->text().toDouble()
354         ;
355
356         // cria o modelo de Bingham para calcular no anular (entre tubo e revestimento)
357         modeloBingham = std::make_shared<CModeloBingham>(poco.get(), viscosidadePlastica,
358             pontoEscoamento);
359
360         // preenche os campos com os valores calculados para o anular
361         ui->lbnVelocidadeAnularBigham->setText(QString::number(modeloBingham->
362             VMediaAnular()));
363         ui->lbnReynoldsAnularBigham->setText(QString::number(modeloBingham->
364             ReynoldsAnular()));
365         ui->lbnReynoldsHedstromAnularBigham->setText(QString::number(modeloBingham->
366             ReynoldsHedstromAnular()));
367         ui->lbnReynoldsCriticoAnularBigham->setText(QString::number(modeloBingham->
368             ReynoldsCriticoAnular()));
369         ui->lbnTipoFluxoAnularBigham->setText(QString::fromStdString(modeloBingham->
370             FluxoAnular()));
371         ui->lbnPerdaFriccionalAnularBigham->setText(QString::number(modeloBingham->
372             CalcularPerdaPorFriccaoAnular(), 'f', 6));
373     }
374 }
375
376 void CSimuladorReologico::on_btnCalcularModeloPotenciaPoco_clicked()
377 {
378     if (ui->editIndiceConsistenciaPotenciaPoco->text().isEmpty() && ui->

```

```

    editIndiceComportamentoPoco->text().isEmpty()) {
363     QMessageBox::warning(nullptr, "Aviso", "Preencha o Indice de Consistencia e o de
            Comportamento.");
364 } else if (ui->editIndiceConsistenciaPotenciaPoco->text().isEmpty()) {
365     QMessageBox::warning(nullptr, "Aviso", "Preencha o Indice de Consistencia.");
366 } else if (ui->editIndiceComportamentoPoco->text().isEmpty()) {
367     QMessageBox::warning(nullptr, "Aviso", "Preencha o Indice de Comportamento");
368 } else {
369     // converte os valores digitados para numeros reais
370     double indiceConsistencia = ui->editIndiceConsistenciaPotenciaPoco->text().
            toDouble();
371     double indiceComportamento = ui->editIndiceComportamentoPoco->text().toDouble();
372
373     // cria o modelo da lei da potencia com o indice fornecido
374     modeloPotencia = std::make_shared<CModeloPotencia>(poco.get(), indiceConsistencia
            , indiceComportamento);
375
376     // atualiza os campos da interface com os valores calculados no espaco do tubo
377     ui->lbnVelocidadePocoPotencia->setText(QString::number(modeloPotencia->VMediaPoco
           ()));
378     ui->lbnReynoldsPocoPotencia->setText(QString::number(modeloPotencia->ReynoldsPoco
           ()));
379     ui->lbnReynoldsCriticoPocoPotencia->setText(QString::number(modeloPotencia->
            ReynoldsCriticoPoco()));
380     ui->lbnTipoFluxoPocoPotencia->setText(QString::fromStdString(modeloPotencia->
            FluxoPoco()));
381     ui->lbnPPerdaFriccionalPocoPotencia->setText(QString::number(modeloPotencia->
            CalcularPerdaPorFriccaoPoco(), 'f', 6));
382 }
383 }
384
385 void CSimuladorReologico::on_btnCalcularModeloPotenciaAnular_clicked()
386 {
387     if (ui->editIndiceConsistenciaPotenciaAnular->text().isEmpty() && ui->
            editIndiceComportamentoPoco->text().isEmpty()) {
388         QMessageBox::warning(nullptr, "Aviso", "Preencha o Indice de Consistencia e o de
            Comportamento.");
389     } else if (ui->editIndiceConsistenciaPotenciaAnular->text().isEmpty()) {
390         QMessageBox::warning(nullptr, "Aviso", "Preencha o Indice de Consistencia.");
391     } else if (ui->editIndiceComportamentoAnular->text().isEmpty()) {
392         QMessageBox::warning(nullptr, "Aviso", "Preencha o Indice de Comportamento");
393     } else {
394         // converte os valores digitados para numeros reais
395         double indiceConsistencia = ui->editIndiceConsistenciaPotenciaAnular->text().
            toDouble();
396         double indiceComportamento = ui->editIndiceComportamentoAnular->text().toDouble()
            ;
397
398         // cria o modelo da lei da potencia com o indice fornecido
399         modeloPotencia = std::make_shared<CModeloPotencia>(poco.get(), indiceConsistencia
            , indiceComportamento);
400
401         // atualiza os campos com os resultados dos calculos no anular
402         ui->lbnVelocidadeAnularPotencia->setText(QString::number(modeloPotencia->
            VMediaAnular()));
403         ui->lbnReynoldsAnularPotencia->setText(QString::number(modeloPotencia->
            ReynoldsAnular()));
404         ui->lbnReynoldsCriticoAnularPotencia->setText(QString::number(modeloPotencia->
            ReynoldsCriticoAnular()));
405         ui->lbnTipoFluxoAnularPotencia->setText(QString::fromStdString(modeloPotencia->

```

```

        FluxoAnular()));
406     ui->lbnPerdaFriccionalAnularPotencia->setText(QString::number(modeloPotencia->
407         CalcularPerdaPorFriccaoAnular(), 'f', 6));
408 }
409
410 void CSimuladorReologico::makePlotPoco()
411 {
412     // limpa o grafico anterior (caso ja tenha algo desenhado)
413     ui->customPlotPoco->clearItems();
414
415     // configura os nomes dos eixos
416     ui->customPlotPoco->xAxis->setLabel("Diametro do Poco (m)");
417     ui->customPlotPoco->yAxis->setLabel("Profundidade (m)");
418
419     // define uma faixa inicial de visualizacao nos eixos
420     ui->customPlotPoco->xAxis->setRange(-10, 10); // largura do poco no grafico
421     ui->customPlotPoco->yAxis->setRange(0, 300); // profundidade inicial do grafico
422     ui->customPlotPoco->yAxis->setRangeReversed(true); // deixa o zero em cima e
423         profundidade pra baixo
424
425     // verifica se o objeto poco existe e se ja tem trechos adicionados
426     if (!poco || poco->Trechos().empty()) {
427         return; // nao desenha nada se o poco nao estiver pronto
428     }
429
430     // pega a maior profundidade entre todos os trechos pra ajustar o eixo Y
431     double profundidadeMaxima = 0.0;
432     for (const auto& trecho : poco->Trechos()) {
433         if (trecho->ProfundidadeFinal() > profundidadeMaxima) {
434             profundidadeMaxima = trecho->ProfundidadeFinal();
435         }
436     }
437     ui->customPlotPoco->yAxis->setRange(0, profundidadeMaxima);
438
439     // cria um mapa que associa um nome de fluido a uma cor
440     QMap<QString, QColor> mapaCores;
441
442     // define algumas cores diferentes pra usar nos fluidos
443     QVector<QColor> coresDisponiveis = {
444         QColor(255, 0, 0, 150), // vermelho claro
445         QColor(0, 255, 0, 150), // verde claro
446         QColor(0, 0, 255, 150), // azul claro
447         QColor(255, 165, 0, 150), // laranja
448         QColor(128, 0, 128, 150), // roxo
449         QColor(0, 255, 255, 150) // ciano
450     };
451     int corIndex = 0;
452
453     // para cada trecho do poco, desenha um retangulo com o fluido correspondente
454     for (const auto& trecho : poco->Trechos()) {
455         double profundIncial = trecho->ProfundidadeInicial();
456         double profundFinal = trecho->ProfundidadeFinal();
457         double diametroPoco = poco->DiametroPoco(); // valor fixo usado aqui como
458             referencia visual
459         double diametroSecao = poco->DiametroRevestimentoOD(); // o diametro da tubulacao
460             eh menor que o do buraco
461         QString nomeFluido = QString::fromStdString(trecho->Fluido()->Nome());
462
463         // se for a primeira vez que esse fluido aparece, define uma cor nova pra ele

```

```

461     if (!mapaCores.contains(nomeFluido)) {
462         mapaCores[nomeFluido] = coresDisponiveis[corIndex % coresDisponiveis.size()];
463         corIndex++;
464     }
465     QColor corFluido = mapaCores[nomeFluido];
466
467     // desenha o buraco do poco (cinza claro)
468     QCPIItemRect *retanguloPoco = new QCPIItemRect(ui->customPlotPoco);
469     retanguloPoco->topLeft->setCoords(-diametroPoco / 2, profundInicial);
470     retanguloPoco->bottomRight->setCoords(diametroPoco / 2, profundFinal);
471     retanguloPoco->setPen(QPen(Qt::black));
472     retanguloPoco->setBrush(QBrush(QColor(150, 150, 150, 100))); // cinza
473     transparente
474
475     // desenha a tubulacao com o fluido dentro (com a cor do fluido)
476     QCPIItemRect *retanguloSecao = new QCPIItemRect(ui->customPlotPoco);
477     retanguloSecao->topLeft->setCoords(-diametroSecao / 2, profundInicial);
478     retanguloSecao->bottomRight->setCoords(diametroSecao / 2, profundFinal);
479     retanguloSecao->setPen(QPen(Qt::black));
480     retanguloSecao->setBrush(QBrush(corFluido));
481
482     // coloca o nome do fluido centralizado na altura da secao
483     QCPIItemText *rotulo = new QCPIItemText(ui->customPlotPoco);
484     rotulo->position->setCoords(0, (profundInicial + profundFinal) / 2); // centro
485     vertical
486     rotulo->setText(nomeFluido);
487     rotulo->setFont(QFont("Arial", 10, QFont::Bold));
488     rotulo->setColor(Qt::black);
489     rotulo->setPositionAlignment(Qt::AlignCenter);
490
491     // redesenha o grafico com os novos elementos
492     ui->customPlotPoco->replot();
493 }
494
495 void CSimuladorReologico::on_actionExportar_como_Imagen_triggered()
496 {
497     QString fileName = QFileDialog::getSaveFileName(this, "Salvar_imagem", "", "PNG (*.png);JPEG (*.jpg)");
498
499     if (!fileName.isEmpty()) {
500         QPixmap pixmap = this->grab();
501         pixmap.save(fileName);
502     }
503 }
504
505
506 void CSimuladorReologico::on_actionSobre_o_Programa_triggered()
507 {
508     CJanelaSobreSoftware janelaSobre;
509     janelaSobre.setWindowTitle("Sobre_o_Software");
510     janelaSobre.exec();
511 }
512
513
514 void CSimuladorReologico::SalvarArquivo(bool salvarComo)
515 {
516     QString caminho;
517

```

```

518 // Se for salvarComo ou ainda n o tiver caminho, abrir o di logo
519 if (salvarComo || CaminhoArquivo().isEmpty()) {
520     caminho = QFileDialog::getSaveFileName(this, "Salvar Arquivo", "", "Arquivo DAT (*.dat)");
521     if (caminho.isEmpty()) return; // usuário cancelou
522     CaminhoArquivo(caminho);
523     NomeArquivo(QFileInfo(caminho).fileName());
524 } else {
525     caminho = CaminhoArquivo(); // salva direto
526 }
527
528 QFile arquivo(caminho);
529 if (!arquivo.open(QIODevice::WriteOnly | QIODevice::Text)) {
530     QMessageBox::warning(this, "Erro", "Nao foi possivel salvar o arquivo.");
531     return;
532 }
533
534 QTextStream out(&arquivo);
535
536 out << "# Configuracao do Poco
537 -----
538     n";
539 out << "#"
540     << QString("Nome").leftJustified(15, ' ');
541     << QString("Profund (ft)").leftJustified(15, ' ');
542     << QString("Pressao (psi)").leftJustified(15, ' ');
543     << QString("Diametro (in)").leftJustified(15, ' ');
544     << QString("OD (in)").leftJustified(15, ' ');
545     << QString("ID (in)").leftJustified(15, ' ');
546     << QString("Vazao (bb/d)").leftJustified(15, ' ');
547     << "\n";
548
549 // dados tamb m com 15 caracteres por campo
550 out << " ";
551 out << ui->editNomePoco->text().leftJustified(15, ' ');
552 out << ui->editProfundidadeTotal->text().leftJustified(15, ' ');
553 out << ui->editPressaoSuperficie->text().leftJustified(15, ' ');
554 out << ui->editDiametroPoco->text().leftJustified(15, ' ');
555 out << ui->editDiametroOD->text().leftJustified(15, ' ');
556 out << ui->editDiametroID->text().leftJustified(15, ' ');
557 out << ui->editVazao->text().leftJustified(15, ' ');
558 out << "\n";
559
560 // Escreve os dados dos fluidos
561 out << "\n\n#\nConfiguracao dos Fluidos
562 -----
563     n";
564 out << "#"
565     << QString("Nome").leftJustified(15, ' ');
566     << QString("Densidade (lbm/gal)").leftJustified(23, ' ');
567     << QString("Viscosidade (cP)").leftJustified(23, ' ');
568     << QString("Prof. Inicial (ft)").leftJustified(23, ' ');
569     << QString("Prof. Final (ft)").leftJustified(20, ' ');
570     << "\n";
571
572 int linhas = ui->tblFluidos->rowCount();
573 for (int i = 0; i < linhas; ++i) {
574     out << " "
575         << ui->tblFluidos->item(i, 0)->text().leftJustified(15, ' ')

```

```

573     << ui->tblFluidos->item(i, 1)->text().leftJustified(23, 'L')
574     << ui->tblFluidos->item(i, 2)->text().leftJustified(23, 'L')
575     << ui->tblFluidos->item(i, 3)->text().leftJustified(23, 'L')
576     << ui->tblFluidos->item(i, 4)->text().leftJustified(20, 'L')
577     << "\n";
578 }
579
580 arquivo.close();
581 QMessageBox::information(this, "Salvo", "Arquivo salvo com sucesso!");
582 }
583
584 void CSimuladorReologico::on_actionArquivo_dat_triggered()
585 {
586     // Abre uma janelinha pro usuario escolher o arquivo .dat
587     QString caminhoDoArquivo = QFileDialog::getOpenFileName(
588         this,
589         "Selecione um arquivo",
590         "",
591         "Todos os arquivos (*.*)"
592     );
593
594     // Se o usuario nao escolheu nada, sai da funcao
595     if (caminhoDoArquivo.isEmpty())
596         return;
597
598     // Converte o caminho do Qt para string do C++
599     std::string caminhoDoArquivoStr = caminhoDoArquivo.toStdString();
600
601     // Tenta abrir o arquivo
602     std::ifstream file(caminhoDoArquivoStr);
603     if (!file.is_open()) {
604         QMessageBox::warning(this, "Erro", "Nao foi possivel abrir o arquivo!");
605         return;
606     }
607
608     std::string linha;
609     bool lendoFluidos = false;
610     bool encontrouCabecalho = false;
611     bool leuPoco = false;
612     bool leuFluido = false;
613
614     // Le o arquivo linha por linha
615     while (std::getline(file, linha)) {
616         // Ignora linhas vazias ou comentarios
617         if (linha.empty() || linha[0] == '#') {
618             if (linha.find("Fluidos") != std::string::npos) {
619                 lendoFluidos = true;
620                 encontrouCabecalho = true;
621             }
622             continue;
623         }
624
625         if (!lendoFluidos) {
626             // Tentativa de ler os dados do po
627             std::istringstream iss(linha);
628             std::string nome;
629             double profundidade, pressaoSuperficie, diametro, OD, ID, vazao;
630
631             if (iss >> nome >> profundidade >> pressaoSuperficie >> diametro >> OD >> ID
632                 >> vazao) {

```

```

632         poco = std::make_unique<CObjetoPoco>(
633             CObjetoPoco::CriarParaModulo01(nome, profundidade, pressaoSuperficie,
634                                         diametro, OD, ID, vazao)
635         );
636         leuPoco = true;
637
638         // Ativa botoes da interface
639         ui->btnCalcularPressaoHidroestatica->setEnabled(true);
640         ui->btnCalcularModeloNewtonianoPoco->setEnabled(true);
641         ui->btnCalcularModeloNewtonianoAnular->setEnabled(true);
642         ui->btnCalcularModeloBighamPoco->setEnabled(true);
643         ui->btnCalcularModeloBighamAnular->setEnabled(true);
644         ui->btnCalcularModeloPotenciaPoco->setEnabled(true);
645         ui->btnCalcularModeloPotenciaAnular->setEnabled(true);
646         ui->btnExibirGraficoPressaoHidroestatica->setEnabled(true);
647     }
648
649 } else {
650     // Leitura dos fluidos associados ao po o
651     std::istringstream iss(linha);
652     std::string nome;
653     double densidade, viscosidade, profInicial, profFinal;
654
655     if (iss >> nome >> densidade >> viscosidade >> profInicial >> profFinal) {
656         auto fluido = std::make_unique<CFluido>(nome, densidade, viscosidade);
657         auto trechoPoco = std::make_unique<CTrechoPoco>(profInicial, profFinal,
658                                                       std::move(fluido));
659         poco->AdicionarTrechoPoco(std::move(trechoPoco));
660         leuFluido = true;
661     }
662 }
663 file.close(); // fecha o arquivo apos a leitura
664
665 // Se nao conseguiu ler nem o po o nem fluidos, mostra um alerta
666 if (!leuPoco || !leuFluido || !encontrouCabecalho) {
667     QMessageBox::warning(this, "Arquivo Inorreto",
668                           "Aten o :o:arquivo: selecionado: n o est uno:formato:
669 esperado.\n"
670                           "Por favor, abra um arquivo de configura o valido.");
671     return;
672 }
673
674 // Atualiza a interface com os dados lidos
675 AtualizarDados();
676 ui->statusbar->showMessage("Dados importados com sucesso!");
677
678 NomeArquivo(QFileInfo(caminhoDoArquivo).fileName());
679 CaminhoArquivo(caminhoDoArquivo);
680
681
682
683 void CSimuladorReologico::on_btnExibirGraficoPressaoHidroestatica_clicked()
684 {
685     CJanelaGraficoPressaoHidroestatica grafico(poco->PlotarProfundidadePorPressaoMedia())
686     ;
687     grafico.exec();
688     ui->statusbar->showMessage("Dados Plotador com Sucesso!");

```

```

688 }
689
690
691 void CSimuladorReologico::on_actionSalvar_triggered()
692 {
693     SalvarArquivo(false); // salvar direto
694 }
695
696 void CSimuladorReologico::on_actionSalvar_Como_triggered()
697 {
698     SalvarArquivo(true); // for ar abrir QFileDialog
699 }
700
701 void CSimuladorReologico::on_actionNova_Simula_o_triggered()
702 {
703     QMessageBox::StandardButton resposta = QMessageBox::question(
704         this,
705         "",
706         "Tem certeza que deseja iniciar uma nova simula o?",
707         QMessageBox::Yes | QMessageBox::No
708     );
709
710     if (resposta == QMessageBox::Yes) {
711         CSimuladorReologico *newWindow = new CSimuladorReologico();
712         newWindow->show();
713         this->close();
714     }
715 }
716
717
718
719 void CSimuladorReologico::on_actionAjuda_triggered()
720 {
721     QString caminho = QCoreApplication::applicationDirPath() + "/PDFs/ManualUsuario.pdf";
722     QDesktopServices::openUrl(QUrl::fromLocalFile(caminho));
723 }
724
725
726 void CSimuladorReologico::on_actionSobre_os_Modelos_Reol_gicos_triggered()
727 {
728     QString caminho = QCoreApplication::applicationDirPath() + "/PDFs/FormulasUtilizadas.pdf";
729     QDesktopServices::openUrl(QUrl::fromLocalFile(caminho));
730 }
731 }

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 4 o arquivo de cabeçalho da classe CSimuladorPerdaTubulacao.frame

Listing 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
17 // variacao)
18 // aqui que o usuario interage com os dados do po o , dos trechos e calcula L , perda,
19 // efeito balao etc
20 class CSimuladorPerdaTubulacao : public QMainWindow
21 {
22     Q_OBJECT
23
24 public:
25     // construtor e destrutor
26     explicit CSimuladorPerdaTubulacao(QWidget *parent = nullptr);
27     ~CSimuladorPerdaTubulacao();
28     QString nomeArquivo;
29     QString caminhoArquivo;
30
31 private slots:
32     void on_btnAdicionarPropriedades_clicked();           // adiciona as propriedades termicas
33                                         // e mecanicas do fluido
34     void AtualizarDados();                                // atualiza os dados na tela com base no objeto do
35                                         // po o
36     void on_btnAdicionarTrecho_clicked();                 // adiciona um novo trecho de
37                                         // tubulacao ao po o
38     void makePlotTemperatura(double TempInicial, double TempFinal, double profundidade,
39                               QCustomPlot* plot); // gera grafico de temperatura com profundidade
40     void on_btnRemoverTrecho_clicked();                   // remove um trecho da tubulacao
41     void makePlotPoco();                                 // desenha o perfil visual do po o
42     void on_btnCalcularVariacoes_clicked();              // calcula L , efeito balao, forca
43                                         // etc
44
45     void on_actionArquivo_Dat_triggered();                // importa dados do arquivo .dat
46     void EditarDadosPoco();                             // edita os dados gerais do po o (
47                                         // nome , pressao etc)
48
49     // edita uma linha da tabela de fluidos ou trechos do poco
50     void EditarLinhaTabela(int row);
51
52     // opcoes do menu da interface
53     void on_actionNova_Simula_o_triggered();
54     void on_actionExportar_Como_Imagen_triggered();
55     void on_actionSobre_o_SEEP_triggered();
56     void SalvarArquivo(bool salvarComo);
57     void on_actionSalvar_como_triggered();
58     void on_actionSalvar_triggered();
59
60     // getters
61     QString NomeArquivo() { return nomeArquivo; }
62     QString CaminhoArquivo() { return caminhoArquivo; }
63
64     // setters

```

```
59 void NomeArquivo(QString nome) { nomeArquivo = nome; }
60 void CaminhoArquivo(QString caminho) { caminhoArquivo = caminho; }
61
62
63 void on_actionManual_do_Usuario_triggered();
64
65 void on_actionFormulas_Utilizadas_triggered();
66
67 private:
68 Ui::CSimuladorPerdaTubulacao *ui; // ponteiro pra interface gerada pelo Qt Designer
69
70 // ponteiros para os objetos principais que compoem o modelo do po o
71 std::shared_ptr<COBJETOPOCO> poco = nullptr; // representa o po o como
    um todo
72 std::shared_ptr<CTrechoPoco> trechoPoco = nullptr; // trecho individual de
    tubulacao
73 std::shared_ptr<CFLUIDO> fluido = nullptr; // fluido associado aos
    trechos
74
75 // modelos reologicos usados pra calcular propriedades de escoamento
76 std::shared_ptr<CModeloNewtoniano> modeloNewtoniano = nullptr;
77 std::shared_ptr<CModeloBingham> modeloBingham = nullptr;
78 std::shared_ptr<CModeloPotencia> modeloPotencia = nullptr;
79 };
80
81 #endif // CSIMULADORPERDATUBULACAO_H
```

Fonte: produzido pelo autor.

Apresenta-se na listagem 5 a implementação da classe CSimuladorPerdaTubulacao.frame

Listing 5: Arquivo de implementação da classe CModeloReologico

```

25     ui->tblTrechos->setEditTriggers(QAbstractItemView::DoubleClicked | QAbstractItemView
26             ::EditKeyPressed);
27
28     // Sinal para alterar os valores das caixas
29     QObject::connect(ui->editNomePoco, &QLineEdit::editingFinished, this, &
30             CSimuladorPerdaTubulacao::EditarDadosPoco);
31     QObject::connect(ui->editProfundidadeTotal, &QLineEdit::editingFinished, this, &
32             CSimuladorPerdaTubulacao::EditarDadosPoco);
33     QObject::connect(ui->editPressaoSupInicial, &QLineEdit::editingFinished, this, &
34             CSimuladorPerdaTubulacao::EditarDadosPoco);
35     QObject::connect(ui->editPressaoSupFinal, &QLineEdit::editingFinished, this, &
36             CSimuladorPerdaTubulacao::EditarDadosPoco);
37     QObject::connect(ui->editTemperaturaSuperiorInicial, &QLineEdit::editingFinished,
38             this, &CSimuladorPerdaTubulacao::EditarDadosPoco);
39     QObject::connect(ui->editTemperaturaFundoInicial, &QLineEdit::editingFinished, this,
40             &CSimuladorPerdaTubulacao::EditarDadosPoco);
41     QObject::connect(ui->editTemperaturaSuperiorFinal, &QLineEdit::editingFinished, this,
42             &CSimuladorPerdaTubulacao::EditarDadosPoco);
43     QObject::connect(ui->editProfundidadeMedicao, &QLineEdit::editingFinished, this, &
44             CSimuladorPerdaTubulacao::EditarDadosPoco);
45     QObject::connect(ui->checkBoxPacker, &QCheckBox::stateChanged, this, &
46             CSimuladorPerdaTubulacao::EditarDadosPoco);
47
48     // iniciar com botões desativado
49     ui->btnAdicionarTrecho->setEnabled(false);
50     ui->btnRemoverTrecho->setEnabled(false);
51     ui->btnCalcularVariacoes->setEnabled(false);
52
53     QObject::connect(ui->tblFluidos, &QTableWidget::cellChanged, this, &
54             CSimuladorPerdaTubulacao::EditarLinhaTabela);
55     QObject::connect(ui->tblTrechos, &QTableWidget::cellChanged, this, &
56             CSimuladorPerdaTubulacao::EditarLinhaTabela);
57
58     //abrir janela no meio do monitor
59     QScreen *screen = QGuiApplication::primaryScreen();
60     QRect screenGeometry = screen->geometry();
61
62     int x = (screenGeometry.width() - this->width()) / 2;
63     int y = (screenGeometry.height() - this->height()) / 2;
64
65     this->move(x, y);
66
67     makePlotPoco();
68 }
69
70 CSimuladorPerdaTubulacao::~CSimuladorPerdaTubulacao()
71 {
72     delete ui;
73 }
74
75 void CSimuladorPerdaTubulacao::EditarDadosPoco() {
76     QString nome = ui->editNomePoco->text();
77     bool ok1, ok2, ok3, ok4, ok5, ok6, ok7, ok8;
78     double profund = ui->editProfundidadeTotal->text().toDouble(&ok1);
79     double diamPoco = ui->editDiametroPoco->text().toDouble(&ok2);
80     double pressao = ui->editPressaoSupInicial->text().toDouble(&ok3);

```

```

72     double pressaoFim = ui->editPressaoSupFinal->text().toDouble(&ok4);
73     double temperaturaSuperiorInicial = ui->editTemperaturaSuperiorInicial->text().
74         toDouble(&ok5);
75     double temperaturaFundoInicial = ui->editTemperaturaFundoInicial->text().toDouble(&
76         ok6);
77     double temperaturaSuperiorFinal = ui->editTemperaturaSuperiorFinal->text().toDouble(&
78         ok7);
79     double temperaturaFundoFinal = ui->editTemperaturaFundoFinal->text().toDouble(&ok8);
80     bool haPacker = ui->checkBoxPacker->isChecked();
81
82
83     if (!nome.isEmpty() && ok1 && ok2 && ok3 && ok4 && ok5 && ok6 && ok7 && ok8) {
84         if (!poco) {
85             // Cria o p o o
86
87             poco = std::make_unique<CObjetoPoco>(
88                 CObjetoPoco::CriarParaModulo02(nome.toStdString(), profund, diamPoco,
89                     pressao, pressaoFim, temperaturaSuperiorInicial,
90                     temperaturaFundoInicial, temperaturaSuperiorFinal,
91                     temperaturaFundoFinal, haPacker)
92             );
93
94             ui->btnAdicionarTrecho->setEnabled(true);
95             ui->btnRemoverTrecho->setEnabled(true);
96             ui->btnCalcularVariacoes->setEnabled(true);
97
98             ui->statusbar->showMessage(" Po o criado com Sucesso!");
99         } else {
100             // Atualiza dados do p o o j existente
101             poco->NomePoco(nome.toStdString());
102             poco->ProfundidadeTotal(profund);
103             poco->PressaoSuperficie(pressao);
104             poco->PressaoSuperficieFim(pressaoFim);
105             poco->TemperaturaTopoInicial(temperaturaSuperiorInicial);
106             poco->TemperaturaFundoInicial(temperaturaFundoInicial);
107             poco->TemperaturaTopoFinal(temperaturaSuperiorFinal);
108             poco->TemperaturaFundoFinal(temperaturaFundoFinal);
109             poco->Packer(haPacker);
110             ui->statusbar->showMessage(" Dados de Po o Atualizado com Sucesso!");
111         }
112
113         AtualizarDados(); // Atualiza os dados calculados e a interface
114     }
115
116     void CSimuladorPerdaTubulacao::on_btnAdicionarPropriedades_clicked()
117 {
118     std::string nome;
119     double profundidade, diamPoco, pressaoSup, pressaoSupFim, temperaturaSuperiorInicial,
120         temperaturaFundoInicial, temperaturaSuperiorFinal, temperaturaFundoFinal;
121     bool haPacker;
122
123     QString text;
124
125     text = ui->editNomePoco->text();
126     nome = text.toStdString();
127     text = ui->editPressaoSupInicial->text();
128     pressaoSup = text.toDouble();
129     text = ui->editPressaoSupFinal->text();
130     pressaoSupFim = text.toDouble();
131     text = ui->editProfundidadeTotal->text();

```

```

125 profundidade = text.toDouble();
126 text = ui->editDiametroPoco->text();
127 diamPoco = text.toDouble();
128 text = ui->editTemperaturaSuperiorInicial->text();
129 temperaturaSuperiorInicial = text.toDouble();
130 text = ui->editTemperaturaFundoInicial->text();
131 temperaturaFundoInicial = text.toDouble();
132 text = ui->editTemperaturaSuperiorFinal->text();
133 temperaturaSuperiorFinal = text.toDouble();
134 text = ui->editTemperaturaFundoFinal->text();
135 temperaturaFundoFinal = text.toDouble();
136 haPacker = ui->checkBoxPacker->isChecked();
137
138
139 if (poco) {
140     QMessageBox::StandardButton resposta = QMessageBox::question(
141         this,
142         "",
143         "Ao confirmar, todos os fluidos ser\u00e3o deletados! Tem certeza?",
144         QMessageBox::Yes | QMessageBox::No
145     );
146
147     if (resposta == QMessageBox::Yes) {
148         poco = std::make_unique<CObjetoPoco>(
149             CObjetoPoco::CriarParaModulo02(nome, profundidade, diamPoco, pressaoSup,
150                 pressaoSupFim, temperaturaSuperiorInicial, temperaturaFundoInicial,
151                 temperaturaSuperiorFinal, temperaturaFundoFinal, haPacker)
152             );
153         AtualizarDados();
154     } else {
155         poco = std::make_unique<CObjetoPoco>(
156             CObjetoPoco::CriarParaModulo02(nome, profundidade, diamPoco, pressaoSup,
157                 pressaoSupFim, temperaturaSuperiorInicial, temperaturaFundoInicial,
158                 temperaturaSuperiorFinal, temperaturaFundoFinal, haPacker)
159             );
160     }
161     makePlotTemperatura(temperaturaSuperiorInicial, temperaturaFundoInicial, profundidade
162         , ui->customPlotTemperaturaInicial);
163     makePlotTemperatura(temperaturaSuperiorFinal, temperaturaFundoFinal, profundidade, ui
164         ->customPlotTemperaturaFinal);
165     makePlotPoco();
166 }
167 void CSimuladorPerdaTubulacao::AtualizarDados()
168 {
169     ui->tblFluidos->blockSignals(true);
170     ui->tblTreichos->blockSignals(true);
171
172     if (poco){
173         // Atualiza os valores dos QLineEdits com os dados do objeto poco
174         ui->editNomePoco->setText(QString::fromStdString(poco->NomePoco()));           //
175         Profundidade total do po o
176         ui->editProfundidadeTotal->setText(QString::number(poco->ProfundidadeTotal()));
177             // Profundidade total do po o
178         ui->editDiametroPoco->setText(QString::number(poco->DiametroPoco()));
179         ui->editPressaoSupInicial->setText(QString::number(poco->PressaoSuperficie()));
180             // Profundidade ocupada

```

```

176     ui->editPressaoSupFinal->setText(QString::number(poco->PressaoSuperficieFim()));
177     ui->editTemperaturaSuperiorInicial->setText(QString::number(poco->
178         TemperaturaTopoInicial())); // Dimetro do p o o
179     ui->editTemperaturaFundoInicial->setText(QString::number(poco->
180         TemperaturaFundoInicial())); // Dimetro externo do revestimento (OD)
181     ui->editTemperaturaSuperiorFinal->setText(QString::number(poco->
182         TemperaturaTopoFinal())); // Dimetro interno do revestimento (ID)
183     ui->editTemperaturaFundoFinal->setText(QString::number(poco->
184         TemperaturaFundoFinal())); // Vaz o do fluido no
185         p o o
186
187
188
189 // Atualizar QTableWidget com os dados dos trechos
190 ui->tblTrechos->setRowCount(static_cast<int>(poco->Trechos().size()));
191 ui->tblFluidos->setRowCount(static_cast<int>(poco->Trechos().size()));
192 int row = 0;
193 for (const auto& trecho : poco->Trechos()) {
194
195     ui->tblTrechos->setItem(row, 0, new QTableWidgetItem(QString::fromStdString(
196         trecho->Nome())));
197     ui->tblTrechos->setItem(row, 1, new QTableWidgetItem(QString::number(trecho->
198         ProfundidadeInicial(), 'f', 2)));
199     ui->tblTrechos->setItem(row, 2, new QTableWidgetItem(QString::number(trecho->
200         ProfundidadeFinal(), 'f', 2)));
201     ui->tblTrechos->setItem(row, 3, new QTableWidgetItem(QString::number(trecho->
202         DiametroExterno(), 'f', 2)));
203     ui->tblTrechos->setItem(row, 4, new QTableWidgetItem(QString::number(trecho->
204         DiametroInterno(), 'f', 2)));
205     ui->tblTrechos->setItem(row, 5, new QTableWidgetItem(QString::number(trecho->
206         CoeficientePoisson(), 'f', 2)));
207     ui->tblTrechos->setItem(row, 6, new QTableWidgetItem(QString::number(trecho->
208         CoeficienteExpancaoTermica(), 'f', 8)));
209     ui->tblTrechos->setItem(row, 7, new QTableWidgetItem(QString::number(trecho->
210         ModuloElasticidade(), 'f', 2)));
211     ui->tblTrechos->setItem(row, 8, new QTableWidgetItem(QString::number(trecho->
212         PesoUnidade(), 'f', 2)));
213
214     ui->tblFluidos->setItem(row, 0, new QTableWidgetItem(QString::fromStdString(
215         trecho->Fluido()->Nome())));
216     ui->tblFluidos->setItem(row, 1, new QTableWidgetItem(QString::number(trecho->
217         Fluido()->Densidade(), 'f', 2)));
218     ui->tblFluidos->setItem(row, 2, new QTableWidgetItem(QString::number(trecho->
219         Fluido()->Viscosidade(), 'f', 2)));
220
221     ++row;
222 }
223
224 makePlotTemperatura(poco->TemperaturaTopoInicial(), poco->TemperaturaFundoInicial(),
225     poco->ProfundidadeTotal(), ui->customPlotTemperaturaInicial);
226 makePlotTemperatura(poco->TemperaturaTopoFinal(), poco->TemperaturaFundoFinal(), poco
227     ->ProfundidadeTotal(), ui->customPlotTemperaturaFinal);
228 makePlotPoco();
229
230 ui->tblFluidos->blockSignals(false);

```

```

217     ui->tblTrechos->blockSignals(false);
218 }
219
220 void CSimuladorPerdaTubulacao::on_btnAdicionarTrecho_clicked()
221 {
222     ui->tblTrechos->setEditTriggers(QAbstractItemView::NoEditTriggers);
223     ui->tblFluidos->setEditTriggers(QAbstractItemView::NoEditTriggers);
224
225     if (!poco) {
226         QMessageBox::warning(this, "Erro", "As propriedades do ponto precisam estar preenchidas!");
227     }
228
229     else{
230         CJanelaAdicionarTrechoTubulacao JanelaTrecho;
231         JanelaTrecho.exec();
232
233         if (JanelaTrecho.Trecho() != "" &&
234             JanelaTrecho.ProfundidadeInicial() != "" &&
235             JanelaTrecho.ProfundidadeFinal() != "" &&
236             JanelaTrecho.DiametroExterno() != "" &&
237             JanelaTrecho.DiametroInterno() != "" &&
238             JanelaTrecho.CoefficientePoisson() != "" &&
239             JanelaTrecho.CoefficienteExpansaoTermica() != "" &&
240             JanelaTrecho.ModuloElasticidade() != "" &&
241             JanelaTrecho.PesoUnidade() != "" &&
242             JanelaTrecho.NomeFluido() != "" &&
243             JanelaTrecho.Densidade() != "" &&
244             JanelaTrecho.Viscosidade() != ""){
```

245

246 int numLinhas = ui->tblTrechos->rowCount();

247

248 ui->tblTrechos->insertRow(numLinhas);

249 ui->tblTrechos->setItem(numLinhas, 0, new QTableWidgetItem(JanelaTrecho.Trecho()));

250 ui->tblTrechos->setItem(numLinhas, 1, new QTableWidgetItem(JanelaTrecho.ProfundidadeInicial()));

251 ui->tblTrechos->setItem(numLinhas, 2, new QTableWidgetItem(JanelaTrecho.ProfundidadeFinal()));

252 ui->tblTrechos->setItem(numLinhas, 3, new QTableWidgetItem(JanelaTrecho.DiametroExterno()));

253 ui->tblTrechos->setItem(numLinhas, 4, new QTableWidgetItem(JanelaTrecho.DiametroInterno()));

254 ui->tblTrechos->setItem(numLinhas, 5, new QTableWidgetItem(JanelaTrecho.CoefficientePoisson()));

255 ui->tblTrechos->setItem(numLinhas, 6, new QTableWidgetItem(JanelaTrecho.CoefficienteExpansaoTermica()));

256 ui->tblTrechos->setItem(numLinhas, 7, new QTableWidgetItem(JanelaTrecho.ModuloElasticidade()));

257 ui->tblTrechos->setItem(numLinhas, 8, new QTableWidgetItem(JanelaTrecho.PesoUnidade()));

258

259 ui->tblFluidos->insertRow(numLinhas);

260 ui->tblFluidos->setItem(numLinhas, 0, new QTableWidgetItem(JanelaTrecho.NomeFluido()));

261 ui->tblFluidos->setItem(numLinhas, 1, new QTableWidgetItem(JanelaTrecho.Densidade()));

262 ui->tblFluidos->setItem(numLinhas, 2, new QTableWidgetItem(JanelaTrecho.Viscosidade()));

```

264
265     std::string NomeTrecho = JanelaTrecho.Trecho().toStdString();
266     double profundInicial = JanelaTrecho.ProfundidadeInicial().toDouble();
267     double profundFinal = JanelaTrecho.ProfundidadeFinal().toDouble();
268     double diametroExterno = JanelaTrecho.DiametroExterno().toDouble();
269     double diametroInterno = JanelaTrecho.DiametroInterno().toDouble();
270     double coeficientePoisson = JanelaTrecho.CoefficientePoisson().toDouble();
271     double coeficienteExpansaoTermica = JanelaTrecho.CoefficienteExpansaoTermica()
272         .toDouble();
273     double moduloElasticidade = JanelaTrecho.ModuloElasticidade().toDouble();
274     double pesoUnidade = JanelaTrecho.PesoUnidade().toDouble();
275
276     std::string nome = JanelaTrecho.NomeFluido().toStdString();
277     double densidade = JanelaTrecho.Densidade().toDouble();
278     double viscosidade = JanelaTrecho.Viscosidade().toDouble();
279
280     auto fluido = std::make_unique<CFluido>(nome, densidade, viscosidade);
281     auto trechoPoco = std::make_unique<CTrechoPoco>(NomeTrecho, profundInicial,
282         profundFinal, std::move(fluido), diametroExterno, diametroInterno,
283         coeficientePoisson, coeficienteExpansaoTermica, moduloElasticidade,
284         pesoUnidade);
285     poco->AdicionarTrechoPoco(std::move(trechoPoco));
286
287     AtualizarDados();
288 }
289 }
290
291 void CSimuladorPerdaTubulacao::makePlotTemperatura(double TempInicial, double TempFinal,
292     double profundidade, QCustomPlot* plot)
293 {
294     // Limpa graficos anteriores
295     plot->clearItems();
296     plot->clearPlottables();
297
298     // Configura eixos
299     plot->xAxis->setLabel("Temperatura ( F )");
300     plot->yAxis->setLabel("Profundidade (ft)");
301     plot->yAxis->setRangeReversed(true); // profundidade cresce pra baixo
302
303     // Adiciona "respiro" nos extremos
304     double margemProfundidade = profundidade * 0.05; // 5% de margem visual
305
306     // Define pontos
307     double tempMeio = (TempInicial + TempFinal) / 2.0;
308     QVector<double> temperaturas = {TempInicial, tempMeio, TempFinal};
309     QVector<double> profundidades = {0.0 + margemProfundidade, profundidade / 2.0,
310         profundidade - margemProfundidade};
311
312     // Ajuste de range
313     double tempMin = *std::min_element(temperaturas.begin(), temperaturas.end());
314     double tempMax = *std::max_element(temperaturas.begin(), temperaturas.end());
315
316     plot->xAxis->setRange(tempMin - 5, tempMax + 5);
317     plot->yAxis->setRange(0.0, profundidade); // Mantem 0 a profundidade total, o respiro
318         e visual apenas
319
320     // Grid leve
321     plot->xAxis->grid()->setVisible(true);
322     plot->yAxis->grid()->setVisible(true);

```

```

317     plot->xAxis->grid()->setPen(QPen(QColor(220, 220, 220)));
318     plot->yAxis->grid()->setPen(QPen(QColor(220, 220, 220)));
319
320     // Linha do perfil
321     QCPGraph *perfilTemp = plot->addGraph();
322     perfilTemp->setData(temperaturas, profundidades);
323     perfilTemp->setPen(QPen(QColor(200, 0, 0), 2));
324
325     // Marcar e rotular os 3 pontos (topo, meio e fundo)
326     for (int i = 0; i < temperaturas.size(); ++i) {
327         QCPIItemEllipse *ponto = new QCPIItemEllipse(plot);
328         ponto->topLeft->setCoords(temperaturas[i] - 2, profundidades[i] - 20);
329         ponto->bottomRight->setCoords(temperaturas[i] + 2, profundidades[i] + 20);
330         ponto->setPen(Qt::NoPen);
331         ponto->setBrush(QBrush(Qt::darkGray));
332
333         QCPIItemText *rotulo = new QCPIItemText(plot);
334         rotulo->position->setCoords(temperaturas[i] + 4, profundidades[i]);
335         rotulo->setText(QString::number(temperaturas[i], 'f', 1) + " ° F ");
336         rotulo->setFont(QFont("Arial", 8));
337         rotulo->setColor(Qt::darkGray);
338         // Ajusta alinhamento do rotulo do ultimo ponto para dentro do grafico
339         if (i == temperaturas.size() - 1)
340             rotulo->setPositionAlignment(Qt::AlignRight | Qt::AlignVCenter); // alinha
341             para a esquerda do ponto final
342         else
343             rotulo->setPositionAlignment(Qt::AlignLeft | Qt::AlignVCenter);
344     }
345
346     // Limpeza estatica
347     plot->legend->setVisible(false);
348     plot->setBackground(Qt::white);
349     plot->xAxis->setBasePen(QPen(Qt::black));
350     plot->yAxis->setBasePen(QPen(Qt::black));
351     plot->xAxis->setTickPen(QPen(Qt::black));
352     plot->yAxis->setTickPen(QPen(Qt::black));
353     plot->xAxis->setTickLabelColor(Qt::black);
354     plot->yAxis->setTickLabelColor(Qt::black);
355
356     // Replotar
357     plot->replot();
358 }
359
360
361
362 void CSimuladorPerdaTubulacao::on_btnRemoverTrecho_clicked()
363 {
364     // pega qual linha ta selecionada em cada tabela
365     int linhaSelecionadaTrecho = ui->tblTrechos->currentRow();
366     int linhaSelecionadaFluido = ui->tblFluidos->currentRow();
367
368     // se a selecao for feita na tabela de trecho, faz a remocao
369     if (linhaSelecionadaTrecho >= 0) {
370         QMessageBox::StandardButton resposta = QMessageBox::question(
371             this,
372             "Confirmação",
373             "Deseja remover o trecho e o fluido associado?",
374             QMessageBox::Yes | QMessageBox::No
375         );

```

```

376
377     if (resposta == QMessageBox::Yes) {
378         // pega o nome do trecho pela tabela (pra remover do objeto poco)
379         QString nomeTrecho = ui->tblTreichos->item(linhaSelecionadaTrecho, 0)->text();
380
381         // remove a mesma linha das duas tabelas
382         ui->tblTreichos->removeRow(linhaSelecionadaTrecho);
383         ui->tblFluidos->removeRow(linhaSelecionadaTrecho);
384
385         // remove o trecho (e junto o fluido) do objeto poco
386         poco->RemoverTrechoPoco(nomeTrecho.toStdString());
387
388         // atualiza visualmente os dados
389         AtualizarDados();
390         ui->statusbar->showMessage("Trecho removido com sucesso!");
391     }
392
393 } else if (linhaSelecionadaFluido >= 0) {
394     // se clicou so na tabela de fluido, avisa que deve usar a outra
395     QMessageBox::warning(this, "Aviso", "A remoção deve ser feita pela tabela de trechos.");
396 } else {
397     // nenhuma linha foi selecionada
398     QMessageBox::warning(this, "Erro", "Nenhuma linha foi selecionada.");
399 }
400 }

401
402

403 void CSimuladorPerdaTubulacao::makePlotPoco()
404 {
405     ui->customPlotPoco->clearItems();
406     ui->customPlotPoco->xAxis->setLabel("Diâmetro(poli)");
407     ui->customPlotPoco->yAxis->setLabel("Profundidade(poli)");
408     ui->customPlotPoco->yAxis->setRangeReversed(true);

409
410     if (!poco || poco->Trechos().empty())
411         return;
412
413     // 1. Profundidade máxima e maior diâmetro externo
414     double profundidadeMaxima = 0.0;
415     double maiorDiametroExterno = 0.0;
416     for (const auto& trecho : poco->Trechos()) {
417         profundidadeMaxima = std::max(profundidadeMaxima, trecho->ProfundidadeFinal());
418         maiorDiametroExterno = std::max(maiorDiametroExterno, trecho->DiametroExterno());
419     }
420
421     // 2. Buraco = maior diâmetro externo + 3 polegadas
422     double diametroBuraco = maiorDiametroExterno + 3.0;
423
424     // 3. Ajuste visual no eixo X: 1.5 vezes o furo
425     double larguraGrafico = diametroBuraco * 1.5;
426     ui->customPlotPoco->xAxis->setRange(-larguraGrafico / 2.0, larguraGrafico / 2.0);
427     ui->customPlotPoco->yAxis->setRange(0, profundidadeMaxima);

428
429     // 4. Cores dos fluidos
430     QMap<QString, QColor> mapaCores;
431     QVector<QColor> coresDisponiveis = {
432         QColor(70, 130, 180, 180), QColor(255, 0, 0, 150),
433         QColor(0, 255, 0, 150), QColor(0, 0, 255, 150),
434         QColor(255, 165, 0, 150), QColor(128, 0, 128, 150)

```

```

435     };
436     int corIndex = 0;
437
438     // 5. Desenho dos trechos
439     for (const auto& trecho : poco->Trechos()) {
440         double z1 = trecho->ProfundidadeInicial();
441         double z2 = trecho->ProfundidadeFinal();
442         double dExt = trecho->DiametroExterno();
443         QString nomeFluido = QString::fromStdString(trecho->Fluido()->Nome());
444
445         if (!mapaCores.contains(nomeFluido)) {
446             mapaCores[nomeFluido] = coresDisponiveis[corIndex++ % coresDisponiveis.size()];
447         }
448         QColor corFluido = mapaCores[nomeFluido];
449
450         // === Retângulo cinza (buraco do poço) ===
451         QCPIItemRect *rectBuraco = new QCPIItemRect(ui->customPlotPoco);
452         rectBuraco->topLeft->setCoords(-diametroBuraco / 2.0, z1);
453         rectBuraco->bottomRight->setCoords(diametroBuraco / 2.0, z2);
454         rectBuraco->setPen(QPen(Qt::black));
455         rectBuraco->setBrush(QBrush(QColor(150, 150, 150, 100)));
456
457         // === Retângulo da seção (fluido) ===
458         QCPIItemRect *rectSecao = new QCPIItemRect(ui->customPlotPoco);
459         rectSecao->topLeft->setCoords(-dExt / 2.0, z1);
460         rectSecao->bottomRight->setCoords(dExt / 2.0, z2);
461         rectSecao->setPen(QPen(Qt::black));
462         rectSecao->setBrush(QBrush(corFluido));
463
464         // === Rótulo ===
465         QCPIItemText *label = new QCPIItemText(ui->customPlotPoco);
466         label->position->setCoords(0, (z1 + z2) / 2.0);
467         label->setText(nomeFluido);
468         label->setFont(QFont("Arial", 10, QFont::Bold));
469         label->setColor(Qt::black);
470         label->setPositionAlignment(Qt::AlignCenter);
471     }
472     // 6. Desenhar packer se existir
473     bool haPacker = poco->Packer();
474     if (haPacker == true) {
475
476         // Pega a profundidade do último trecho como profundidade do packer
477         double profundidadePacker = 0.0;
478         if (!poco->Trechos().empty()) {
479             profundidadePacker = poco->Trechos().back()->ProfundidadeFinal();
480         }
481
482         double alturaPacker = std::max(profundidadeMaxima * 0.01, 12.0);
483         double zTop, zBottom;
484
485         // Se o packer estiver exatamente na profundidade máxima, desenha ele todo acima
486         if (std::abs(profundidadePacker - profundidadeMaxima) < 1e-3) {
487             zBottom = profundidadePacker;
488             zTop = profundidadePacker - alturaPacker;
489         } else {
490             zTop = profundidadePacker - alturaPacker / 2.0;
491             zBottom = profundidadePacker + alturaPacker / 2.0;
492         }
493

```

```

494 // Encontrar o trecho correspondente profundidade do packer
495 double diametroNoPacker = 0.0;
496 for (const auto& trecho : poco->Trechos()) {
497     if (profundidadePacker >= trecho->ProfundidadeInicial() &&
498         profundidadePacker <= trecho->ProfundidadeFinal()) {
499         diametroNoPacker = trecho->DiametroExterno();
500         break;
501     }
502 }
503
504 // Se n o achou trecho correspondente, usa o maior conhecido como fallback
505 if (diametroNoPacker == 0.0)
506     diametroNoPacker = maiorDiametroExterno;
507
508 // Coordenadas horizontais para os quadrados laterais
509 double xEsq1 = -diametroBuraco / 2.0;
510 double xEsq2 = -diametroNoPacker / 2.0;
511 double xDir1 = diametroNoPacker / 2.0;
512 double xDir2 = diametroBuraco / 2.0;
513
514 // === Quadrado esquerdo ===
515 QCPIItemRect* rectPackerEsq = new QCPIItemRect(ui->customPlotPoco);
516 rectPackerEsq->topLeft->setCoords(xEsq1, zTop);
517 rectPackerEsq->bottomRight->setCoords(xEsq2, zBottom);
518 rectPackerEsq->setPen(QPen(Qt::red, 1.5));
519 rectPackerEsq->setBrush(Qt::NoBrush);
520
521 // === Quadrado direito ===
522 QCPIItemRect* rectPackerDir = new QCPIItemRect(ui->customPlotPoco);
523 rectPackerDir->topLeft->setCoords(xDir1, zTop);
524 rectPackerDir->bottomRight->setCoords(xDir2, zBottom);
525 rectPackerDir->setPen(QPen(Qt::red, 1.5));
526 rectPackerDir->setBrush(Qt::NoBrush);
527
528 // === X vermelho esquerdo ===
529 QCPIItemLine* linh1Esq = new QCPIItemLine(ui->customPlotPoco);
530 linh1Esq->start->setCoords(xEsq1, zTop);
531 linh1Esq->end->setCoords(xEsq2, zBottom);
532 linh1Esq->setPen(QPen(Qt::red, 1.5));
533
534 QCPIItemLine* linh2Esq = new QCPIItemLine(ui->customPlotPoco);
535 linh2Esq->start->setCoords(xEsq2, zTop);
536 linh2Esq->end->setCoords(xEsq1, zBottom);
537 linh2Esq->setPen(QPen(Qt::red, 1.5));
538
539 // === X vermelho direito ===
540 QCPIItemLine* linh1Dir = new QCPIItemLine(ui->customPlotPoco);
541 linh1Dir->start->setCoords(xDir1, zTop);
542 linh1Dir->end->setCoords(xDir2, zBottom);
543 linh1Dir->setPen(QPen(Qt::red, 1.5));
544
545 QCPIItemLine* linh2Dir = new QCPIItemLine(ui->customPlotPoco);
546 linh2Dir->start->setCoords(xDir2, zTop);
547 linh2Dir->end->setCoords(xDir1, zBottom);
548 linh2Dir->setPen(QPen(Qt::red, 1.5));
549 }
550
551 // 7. Linha tracejada indicando profundidade de medi o, se v lida
552 bool ok = false;
553 double profundidadeMedicao = ui->editProfundidadeMedicao->text().toDouble(&ok);

```

```

554
555 // So desenha se a conversao foi bem-sucedida e o valor for maior que zero
556 if (ok && profundidadeMedicao > 0.0) {
557     QCPItemLine* linhaMedicao = new QCPItemLine(ui->customPlotPoco);
558     linhaMedicao->start->setCoords(-larguraGrafico / 2.0, profundidadeMedicao);
559     linhaMedicao->end->setCoords(larguraGrafico / 2.0, profundidadeMedicao);
560
561     // Define o estílo como linha tracejada preta
562     QPen pen(Qt::black);
563     pen.setStyle(Qt::DashLine);
564     pen.setWidthF(1.5);
565     linhaMedicao->setPen(pen);
566 }
567
568 ui->customPlotPoco->replot();
569 }
570
571 void CSimuladorPerdaTubulacao::on_btnCalcularVariacoes_clicked()
572 {
573
574     double pressaoCabeça = ui->editPressaoSupFinal->text().toDouble();
575
576     QString profundidadeStr = ui->editProfundidadeMedicao->text();
577     double profundidade = profundidadeStr.toDouble();
578
579     ui->lbnPressaoHidroestatica->setText(QString::number( (poco->
580         PressaoHidroestaticaNoPonto(profundidade)) ));
581     ui->lbnCargaInicial->setText(QString::number(poco->Carga(profundidade, true)));
582
583     ui->lbnTituloDeltaLTemperatura->setText(QString::number(poco->DeltaLTemperatura(
584         profundidade)));
585     ui->lbnCargaInjecãoColunaFixa->setText(QString::number(poco->CargaInjecão(
586         profundidade)));
587     ui->lbnCargaInjecãoColunaLivre->setText(QString::number(poco->Carga(profundidade,
588         false)));
589     ui->lbnDeltaLPistaoPacker->setText(QString::number(poco->DeltaLPistaoPacker(
590         profundidade)));
591     ui->lbnTituloDeltaLBalao->setText(QString::number(poco->DeltaLEfeitoBalao(
592         profundidade)));
593     ui->lbnDeltaLPistaoCrossover->setText(QString::number(poco->DeltaLPistaoCrossover(
594         profundidade)));
595     ui->lbnDeltaLForcaRestauradora->setText(QString::number(poco->DeltaLForcaRestauradora(
596         profundidade)));
597
598 }
599
600
601
602 void CSimuladorPerdaTubulacao::on_actionArquivo_Dat_triggered()
603 {
604     // Abre uma janelinha pro usuário escolher o arquivo .dat
605     QString caminhoDoArquivo = QFileDialog::getOpenFileName(
606         this,
607         "Selecione um arquivo",
608         "",
609         "Todos os arquivos (*.*)"
610     );
611
612     // Verifica se o usuário não escolheu nada
613     if (caminhoDoArquivo.isEmpty())
614         return;
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
999

```

```

606     std::string caminhoDoArquivoStr = caminhoDoArquivo.toStdString();
607     std::ifstream file(caminhoDoArquivoStr);
608
609     if (!file.is_open()) {
610         ui->statusbar->showMessage("Falha ao abrir o arquivo!");
611         return;
612     }
613
614     std::string linha;
615     bool lendoTrechos = false;
616     bool leuPoco = false;
617     bool leuTrecho = false;
618
619     while (std::getline(file, linha)) {
620         // Verifica se chegou na parte de trechos e fluidos
621         if (linha.find("Configuracao dos Fluidos") != std::string::npos) {
622             lendoTrechos = true;
623             continue;
624         }
625
626         // Ignora linhas vazias ou de comentario
627         if (linha.empty() || linha[0] == '#') {
628             continue;
629         }
630
631         if (!lendoTrechos) {
632             // Aqui lemos a linha com os dados principais do p o
633             std::istringstream iss(linha);
634             std::string nome;
635             double profundidade, diamPoco, pressaoSup, pressaoSupFim;
636             double temperaturaSuperiorInicial, temperaturaFundoInicial;
637             double temperaturaSuperiorFinal, temperaturaFundoFinal;
638             std::string strHaPacker;
639
640             if (iss >> nome >> profundidade >> diamPoco >> pressaoSup >> pressaoSupFim
641                 >> temperaturaSuperiorInicial >> temperaturaFundoInicial
642                 >> temperaturaSuperiorFinal >> temperaturaFundoFinal >> strHaPacker) {
643
644                 bool haPacker = (strHaPacker == "true" || strHaPacker == "1");
645
646                 ui->checkBoxPacker->setChecked(haPacker);
647                 ui->btnAdicionarTrecho->setEnabled(true);
648                 ui->btnRemoverTrecho->setEnabled(true);
649                 ui->btnCalcularVariacoes->setEnabled(true);
650
651                 poco = std::make_unique<CObjetoPoco>(
652                     CObjetoPoco::CriarParaModulo02(nome, profundidade, diamPoco,
653                                         pressaoSup, pressaoSupFim,
654                                         temperaturaSuperiorInicial,
655                                         temperaturaFundoInicial,
656                                         temperaturaSuperiorFinal,
657                                         temperaturaFundoFinal, haPacker)
658                 );
659
660                 leuPoco = true;
661
662             } else {
663                 std::cerr << "Erro ao ler linha de p o " << linha << std::endl;
664             }
665
666         }

```

```

663     } else {
664         // Aqui lemos os trechos e fluidos do p o o
665         std::istringstream iss(linha);
666         std::string nomeTrecho, nomeFluido;
667         double profundInicial, profundFinal;
668         double diametroExterno, diametroInterno;
669         double coeficientePoisson, coeficienteExpansaoTermica;
670         double moduloElasticidade, pesoUnidade;
671         double densidade, viscosidade;
672
673         if (iss >> nomeTrecho >> profundInicial >> profundFinal
674             >> diametroExterno >> diametroInterno
675             >> coeficientePoisson >> coeficienteExpansaoTermica
676             >> moduloElasticidade >> pesoUnidade
677             >> nomeFluido >> densidade >> viscosidade) {
678
679             auto fluido = std::make_unique<CFluido>(nomeFluido, densidade,
680                                                 viscosidade);
681             auto trechoPoco = std::make_unique<CTrechoPoco>(nomeTrecho,
682                                                 profundInicial,
683                                                 profundFinal, std::
684                                                 move(fluido),
685                                                 diametroExterno,
686                                                 diametroInterno,
687                                                 coeficientePoisson,
688                                                 coeficienteExpansaoTermica
689                                                 ,
690                                                 moduloElasticidade,
691                                                 pesoUnidade
692                                                 );
693
694         if (!poco->AdicionarTrechoPoco(std::move(trechoPoco))) {
695             std::cerr << "Falha ao adicionar trecho ao p o o.\n";
696         } else {
697             leuTrecho = true;
698         }
699     }
700
701     file.close();
702
703     // Se nao leu nem o p o o ou nenhum trecho, mostra alerta
704     if (!leuPoco || !leuTrecho) {
705         QMessageBox::warning(this, "Arquivo Incorreto",
706                             "Aten o : o o arquivo selecionado n o est no formato
707                             esperado.\n"
708                             "Por favor, abra um arquivo de configura o v lido.");
709     }
710
711     AtualizarDados();
712     ui->statusbar->showMessage("Dados importados com sucesso!");
713
714

```

```

715 void CSimuladorPerdaTubulacao::on_actionNova_Simula_o_triggered()
716 {
717     QMessageBox::StandardButton resposta = QMessageBox::question(
718         this,
719         "",
720         "Tem certeza que deseja iniciar uma nova simulação?",
721         QMessageBox::Yes | QMessageBox::No
722     );
723
724     if (resposta == QMessageBox::Yes) {
725         CSimuladorPerdaTubulacao *newWindow = new CSimuladorPerdaTubulacao();
726         newWindow->show();
727         this->close();
728     }
729 }
730
731
732 void CSimuladorPerdaTubulacao::on_actionExportar_Como_Imagem_triggered()
733 {
734     QString fileName = QFileDialog::getSaveFileName(this, "Salvar imagem", "", "PNG (*.png);;JPEG (*.jpg)");
735
736     if (!fileName.isEmpty()) {
737         QPixmap pixmap = this->grab();
738         pixmap.save(fileName);
739     }
740 }
741
742
743 void CSimuladorPerdaTubulacao::on_actionSobre_o_SEEP_triggered()
744 {
745     CJanelaSobreSofware janelaSobre;
746     janelaSobre.setWindowTitle("Sobre o Software");
747     janelaSobre.exec();
748 }
749
750
751 void CSimuladorPerdaTubulacao::SalvarArquivo(bool salvarComo)
752 {
753     QString caminho;
754
755     // Se for salvarComo ou ainda não tiver caminho, abrir o diretório
756     if (salvarComo || CaminhoArquivo().isEmpty()) {
757         caminho = QFileDialog::getSaveFileName(this, "Salvar Arquivo", "", "Arquivo DAT (*.dat)");
758         if (caminho.isEmpty()) return; // usuário cancelou
759         CaminhoArquivo(caminho);
760         NomeArquivo(QFileInfo(caminho).fileName());
761     } else {
762         caminho = CaminhoArquivo(); // salva direto
763     }
764
765     QFile arquivo(caminho);
766     if (!arquivo.open(QIODevice::WriteOnly | QIODevice::Text)) {
767         QMessageBox::warning(this, "Erro", "Não foi possível salvar o arquivo.");
768         return;
769     }
770
771     QTextStream out(&arquivo);
772

```

```

773     out << "#_Configuracao do Poco"
774     -----
775     n";
774     out << "#"
775     << QString("Nome").leftJustified(35, ' ')
776     << QString("Profundidade (ft)").leftJustified(35, ' ')
777     << QString("Diametro do Poco (ft)").leftJustified(35, ' ')
778     << QString("Pressao Sup. Inicial (psi)").leftJustified(35, ' ')
779     << QString("Pressao Sup. Final (psi)").leftJustified(35, ' ')
780     << QString("Tempo Sup. Inicial ( F )").leftJustified(35, ' ')
781     << QString("Tempo Fund. Inicial ( F )").leftJustified(35, ' ')
782     << QString("Tempo Sup. Final ( F )").leftJustified(35, ' ')
783     << QString("Tempo Fund. Final ( F )").leftJustified(35, ' ')
784     << QString("Prof Packer (ft)").leftJustified(35, ' ')
785     << "\n";
786
787     out << "##"
788     << ui->editNomePoco->text().leftJustified(35, ' ')
789     << ui->editProfundidadeTotal->text().leftJustified(35, ' ')
790     << ui->editDiametroPoco->text().leftJustified(35, ' ')
791     << ui->editPressaoSupInicial->text().leftJustified(35, ' ')
792     << ui->editPressaoSupFinal->text().leftJustified(35, ' ')
793     << ui->editTemperaturaSuperiorInicial->text().leftJustified(35, ' ')
794     << ui->editTemperaturaFundoInicial->text().leftJustified(35, ' ')
795     << ui->editTemperaturaSuperiorFinal->text().leftJustified(35, ' ')
796     << ui->editTemperaturaFundoFinal->text().leftJustified(35, ' ');
797
798 // Checkbox de saida
799 if (ui->checkBoxPacker->isChecked()) {
800     out << "true";
801 } else {
802     out << "false";
803 }
804
805     out << "\n";
806
807
808 // Escreve os dados dos fluidos
809 out << "\n\n\n#_Configuracao dos Fluidos"
810 -----
811     n";
810     out << "#"
811     << QString("Nome Trecho").leftJustified(25, ' ')
812     << QString("Prof. Inicial (ft)").leftJustified(25, ' ')
813     << QString("Prof. Final (ft)").leftJustified(25, ' ')
814     << QString("Diam. externo (in)").leftJustified(25, ' ')
815     << QString("Diam. interno (in)").leftJustified(25, ' ')
816     << QString("Coef. Poisson").leftJustified(25, ' ')
817     << QString("Coef. Exp. Term. (1/F)").leftJustified(25, ' ')
818     << QString("Mod. Elast. (psi)").leftJustified(25, ' ')
819     << QString("Peso/unid. (lb/ft)").leftJustified(25, ' ')
820     << QString("Nome fluido").leftJustified(25, ' ')
821     << QString("Densidade (lbm/gal)").leftJustified(25, ' ')
822     << QString("Viscosidade (cP)").leftJustified(25, ' ')
823     << "\n";
824
825     int linhas = ui->tblFluidos->rowCount();
826     for (int i = 0; i < linhas; ++i) {
827         out << "##" // recuo
828         << ui->tblTrechos->item(i, 0)->text().leftJustified(25, ' ')

```

```

829     << ui->tblTreichos->item(i, 1)->text().leftJustified(25, 'L')
830     << ui->tblTreichos->item(i, 2)->text().leftJustified(25, 'L')
831     << ui->tblTreichos->item(i, 3)->text().leftJustified(25, 'L')
832     << ui->tblTreichos->item(i, 4)->text().leftJustified(25, 'L')
833     << ui->tblTreichos->item(i, 5)->text().leftJustified(25, 'L')
834     << ui->tblTreichos->item(i, 6)->text().leftJustified(25, 'L')
835     << ui->tblTreichos->item(i, 7)->text().leftJustified(25, 'L')
836     << ui->tblTreichos->item(i, 8)->text().leftJustified(25, 'L')
837     << ui->tblFluidos->item(i, 0)->text().leftJustified(25, 'L')
838     << ui->tblFluidos->item(i, 1)->text().leftJustified(25, 'L')
839     << ui->tblFluidos->item(i, 2)->text().leftJustified(25, 'L')
840     << "\n";
841 }
842
843 arquivo.close();
844
845 // Atualiza o caminho salvo apenas se for novo
846 if (CaminhoArquivo().isEmpty())
847 {
848     CaminhoArquivo(caminho);
849     NomeArquivo(QFileInfo(caminho).fileName());
850 }
851
852 QMessageBox::information(this, "Salvo", "Arquivo salvo com sucesso!");
853 }
854
855 void CSimuladorPerdaTubulacao::on_actionSalvar_triggered()
856 {
857     SalvarArquivo(false); // salvar direto
858 }
859
860 void CSimuladorPerdaTubulacao::on_actionSalvar_como_triggered()
861 {
862     SalvarArquivo(true); // for ar abrir QFileDialog
863 }
864
865 // essa funcao edita os dados do fluido e do trecho com base na linha da tabela de
866 // fluidos
867 void CSimuladorPerdaTubulacao::EditarLinhaTabela(int row)
868 {
869     // garante que o indice da linha v lido
870     if (row < 0 || row >= poco->Treichos().size()) {
871         return;
872     }
873
874     // obtém o trecho e fluido da mesma linha
875     CTrechoPoco* trecho = poco->Treichos().at(row);
876     CFluido* fluido = trecho->Fluido();
877
878     if (!fluido) return;
879
880     trecho->Nome(ui->tblTreichos->item(row, 0)->text().toString());
881     trecho->ProfundidadeInicial(ui->tblTreichos->item(row, 1)->text().toDouble());
882     trecho->ProfundidadeFinal(ui->tblTreichos->item(row, 2)->text().toDouble());
883     trecho->DiametroExterno(ui->tblTreichos->item(row, 3)->text().toDouble());
884     trecho->DiametroInterno(ui->tblTreichos->item(row, 4)->text().toDouble());
885     trecho->CoeficientePoisson(ui->tblTreichos->item(row, 5)->text().toDouble());
886     trecho->CoeficienteExpancaoTermica(ui->tblTreichos->item(row, 6)->text().toDouble());
887     trecho->ModuloElasticidade(ui->tblTreichos->item(row, 7)->text().toDouble());
888     trecho->PesoUnidade(ui->tblTreichos->item(row, 8)->text().toDouble());

```

```

888
889     fluido->Nome(ui->tblFluidos->item(row, 0)->text().toStdString());
890     fluido->Densidade(ui->tblFluidos->item(row, 1)->text().toDouble());
891     fluido->Viscosidade(ui->tblFluidos->item(row, 2)->text().toDouble());
892
893     // atualiza valores globais do simulador
894     AtualizarDados();
895
896     ui->statusbar->showMessage("Fluido e profundidades atualizados com sucesso!");
897 }
898
899 void CSimuladorPerdaTubulacao::on_actionManual_do_Usu_rio_triggered()
900 {
901     QString caminho = QCoreApplication::applicationDirPath() + "/PDFs/ManualUsuario.pdf";
902     QDesktopServices::openUrl(QUrl::fromLocalFile(caminho));
903 }
904
905
906 void CSimuladorPerdaTubulacao::on_actionF_rmulas_Utilizadas_triggered()
907 {
908     QString caminho = QCoreApplication::applicationDirPath() + "/PDFs/FormulasUtilizadas.pdf";
909     QDesktopServices::openUrl(QUrl::fromLocalFile(caminho));
910 }

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 6 o arquivo de cabeçalho da classe CObjetoPoco.
frame

Listing 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 o 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 padro
31     CObjetoPoco() = default;
32     ~CObjetoPoco() = default;
33
34     // Evita c pia (pra evitar duplica o desnecessaria dos trechos)
35     CObjetoPoco(const CObjetoPoco&) = delete;
36     CObjetoPoco& operator=(const CObjetoPoco&) = delete;
37
38     // Permite movimenta o (move semantics)
39     CObjetoPoco(CObjetoPoco&&) = default;
40     CObjetoPoco& operator=(CObjetoPoco&&) = default;
41
42     // Metodos de cria o est ticos, separando claramente os dados exigidos por cada
43     // m dulo
44     static CObjetoPoco CriarParaModulo01(std::string Nome, double Profund, double
45         PressaoSup, double D, double OD, double ID, double q);
46     static CObjetoPoco CriarParaModulo02(std::string Nome, double Profund, double
47         diamPoco, double PressaoSup, double PressaoSupFinal, double TempTopoInicial,
48         double TempFundoInicial, double TempTopoFinal, double TempFundoFinal, bool
49         haPacker);
50
51     // Getters para acessar os atributos de forma segura
52     std::string NomePoco() const { return nomePoco; }
53     double ProfundidadeTotal() const { return profundidadeFinal; }
54     double ProfundidadeOcupada() const { return profundidadeOcupada; }
55     double PressaoSuperficie() const { return pressaoSuperficie; }
56     double PressaoSuperficieFim() const { return pressaoSuperficieFim; }
57     double DiametroPoco() const { return diametroPoco; }
58     double DiametroRevestimentoOD() const { return diametroRevestimentoOD; }
59     double DiametroRevestimentoID() const { return diametroRevestimentoID; }
60     double Vazao() const { return vazao; }
61     double TemperaturaTopoInicial() const { return temperaturaTopoInicial; }
62     double TemperaturaFundoInicial() const { return temperaturaFundoInicial; }
63     double TemperaturaTopoFinal() const { return temperaturaTopoFinal; }
64     double TemperaturaFundoFinal() const { return temperaturaFundoFinal; }
65     bool Packer() const { return packer; }
66
67     // Retorna os trechos adicionados ao po o (em forma de ponteiros brutos)
68     std::vector<CTrechoPoco*> Trechos() const;
69
70     // Setters permitem modificar os atributos do po o
71     void NomePoco(std::string Nome) { nomePoco = Nome; }
72     void ProfundidadeTotal(double Profundidade) { profundidadeFinal = Profundidade; }
73     void ProfundidadeOcupada(double Profundidade) { profundidadeOcupada = Profundidade; }
74     void PressaoSuperficie(double PressaoSuperior) { pressaoSuperficie = PressaoSuperior;
75         }
76     void PressaoSuperficieFim(double PressaoSuperior) { pressaoSuperficieFim =
77         PressaoSuperior; }
78     void DiametroPoco(double D) { diametroPoco = D; }
79     void DiametroRevestimentoOD(double DiametroExterno) { diametroRevestimentoOD =
80         DiametroExterno; }
81     void DiametroRevestimentoID(double DiametroInterno) { diametroRevestimentoID =
82         DiametroInterno; }
83     void Vazao(double q) { vazao = q; }
84     void TemperaturaTopoInicial(double temperatura) { temperaturaTopoInicial =
85         temperatura; }
86     void TemperaturaFundoInicial(double temperatura) { temperaturaFundoInicial =
87         temperatura; }

```

```

77     void TemperaturaTopoFinal(double temperatura) { temperaturaTopoFinal = temperatura; }
78     void TemperaturaFundoFinal(double temperatura) { temperaturaFundoFinal = temperatura;
79         }
80     void Packer(bool haPacker) { packer = haPacker; }
81
82     // M etodos de c lculo
83     double PressaoHidroestaticaTotal() const;
84     double PressaoHidroestaticaNoPonto(double profundidade) const;
85     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 deCimaParaBaixo, bool
92         inicio) const;
93     double VariacaoCargaEfeitoPistao(double profundidade, double ID, double OD) const;
94     double DeltaLPistaoPacker(double profundidade) const;
95     double DeltaLPistaoCrossover(double profundidade) const;
96     double DeltaLForcaRestauradora(double profundidade) const;
97     double CargaInjecao(double profundidade) const;
98     double TemperaturaNoPonto(double profundidade, double T_topo, double T_Fundo) const;
99
100    bool AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco> trechoParaAdicionar); // 
101        Fun o para adicionar um novo trecho ao po o
102    void RemoverFluidoPoco(const std::string& nomeFluido); // Remove trecho do po o com
103        base no nome do fluido
104    void RemoverTrechoPoco(const std::string& nomeTrecho);
105
106    // M etodos que retornam dados para gr ficos
107    std::pair<std::vector<double>, std::vector<double>> PlotarProfundidadePorPressao();
108    std::pair<std::vector<double>, std::vector<double>> PlotarProfundidadePorPressaoMedia
109        ();
110};

111#endif

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 7 a implementa o da classe CObjetoPoco.
frame

Listing 7: Arquivo de implementa o 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, double
12     profundidadeFinalDoPoco, double pressaoNaSuperficie,
13         double diametroDoPoco, double
14             diametroRevestimentoExterno, double
15                 
```

```

13     CObjetoPoco objetoPoco;
14
15     objetoPoco.nomePoco = nomeDoPoco;
16     objetoPoco.profundidadeFinal = profundidadeFinalDoPoco;
17     objetoPoco.pressaoSuperficie = pressaoNaSuperficie;
18     objetoPoco.diametroPoco = diametroDoPoco;
19     objetoPoco.diametroRevestimentoOD = diametroRevestimentoExterno;
20     objetoPoco.diametroRevestimentoID = diametroRevestimentoInterno;
21     objetoPoco.vazao = vazaoDoPoco;
22
23     return objetoPoco;
24 }
25
26
27 CObjetoPoco CObjetoPoco::CriarParaModulo02(std::string nomeDoPoco, double
28     profundidadeFinalDoPoco, double diamPoco, double pressaoNaSuperficie, double
29     pressaoNaSuperficieFim,
30                                         double temperaturaTopoInicial, double
31                                         temperaturaFundoInicial,
32                                         double temperaturaTopoFinal, double
33                                         temperaturaFundoFinal, bool haPacker) {
34     CObjetoPoco objetoPoco;
35
36     objetoPoco.nomePoco = nomeDoPoco;
37     objetoPoco.profundidadeFinal = profundidadeFinalDoPoco;
38     objetoPoco.diametroPoco = diamPoco;
39     objetoPoco.pressaoSuperficie = pressaoNaSuperficie;
40     objetoPoco.pressaoSuperficieFim = pressaoNaSuperficieFim;
41     objetoPoco.temperaturaTopoInicial = temperaturaTopoInicial;
42     objetoPoco.temperaturaFundoInicial = temperaturaFundoInicial;
43     objetoPoco.temperaturaTopoFinal = temperaturaTopoFinal;
44     objetoPoco.temperaturaFundoFinal = temperaturaFundoFinal;
45     objetoPoco.packer = haPacker;
46
47     return objetoPoco;
48 }
49
50
51 std::vector<CTrechoPoco*> CObjetoPoco::Treichos() const {
52     std::vector<CTrechoPoco*> vetorDePonteirosParaTreichos;
53
54     // percorre todos os trechos armazenados no p o o
55     for (const auto& trechoUnico : trechos) {
56         // adiciona o ponteiro cru (n o nico ) de cada trecho ao vetor de sa da
57         vetorDePonteirosParaTreichos.push_back(trechoUnico.get());
58     }
59
60     // retorna o vetor contendo os ponteiros dos trechos
61     return vetorDePonteirosParaTreichos;
62 }
63
64 bool CObjetoPoco::AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco> trechoParaAdicionar) {
65     // calcula o comprimento do trecho com base nas profundidades inicial e final
66     double comprimentoDoTrecho = trechoParaAdicionar->ProfundidadeFinal() -
67         trechoParaAdicionar->ProfundidadeInicial();
68
69     // move o trecho para dentro do vetor principal do p o o
70     trechos.push_back(std::move(trechoParaAdicionar));

```

```

66     // atualiza a profundidade total ocupada no po o somando o novo trecho
67     profundadeOcupada += comprimentoDoTrecho;
68
69     return true; // opera o realizado com sucesso
70 }
71
72
73
74 void CObjetoPoco::RemoverFluidoPoco(const std::string& nomeFluido) {
75     for (auto it = trechos.begin(); it != trechos.end();) {
76         if ((*it)->Fluido()->Nome() == nomeFluido) {
77             double comprimento = (*it)->ProfundidadeFinal() - (*it)->ProfundidadeInicial
78                 ();
79             it = trechos.erase(it);
80             profundadeOcupada -= comprimento;
81         } else {
82             ++it;
83         }
84     }
85
86 void CObjetoPoco::RemoverTrechoPoco(const std::string& nomeTrecho) {
87     for (auto it = trechos.begin(); it != trechos.end();) {
88         if ((*it)->Nome() == nomeTrecho) {
89             double comprimento = (*it)->ProfundidadeFinal() - (*it)->ProfundidadeInicial
90                 ();
91             it = trechos.erase(it);
92             profundadeOcupada -= comprimento;
93         } else {
94             ++it;
95         }
96     }
97
98 double CObjetoPoco::PressaoHidroestaticaTotal() const {
99     double pressaoTotalHidrostatica = 0.0;
100
101    // soma a pressao hidrostatica de todos os trechos do po o
102    for (const auto& trechoAtual : trechos) {
103        pressaoTotalHidrostatica += trechoAtual->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 profundidadeDesejada) const {
111     double pressaoAcumulada = pressaoSuperficie;
112     double profundidadeJaCalculada = 0.0;
113
114     // percorre cada trecho verificando se ele contem a profundidade desejada
115     for (const auto& trechoAtual : trechos) {
116         double comprimentoDoTrecho = trechoAtual->ProfundidadeFinal() - trechoAtual->
117             ProfundidadeInicial();
118
119         // se a profundidade estiver dentro do trecho atual
120         if (profundidadeDesejada <= profundidadeJaCalculada + comprimentoDoTrecho) {
121             double profundidadeDentroDoTrecho = profundidadeDesejada -
122                 profundidadeJaCalculada;

```

```

122         // adiciona somente a parte proporcional da pressao no trecho
123         pressaoAcumulada += trechoAtual->PressaoHidroestatica(
124             profundidadeDentroDoTrecho);
125         break;
126     } else {
127         // adiciona a pressao total do trecho completo
128         pressaoAcumulada += trechoAtual->PressaoHidroestatica();
129         profundidadeJaCalculada += comprimentoDoTrecho;
130     }
131 }
132 return pressaoAcumulada;
133 }

134

135 bool CObjetoPoco::VerificarPreenchimentoColuna() {
136     double profundidadeNaoPreenchida = profundidadeFinal - profundidadeOcupada;
137
138     if (profundidadeNaoPreenchida > 0) {
139         std::cout << "Uma coluna de fluido precisa ser adicionada!" << std::endl;
140         return false; // coluna incompleta
141     } else {
142         std::cout << "A coluna de fluidos equivale a profundade total do po o!" <<
143             std::endl;
144         return true; // coluna completamente preenchida
145     }
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() - trechoAtual->
154             ProfundidadeInicial();
155
156         // multiplica a densidade equivalente pelo comprimento do trecho para ponderar
157         somaDensidadePonderada += trechoAtual->DensidadeEquivalente() * comprimentoTrecho
158         ;
159         comprimentoTotalDaColuna += comprimentoTrecho;
160     }
161
162     return somaDensidadePonderada / comprimentoTotalDaColuna;
163 }
164

165 double CObjetoPoco::ViscosidadeEfetivaTotal() const {
166     double somaDasViscosidades = 0.0;
167
168     // percorre todos os trechos para somar as viscosidades dos fluidos
169     for (const auto& trechoAtual : trechos) {
170         somaDasViscosidades += trechoAtual->Fluido()->Viscosidade();
171     }
172
173     // retorna a media simples das viscosidades
174     return somaDasViscosidades / trechos.size();
175 }

176 std::pair<std::vector<double>, std::vector<double>> CObjetoPoco::
177 PlotarProfundidadePorPressaoMedia() {

```

```

176     std::vector<double> vetorDeProfundidades;
177     std::vector<double> vetorDePressoes;
178
179     // percorre cada metro ao longo da profundidade total do po o
180     for (double profundidadeAtual = 0.0; profundidadeAtual <= ProfundidadeTotal();
181         profundidadeAtual += 1.0) {
182         vetorDeProfundidades.push_back(profundidadeAtual);
183
184         // calcula a pressao no ponto atual usando a funcao apropriada
185         double pressaoNoPonto = PressaoHidroestaticaNoPonto(profundidadeAtual);
186         vetorDePressoes.push_back(pressaoNoPonto);
187     }
188
189     // retorna as duas listas para serem usadas na geracao do grafico
190     return std::make_pair(vetorDeProfundidades, vetorDePressoes);
191 }
192
193 double CObjetoPoco::Carga(double profundidade, bool inicio) const {
194     double cargaTotal = 0.0;
195     double pi = 3.141592653589793;
196
197     if (trechos.empty())
198         return 0.0;
199
200     // 1. Pressao no fundo do poco (base)
201     double profundidadeBase = 0.0;
202     double OD_base = 0.0;
203     double ID_base = 0.0;
204
205     for (const auto& trecho : trechos) {
206         if (trecho->ProfundidadeFinal() > profundidadeBase) {
207             profundidadeBase = trecho->ProfundidadeFinal();
208             OD_base = trecho->DiametroExterno();
209             ID_base = trecho->DiametroInterno();
210         }
211     }
212
213     // 2. Carga por pressao no fundo (negativa)
214     double pressaoBase = PressaoHidroestaticaNoPonto(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 * ID_base);
221     double cargaPressao = -1.0 * pressaoBase * areaBase;
222     cargaTotal += cargaPressao;
223
224     // 3. Soma pesos de trechos abaixo da profundidade
225     for (const auto& trecho : trechos) {
226         double z_i = trecho->ProfundidadeInicial();
227         double z_f = trecho->ProfundidadeFinal();
228         double pesoUnit = trecho->PesoUnidade();
229
230         // Caso 1: trecho totalmente abaixo da profundidade
231         if (z_i >= profundidade) {
232             double L_total = z_f - z_i;
233             double cargaPeso = L_total * pesoUnit;
234             cargaTotal += cargaPeso;
235         }

```

```

234     // Caso 2: profundidade dentro do trecho
235     else if (profundidade > z_i && profundidade < z_f) {
236         double L_parcial = z_f - profundidade;
237         double cargaPeso = L_parcial * pesoUnit;
238         cargaTotal += cargaPeso;
239     }
240 }
241 }
242 }
243 }
244
245 // 4. considerando as Cargas por efeito pist o
246 if (inicio){
247     cargaTotal += VariacaoCargaDevidoCrossover(profundidade , false , true);
248 }
249 else{
250     cargaTotal += VariacaoCargaDevidoCrossover(profundidade , false , false);
251 }
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 profundidade informada
265         if (z_i >= profundidade)
266             continue;
267
268         // Calcula temperatura media inicial e final do trecho
269         double tMedioInicial = (
270             TemperaturaNoPonto(z_i, TemperaturaTopoInicial(),
271                                 TemperaturaFundoInicial()) +
272             TemperaturaNoPonto(z_f, TemperaturaTopoInicial(),
273                                 TemperaturaFundoInicial())
274             ) / 2.0;
275
276         double tMedioFinal = (
277             TemperaturaNoPonto(z_i, TemperaturaTopoFinal(),
278                                 TemperaturaFundoFinal()) +
279             TemperaturaNoPonto(z_f, TemperaturaTopoFinal(),
280                                 TemperaturaFundoFinal())
281             ) / 2.0;
282
283         // Variacao de temperatura final - inicial
284         double deltaT = tMedioFinal - tMedioInicial;
285
286         // Caso raro onde nao houve variacao de temperatura entre as duas condicoes
287         // A gente for a um deltaT usando a temperatura no ponto zero como referencia
288         if (deltaT == 0) {
289             double media = (
290                 TemperaturaNoPonto(z_i, TemperaturaTopoFinal(),
291                                     TemperaturaFundoFinal()) +
292                 TemperaturaNoPonto(z_f, TemperaturaTopoFinal(),
293                                     TemperaturaFundoFinal())
294             )
295             / 2.0;
296
297             deltaT = media - tMedioInicial;
298         }
299
300         deltaLTotal += deltaT * ct;
301     }
302 }
303
304 return deltaLTotal;
305 }

```

```

288             ) / 2.0;
289
290         deltaT = TemperaturaNoPonto(0, TemperaturaTopoFinal(), TemperaturaFundoFinal()
291             () - media;
292     }
293
294     double L = 0.0;
295
296     // Se a profundidade estiver dentro do trecho, calcula o comprimento parcial
297     if (profundidade > z_i && profundidade < z_f) {
298         L = profundidade - z_i;
299     }
300     // Se o trecho estiver totalmente acima da profundidade, considera L total
301     else if (z_f <= profundidade) {
302         L = z_f - z_i;
303     }
304
305     // Soma o deltaL do trecho na variavel acumuladora
306     deltaLTotal += ct * L * deltaT;
307 }
308
309     return deltaLTotal;
310 }
311
312
313
314 double CObjetoPoco::TemperaturaNoPonto(double profundidade, double T_topo, double T_Fundo
315     ) 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 profundidade, bool
323     deCimaParaBaixo, bool inicio) const {
324
325     double pi = 3.141592653589793;
326     double variacao_total = 0.0;
327
328     // tolerancia para comparacoes de ponto flutuante
329     constexpr double tolerancia = 1e-5;
330
331     if (trechos.size() < 2)
332         return 0.0;
333
334     // percorre todas as interfaces entre os trechos consecutivos
335     for (size_t i = 1; i < trechos.size(); ++i) {
336         const auto& trecho_acima = trechos[i - 1];
337         const auto& trecho_abixo = trechos[i];
338
339         double z_interface = trecho_abixo->ProfundidadeInicial();
340
341         // define se a interface deve ser considerada com base na direcao da analise
342         bool considerar = false;
343         if (!deCimaParaBaixo && z_interface >= profundidade - tolerancia)
344             considerar = true;
345         if (deCimaParaBaixo && z_interface <= profundidade + tolerancia)
346             considerar = true;

```

```

345
346     if (!considerar)
347         continue;
348
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 * ID_abixo);
363     double Ao = (pi / 4.0) * (OD_acima * OD_acima - OD_abixo * OD_abixo);
364
365     // pressao no ponto da interface
366     double pressao = PressaoHidroestaticaNoPonto(z_interface);
367     double carga_crossover = 0.0;
368
369     // tratamento com ou sem packer
370     if (Packer()) {
371         if (inicio) {
372             // no inicio da coluna: pressao interna = externa = pressao hidrostatica
373             carga_crossover = pressao * (Ai - Ao);
374         } else {
375             // no fim da coluna: pressao interna = pressao hidrostatica + pressao
376             // superficial final
377             carga_crossover = (pressao - (pressao + PressaoSuperficieFim())) * Ai;
378         }
379     } else {
380         // sem packer: pressao interna e externa sao iguais
381         carga_crossover = pressao * (Ai - Ao);
382     }
383
384     variacao_total += carga_crossover;
385 }
386
387 }
388
389
390 #include <QDebug>
391 double CObjetoPoco::VariacaoCargaEfeitoPistao(double profundidade, double ID, double OD)
392 {
393     const {
394         const double pi = 3.141592653589793;
395
396         // rea da parede interna do tubo
397         double Ain = (pi / 4.0) * (OD * OD - ID * ID);
398
399         double Aout = Ain;
400
401         // Press es internas
402         double Pin_inicio = PressaoHidroestaticaNoPonto(profundidade) + PressaoSuperficie();
```

```

403     double Pin_fim = PressaoHidroestaticaNoPonto(profundidade) + PressaoSuperficieFim();
404     double deltaPin = Pin_fim - Pin_inicio;
405
406     // Pressões externas
407     double Pout_inicio;
408     double Pout_fim;
409
410     if (Packer() == true) {
411         Pout_inicio = 0;
412         Pout_fim = 0;
413     } else {
414         Pout_inicio = 0;
415         Pout_fim = 0;
416     }
417
418     double deltaPout = Pout_fim - Pout_inicio;
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 mudança de diâmetro externo entre os trechos
434     bool possuiDiferencaOD = false;
435     double primeiroOD = trechos[0]->DiametroExterno();
436     for (const auto& trecho : trechos) {
437         if (std::abs(trecho->DiametroExterno() - primeiroOD) > 1e-6) {
438             possuiDiferencaOD = true;
439             break;
440         }
441     }
442
443     for (size_t i = 0; i < trechos.size(); ++i) {
444         const auto& trecho = trechos[i];
445         double z_i = trecho->ProfundidadeInicial();
446         double z_f = trecho->ProfundidadeFinal();
447         double OD = trecho->DiametroExterno();
448         double ID = trecho->DiametroInterno();
449         double E = trecho->ModuloElasticidade();
450
451         double area_anular = (pi / 4.0) * (OD * OD - ID * ID);
452         if (E <= 0.0 || area_anular <= 0.0)
453             continue;
454
455         double L = 0.0;
456
457         if (profundidade > z_i && profundidade < z_f) {
458             L = profundidade - z_i;
459         } else if (z_f <= profundidade) {
460             L = z_f - z_i;
461         } else {
462             continue;

```

```

463     }
464
465     double CargaPistao = 0.0;
466     if (possuiDiferencaOD) {
467         // qualquer mudança 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 diâmetro total do poço
471         CargaPistao = VariacaoCargaEfeitoPistao(z_i, ID, DiametroPoco());
472     }
473
474     double deltaL_trecho = (CargaPistao * L) / (E * area_anular);
475     deltaL_total += deltaL_trecho;
476 }
477
478 return deltaL_total;
479 }
480
481
482
483 double CObjetoPoco::DeltaLEfeitoBalão(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->DiâmetroExterno();
492         double ID = trecho->DiâmetroInterno();
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 * area_anular); // o x12
515         // uma unidade de conversão de ft para in
516         deltaL_total += deltaL_trecho;
517     }
518
519     return deltaL_total;
520 }
521

```

```

522 double CObjetoPoco::DeltaLPistaoCrossover(double profundidade) const {
523     double pi = 3.141592653589793;
524     double deltaL_total = 0.0;
525
526     for (size_t i = 1; i < trechos.size(); ++i) {
527         const auto& trechoAcima = trechos[i - 1];
528         const auto& trechoAbaixo = trechos[i];
529
530         // Profundidade onde ocorre a mudança de trecho
531         double z_interface = trechoAbaixo->ProfundidadeInicial();
532
533         // Ignora se a interface estiver abaixo da profundidade analisada
534         if (z_interface > profundidade)
535             continue;
536
537         // Coleta os diametros externos para comparar se ha mudança
538         double OD_acima = trechoAcima->DiametroExterno();
539         double ID_acima = trechoAcima->DiametroInterno();
540         double OD_abaiixo = trechoAbaixo->DiametroExterno();
541
542         // Se os diametros forem quase iguais, nao ha efeito pistao relevante
543         if (std::abs(OD_acima - OD_abaiixo) < 1e-6)
544             continue;
545
546         // Area da secao transversal da coluna acima (quem vai deformar)
547         double area_secao = (pi / 4.0) * (OD_acima * OD_acima - ID_acima * ID_acima);
548
549         // Obtem o modulo de elasticidade medio entre os dois trechos
550         double E1 = trechoAcima->ModuloElasticidade();
551         double E2 = trechoAbaixo->ModuloElasticidade();
552         double E_medio = (E1 + E2) / 2.0;
553
554         if (E_medio <= 0.0 || area_secao <= 0.0)
555             continue;
556
557         // Carga causada pelo efeito pistao na interface
558         double cargaPistao = VariacaoCargaEfeitoPistao(z_interface, OD_abaiixo, OD_acima);
559
560         // L = comprimento da coluna acima da interface
561         double L = z_interface;
562
563         // Deformacao axial provocada pela carga
564         double deltaL = (cargaPistao * L) / (E_medio * area_secao);
565         deltaL_total += deltaL;
566     }
567
568     return deltaL_total;
569 }
570
571 double CObjetoPoco::DeltaLForcaRestauradora(double profundidade) const{
572
573     double deltaLTemperatura = DeltaLTemperatura(profundidade);
574     double deltaLBalao = DeltaLEfeitoBalao(profundidade);
575     double deltaLPacker = DeltaLPistaoPacker(profundidade);
576     double deltaLCrossover = DeltaLPistaoCrossover(profundidade);
577
578     return deltaLTemperatura + deltaLBalao + deltaLPacker + deltaLCrossover;
579 }
580 }
```

```

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 8 o arquivo de cabeçalho da classe CFluido.frame

Listing 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 9 a implementação da classe CFluido.
frame

Listing 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 10 o arquivo de cabeçalho da classe CModeloReológico.
frame

Listing 10: Arquivo de implementação da classe CModeloReológico

```

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 CModeloReológico {
14
15     protected:
16         // Propriedades relacionadas ao escoamento e cálculos
17         double fatorFricçãoPoco = 0.0;

```

```

18     double fatorFriccaoAnular = 0.0;
19     double reynoldsPoco = 0.0;
20     double reynoldsAnular = 0.0;
21     double vMediaPoco = 0.0;
22     double vMediaAnular = 0.0;
23
24     // Tipo de fluxo (laminar ou turbulento)
25     std::string fluxoPoco;
26     std::string fluxoAnular;
27
28     // Objeto que contem as propriedades do poco
29     CObjetoPoco* poco;
30
31 public:
32     // Construtores
33     CModeloReologico() {}
34     virtual ~CModeloReologico() {}
35     CModeloReologico(CObjetoPoco* poco) : poco(poco) {}
36
37     // Getters
38     double FatorFriccaoPoco() const { return fatorFriccaoPoco; }
39     double FatorFriccaoAnular() const { return fatorFriccaoAnular; }
40     double ReynoldsPoco() const { return reynoldsPoco; }
41     double ReynoldsAnular() const { return reynoldsAnular; }
42     double VMediaPoco() const { return vMediaPoco; }
43     double VMediaAnular() const { return vMediaAnular; }
44     std::string FluxoPoco() const { return fluxoPoco; }
45     std::string FluxoAnular() const { return fluxoAnular; }
46
47     // Metodos 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     // Metodos puros (devem ser implementados pelas classes filhas)
57     virtual std::string DeterminarFluxoPoco() = 0;
58     virtual std::string DeterminarFluxoAnular() = 0;
59     virtual double CalcularPerdaPorFriccaoPoco() = 0;
60     virtual double CalcularPerdaPorFriccaoAnular() = 0;
61 };
62
63 #endif // CMODELOREOLOGICO_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 11 a implementação da classe CModeloReologico.

Listing 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 n mero de Reynolds no poco usando a viscosidade total do fluido
8 double CModeloReologico::DeterminarReynoldsPoco() {
9     reynoldsPoco = (928 * poco->DensidadeEfetivaTotal() * vMediaPoco * poco->
10                  DiametroRevestimentoID()) /
11                  poco->ViscosidadeEfetivaTotal();
12 }
13
14 // Mesmo calculo, mas permitindo passar a viscosidade como parametro
15 double CModeloReologico::DeterminarReynoldsPoco(double viscosidade) {
16     reynoldsPoco = (928 * poco->DensidadeEfetivaTotal() * vMediaPoco * poco->
17                  DiametroRevestimentoID()) /
18                  viscosidade;
19 }
20 }
21
22 // Calcula o n mero de Reynolds no espaco anular
23 double CModeloReologico::DeterminarReynoldsAnular() {
24     reynoldsAnular = (757 * poco->DensidadeEfetivaTotal() * vMediaAnular *
25                         (poco->DiametroPoco() - poco->DiametroRevestimentoOD())) /
26                         poco->ViscosidadeEfetivaTotal();
27 }
28 }
29
30 // 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 }
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 }
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() /
47                     (2.448 * (std::pow(poco->DiametroPoco(), 2) - std::pow(poco->
48                         DiametroRevestimentoOD(), 2)));
49 }

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 12 o arquivo de cabeçalho da classe CModeloNewtoniano.frame

Listing 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 reologico newtoniano
8 Nesse caso, a viscosidade e constante e independe da taxa de deformacao
9 A classe herda de CModeloReologico e implementa os metodos 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 13 a implementação da classe CModeloNewtoniano.
frame

Listing 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 turbulento
10    fluxoPoco = (reynoldsPoco <= 2100) ? "Laminar" : "Turbulento";
11    return fluxoPoco;
12 }
13
14 // Determina o tipo de fluxo no espaco anular com base no numero de Reynolds
15 std::string CModeloNewtoniano::DeterminarFluxoAnular() {
16     DeterminarVelocidadeMediaAnular();
17     DeterminarReynoldsAnular();
18
19     fluxoAnular = (reynoldsAnular <= 2100) ? "Laminar" : "Turbulento";
20     return fluxoAnular;
21 }
22
23 // Calcula a perda de carga por friccao no poco para regime laminar ou turbulento
24 double CModeloNewtoniano::CalcularPerdaPorFriccaoPoco() {
25     if (fluxoPoco.empty()) {

```

```

26         DeterminarFluxoPoco();
27     }
28
29     if (fluxoPoco == "Laminar") {
30         return (poco->ViscosidadeEfetivaTotal() * vMediaPoco) /
31             (1500 * std::pow(poco->DiametroRevestimentoID(), 2));
32     } else {
33         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std::pow(vMediaPoco,
34             1.75) *
35             std::pow(poco->ViscosidadeEfetivaTotal(), 0.25)) /
36             (1800 * std::pow(poco->DiametroRevestimentoID(), 1.25));
37 }
38
39 // Calcula a perda de carga por friccao no espaco anular
40 double CModeloNewtoniano::CalcularPerdaPorFriccaoAnular() {
41     if (fluxoAnular.empty()) {
42         DeterminarFluxoAnular();
43     }
44
45     double diametroAnular = poco->DiametroPoco() - poco->DiametroRevestimentoOD();
46
47     if (fluxoAnular == "Laminar") {
48         return (poco->ViscosidadeEfetivaTotal() * vMediaAnular) /
49             (1000 * std::pow(diametroAnular, 2));
50     } else {
51         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std::pow(vMediaAnular,
52             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 14 o arquivo de cabeçalho da classe CModeloBingham.
frame

Listing 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 definido
9 A implementação baseia-se na herança da classe CModeloReologico
10 */
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     // Numeros de Hedstron (relacionados ao tipo de escoamento)
24     double reynoldsHedstronPoco = 0.0;
25     double reynoldsHedstronAnular = 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, double
37         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     double PontoDeEscoamento() const { return pontoDeEscoamento; }
49     double ReynoldsCriticoPoco() const { return reynoldsCriticoPoco; }
50     double ReynoldsCriticoAnular() const { return reynoldsCriticoAnular; }
51     double ReynoldsHedstronPoco() const { return reynoldsHedstronPoco; }
52     double ReynoldsHedstronAnular() const { return reynoldsHedstronAnular; }
53
54     // Setters
55     void ViscosidadePlastica(double valor) { viscosidadePlastica = valor; }
56     void PontoDeEscoamento(double valor) { pontoDeEscoamento = valor; }
57
58     // Metodos especificos do modelo de Bingham
59     double DeterminarReynoldsCritico(double hedstron);
60     double DeterminarReynoldsHedstronPoco();
61     double DeterminarReynoldsHedstronAnular();
62     std::string DeterminarFluxoPoco() override;
63     std::string DeterminarFluxoAnular() override;
64     double CalcularPerdaPorFriccaoPoco() override;
65     double CalcularPerdaPorFriccaoAnular() override;
66 };
67 #endif // CMODELOBINGHAM_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 15 a implementação da classe CModeloBingham.

Listing 15: Arquivo de implementação da classe CModeloBingham

```

1 #include "CModeloBingham.h"
2 #include <cmath>
3 #include <QApplication>
4 #include <QFileDialog>

```

```

5 #include <QString>
6 #include <QMessageBox>
7
8 // Determina o valor de Reynolds critico com base no numero de Hedstron
9 // Utiliza metodo numerico de Newton-Raphson para resolver a equacao implicita
10 double CModeloBingham::DeterminarReynoldsCritico(double hedstron) {
11     // f(x) = ((x / (1 - x)^3) * 16800) - Hedstron
12     auto func = [hedstron](double x) {
13         return ((x / std::pow(1 - x, 3)) * 16800) - hedstron;
14     };
15
16     // Derivada de f(x)
17     auto derivadaFunc = [] (double x) {
18         return (16800 * (1 + x) / std::pow(1 - x, 4));
19     };
20
21     // Metodo de Newton-Raphson para aproximar a raiz
22     auto newtonRaphson = [&] (double x) {
23         for (int i = 0; i < 10000; ++i) {
24             x = x - func(x) / derivadaFunc(x);
25             if (fabs(func(x)) < 1e-2) break;
26         }
27         return x;
28     };
29
30     // Chute inicial
31     double y = newtonRaphson(0.99);
32
33     // Retorna o Reynolds critico com base em y encontrado
34     return ((1 - ((4.0 / 3.0) * y) + ((1.0 / 3.0) * std::pow(y, 4))) * hedstron) / (8 * y
35 );
36
37 // Calcula o numero de Hedstron para o escoamento no poto
38 double CModeloBingham::DeterminarReynoldsHedstronPoco() {
39     reynoldsHedstronPoco =
40         (37100 * poco->DensidadeEfetivaTotal() * pontoDeEscoamento *
41         std::pow(poco->DiametroRevestimentoID(), 2)) /
42         std::pow(viscosidadePlastica, 2);
43
44     return reynoldsHedstronPoco;
45 }
46
47 // Calcula o numero de Hedstron para o escoamento no espaco anular
48 double CModeloBingham::DeterminarReynoldsHedstronAnular() {
49     double diametroAnular = poco->DiametroPoco() - poco->DiametroRevestimentoOD();
50
51     reynoldsHedstronAnular =
52         (24700 * poco->DensidadeEfetivaTotal() * pontoDeEscoamento *
53         std::pow(diametroAnular, 2)) /
54         std::pow(viscosidadePlastica, 2);
55
56     return reynoldsHedstronAnular;
57 }
58
59 // Determina o tipo de fluxo no poto (laminar ou turbulento)
60 std::string CModeloBingham::DeterminarFluxoPoco() {
61     DeterminarVelocidadeMediaPoco();
62     DeterminarReynoldsPoco(viscosidadePlastica);
63     DeterminarReynoldsHedstronPoco();

```

```

64
65     reynoldsCriticoPoco = DeterminarReynoldsCritico(reynoldsHedstronPoco);
66
67     fluxoPoco = (reynoldsPoco <= reynoldsCriticoPoco) ? "Laminar" : "Turbulento";
68     return fluxoPoco;
69 }
70
71 // Determina o tipo de fluxo no espaço anular
72 std::string CModeloBingham::DeterminarFluxoAnular() {
73     DeterminarVelocidadeMediaAnular();
74     DeterminarReynoldsAnular(viscosidadePlastica);
75     DeterminarReynoldsHedstronAnular();
76
77     reynoldsCriticoAnular = DeterminarReynoldsCritico(reynoldsHedstronAnular);
78
79     fluxoAnular = (reynoldsAnular <= reynoldsCriticoAnular) ? "Laminar" : "Turbulento";
80     return fluxoAnular;
81 }
82
83 // Calcula a perda de carga por fricção no pôco
84 double CModeloBingham::CalcularPerdaPorFriccaoPoco() {
85     if (fluxoPoco == "Laminar") {
86         return ((viscosidadePlastica * vMediaPoco) /
87                 (1500 * std::pow(poco->DiametroRevestimentoID(), 2))) +
88                 (pontoDeEscoamento / (225 * poco->DiametroRevestimentoID()));
89     } else {
90         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std::pow(vMediaPoco,
91             1.75) *
92                 std::pow(viscosidadePlastica, 0.25)) /
93                 (1800 * std::pow(poco->DiametroRevestimentoID(), 1.25));
94     }
95 }
96
97 // Calcula a perda de carga por fricção no espaço anular
98 double CModeloBingham::CalcularPerdaPorFriccaoAnular() {
99     double diametroAnular = poco->DiametroPoco() - poco->DiametroRevestimentoOD();
100
101     if (fluxoAnular == "Laminar") {
102         return ((viscosidadePlastica * vMediaAnular) /
103                 (1000 * std::pow(diametroAnular, 2))) +
104                 (pontoDeEscoamento / (200 * diametroAnular));
105     } else {
106         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std::pow(vMediaAnular,
107             1.75) *
108                 std::pow(viscosidadePlastica, 0.25)) /
109                 (1396 * std::pow(diametroAnular, 1.25));
110     }
111 }
```

Fonte: produzido pelo autor.

Apresenta-se na listagem 16 o arquivo de cabeçalho da classe CModeloPotencia.frame

Listing 16: Arquivo de implementação da classe CModeloPotencia

```

1 #ifndef CMODELOPOTENCIA_H
2 #define CMODELOPOTENCIA_H
3
4 #include "CModeloReológico.h"
```

```

5
6 /*
7 Classe que representa o modelo reologico da Lei da Potencia
8 Esse modelo e aplicado a fluidos pseudoplastico ou dilatante (n diferente de 1)
9 A classe herda de CModeloReologico e implementa os metodos especificos do modelo
10 */
11
12 class CModeloPotencia : public CModeloReologico {
13
14 protected:
15     // Parametros do modelo da potencia
16     double indiceDeConsistencia;        // K
17     double indiceDeComportamento;       // n
18
19     // Reynolds critico arbitrario (2100 como base de separacao)
20     double reynoldsCriticoPoco;
21     double reynoldsCriticoAnular;
22
23 public:
24     // Construtores
25     CModeloPotencia() {}
26     ~CModeloPotencia() {}
27
28     // Construtor com inicializacao do poco e do indice de consistencia
29     CModeloPotencia(CObjetoPoco* poco, double indiceDeConsistencia, double
            indiceDeComportamento)
30         : CModeloReologico(poco),
31             indiceDeConsistencia(indiceDeConsistencia),
32             indiceDeComportamento(indiceDeComportamento)
33
34     {
35         DeterminarFluxoPoco();
36         DeterminarFluxoAnular();
37         IndiceDeComportamento(indiceDeComportamento);
38     }
39
40     // Getters
41     double ReynoldsCriticoPoco() const { return reynoldsCriticoPoco; }
42     double ReynoldsCriticoAnular() const { return reynoldsCriticoAnular; }
43     double IndiceDeConsistencia() const { return indiceDeConsistencia; }
44     double IndiceDeComportamento() const { return indiceDeComportamento; }
45     std::string FluxoPoco() const { return fluxoPoco; }
46     std::string FluxoAnular() const { return fluxoAnular; }
47
48     // Setters
49     void IndiceDeConsistencia(double valor) { indiceDeConsistencia = valor; }
50     void IndiceDeComportamento(double valor) { indiceDeComportamento = valor; }
51     void FluxoPoco(const std::string& fluxo) { fluxoPoco = fluxo; }
52     void FluxoAnular(const std::string& fluxo) { fluxoAnular = fluxo; }
53
54     // Metodos especificos do modelo de potencia
55     double DeterminarFatorFriccao(double reynolds, double n);
56     double DeterminarReynoldsCritico(double reynolds);
57     double DeterminarReynoldsPoco();
58     double DeterminarReynoldsAnular();
59     std::string DeterminarFluxoPoco() override;
60     std::string DeterminarFluxoAnular() override;
61     double CalcularPerdaPorFriccaoPoco() override;
62     double CalcularPerdaPorFriccaoAnular() override;
63 };

```

```

64
65 #endif // CMODELOPOTENCIA_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 17 a implementação da classe CModeloPotencia.
frame

Listing 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 base em NRe e n
6 double CModeloPotencia::DeterminarFatorFriccao(double NRe, double n) {
7
8     // Coeficientes obtidos via regress o log-log com base nos dados reais do gr fico
9     const double n_vals[] = {1.0, 0.8, 0.6, 0.4, 0.2};
10    const double a_vals[] = {-0.23285, -0.25049, -0.24695, -0.20192, -0.30519};
11    const double b_vals[] = {-1.14079, -1.12562, -1.25944, -1.66278, -1.33815};
12
13    // Interpola o linear entre os dois pontos mais pr ximos
14    double a = 0.0, b = 0.0;
15    for (int i = 1; i < 5; ++i) {
16        if (n <= n_vals[i - 1] && n >= n_vals[i]) {
17            double t = (n - n_vals[i]) / (n_vals[i - 1] - n_vals[i]);
18            a = a_vals[i] + t * (a_vals[i - 1] - a_vals[i]);
19            b = b_vals[i] + t * (b_vals[i - 1] - b_vals[i]);
20            break;
21        }
22    }
23
24    double logF = a * std::log10(NRe) + b;
25    return std::pow(10.0, logF);
26}
27
28// Funcao generica para Reynolds critico (nao esta sendo usada diretamente aqui)
29double CModeloPotencia::DeterminarReynoldsCritico(double Reynolds) {
30    return 183.13 * std::pow(Reynolds, 0.3185);
31}
32
33// Calcula o numero de Reynolds para o escoamento no poco
34double CModeloPotencia::DeterminarReynoldsPoco() {
35    reynoldsPoco =
36        ((89100 * poco->DensidadeEfetivaTotal() * std::pow(vMediaPoco, 2 -
37            indiceDeComportamento)) / indiceDeConsistencia) *
38        (std::pow((0.0416 * poco->DiametroRevestimentoID()) / (3 + (1 /
39            indiceDeComportamento)), indiceDeComportamento));
40
41    reynoldsCriticoPoco = DeterminarReynoldsCritico(reynoldsPoco);
42
43}
44
45// Calcula o numero de Reynolds no espaco anular
46double CModeloPotencia::DeterminarReynoldsAnular() {
47    reynoldsAnular =

```

```

48     ((109000 * poco->DensidadeEfetivaTotal() * std::pow(vMediaAnular, 2 -
        indiceDeComportamento)) / indiceDeConsistencia) *
49     (std::pow((0.0208 * (poco->DiametroPoco() - poco->DiametroRevestimentoOD())) / (2
        + (1 / indiceDeComportamento)), indiceDeComportamento));
50
51     reynoldsCriticoAnular = DeterminarReynoldsCritico(reynoldsAnular);
52
53     return reynoldsAnular;
54 }
55
56 // Determina o tipo de fluxo no poco com base no valor de Reynolds calculado
57 std::string CModeloPotencia::DeterminarFluxoPoco() {
58     vMediaPoco = DeterminarVelocidadeMediaPoco();
59     reynoldsPoco = DeterminarReynoldsPoco();
60     fluxoPoco = (reynoldsPoco <= reynoldsCriticoPoco) ? "Laminar" : "Turbulento";
61     return fluxoPoco;
62 }
63
64 // Determina o tipo de fluxo no espaço anular
65 std::string CModeloPotencia::DeterminarFluxoAnular() {
66     vMediaAnular = DeterminarVelocidadeMediaAnular();
67     reynoldsAnular = DeterminarReynoldsAnular();
68     fluxoAnular = (reynoldsAnular <= reynoldsCriticoAnular) ? "Laminar" : "Turbulento";
69     return fluxoAnular;
70 }
71
72 // Calcula a perda de carga por fricção no poco, separando para fluxo laminar e
    turbulentoo
73 double CModeloPotencia::CalcularPerdaPorFriccaoPoco() {
74     fatorFriccaoPoco = DeterminarFatorFriccao(reynoldsPoco, indiceDeComportamento);
75
76     if (fluxoPoco == "Laminar") {
77         return ((indiceDeConsistencia * std::pow(vMediaPoco, -indiceDeComportamento)) *
78                 std::pow(( (3 + (1 / indiceDeComportamento)) / 0.0416),
79                           indiceDeComportamento)) /
80                 (144000 * std::pow(poco->DiametroRevestimentoID(), 1 +
81                           indiceDeComportamento));
82     } else {
83         return (fatorFriccaoPoco * poco->DensidadeEfetivaTotal() * std::pow(vMediaPoco,
84             2)) /
85                 (25.8 * poco->DiametroRevestimentoID());
86     }
87 }
88
89 // Calcula a perda de carga por fricção no espaço anular
90 double CModeloPotencia::CalcularPerdaPorFriccaoAnular() {
91     double diametroAnular = poco->DiametroPoco() - poco->DiametroRevestimentoOD();
92     fatorFriccaoAnular = DeterminarFatorFriccao(reynoldsAnular, indiceDeComportamento);
93
94     if (fluxoAnular == "Laminar") {
95         return ((indiceDeConsistencia * std::pow(vMediaAnular, -indiceDeComportamento)) *
96                 std::pow(((2 + (1 / indiceDeComportamento)) / 0.0208),
97                           indiceDeComportamento)) /
98                 (144000 * std::pow(diametroAnular, 1 + indiceDeComportamento));
99     } else {
100         return (fatorFriccaoAnular * poco->DensidadeEfetivaTotal() * std::pow(
101             vMediaAnular, 2)) /
102                 (21.1 * diametroAnular);
103     }
104 }
```

Fonte: produzido pelo autor.

0.1 Código Qt (interface)

Apresenta-se na listagem 18 o arquivo de cabeçalho da classe CJanelaAdicionarFluido.

frame

Listing 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 editar um fluido
8 Armazena os dados digitados: nome, densidade, viscosidade e faixa de profundidade
9 */
10
11 namespace Ui {
12 class CJanelaAdicionarFluido;
13 }
14
15 class CJanelaAdicionarFluido : public QDialog
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) { profundidadeInicial = valor; }
28     void ProfundidadeFinal(const QString& valor) { profundidadeFinal = valor; }
29     void ModoEdicao(bool opcao) { modoEdicao = opcao; }
30
31     // metodos para acessar os dados
32     QString NomeFluido() const { return nomeFluido; }
33     QString Densidade() const { return densidade; }
34     QString Viscosidade() const { return viscosidade; }
35     QString ProfundidadeInicial() const { return profundidadeInicial; }
36     QString ProfundidadeFinal() const { return profundidadeFinal; }
37     bool ModoEdicao() const { return modoEdicao; }
38
39 private slots:
40     void on_btnReturn_accepted(); // confirma os dados
41     void on_btnReturn_rejected(); // cancela a edicao
42
43 private:
44     bool modoEdicao = true;
45     Ui::CJanelaAdicionarFluido *ui = nullptr;
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 19 a implementação da classe CJanelaAdicionarFluido. frame

Listing 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         ProfundidadeFinal(ui->LnValorProfundidadeFinal->text());
29
30         // Verifica se algum campo ficou vazio
31         if (ui->LnValorNome->text().isEmpty() ||
32             ui->LnValorDensidade->text().isEmpty() ||
33             ui->LnValorViscosidade->text().isEmpty() ||
34             ui->LnValorProfundidadeInicial->text().isEmpty() ||
35             ui->LnValorProfundidadeFinal->text().isEmpty()) {
36             QMessageBox::warning(this, "Erro", "Por favor, preencha todos os campos!");
37         }
38
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 todos os campos!");
52     }
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 20 o arquivo de cabeçalho da classe CJanelaAdicionarTrechoTubulacao.

frame

Listing 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 tubulacao ao sistema
80 usuario insere informacoes da geometria da tubulacao, propriedades fisicas e dados do
     fluido
9 Esses dados sao usados para simulacoes de comportamento termico e mecanico da tubulacao
10 */
11
12 namespace Ui {
13 class CJanelaAdicionarTrechoTubulacao;
14 }
15
16 class CJanelaAdicionarTrechoTubulacao : public QDialog
17 {
18     Q_OBJECT
19
20 public:
21     explicit CJanelaAdicionarTrechoTubulacao(QWidget *parent = nullptr);
22     ~CJanelaAdicionarTrechoTubulacao();
23
24     // metodos para acessar os dados inseridos
25     QString Trecho() const { return trecho; }
26     QString ProfundidadeInicial() const { return profundidadeInicial; }
27     QString ProfundidadeFinal() const { return profundidadeFinal; }
28     QString DiametroExterno() const { return diametroExterno; }
29     QString DiametroInterno() const { return diametroInterno; }
30     QString CoeficientePoisson() const { return coeficientePoisson; }
31     QString CoeficienteExpansaoTermica() const { return coeficienteExpansaoTermica; }
32     QString ModuloElasticidade() const { return moduloElasticidade; }
33     QString PesoUnidade() const { return pesoUnidade; }
34     QString NomeFluido() const { return nomeFluido; }
35     QString Densidade() const { return densidade; }
36     QString Viscosidade() const { return viscosidade; }
37

```

```

38     // metodos para definir os dados
39     void Trecho(const QString& valor) { trecho = valor; }
40     void ProfundidadeInicial(const QString& valor) { profundidadeInicial = valor; }
41     void ProfundidadeFinal(const QString& valor) { profundidadeFinal = valor; }
42     void DiametroExterno(const QString& valor) { diametroExterno = valor; }
43     void DiametroInterno(const QString& valor) { diametroInterno = valor; }
44     void CoeficientePoisson(const QString& valor) { coeficientePoisson = valor; }
45     void CoeficienteExpansaoTermica(const QString& valor) { coeficienteExpansaoTermica =
46         valor; }
47     void ModuloElasticidade(const QString& valor) { moduloElasticidade = valor; }
48     void PesoUnidade(const QString& valor) { pesoUnidade = valor; }
49     void NomeFluido(const QString& valor) { nomeFluido = valor; }
50     void Densidade(const QString& valor) { densidade = valor; }
51     void Viscosidade(const QString& valor) { viscosidade = valor; }
52     void ModoEdicao(bool opcao) { edit = opcao; }

53 private slots:
54     void on_btnReturn_accepted(); //acao ao confirmar
55     void on_btnReturn_rejected(); //acao ao cancelar
56
57 private:
58     bool edit = false; // indica se esta no modo de edicao
59     Ui::CJanelaAdicionarTrechoTubulacao *ui = nullptr;
60
61     // dados da tubulacao
62     QString trecho;
63     QString profundidadeInicial;
64     QString profundidadeFinal;
65     QString diametroExterno;
66     QString diametroInterno;
67     QString coeficientePoisson;
68     QString coeficienteExpansaoTermica;
69     QString moduloElasticidade;
70     QString pesoUnidade;

71     // dados do fluido no trecho
72     QString nomeFluido;
73     QString densidade;
74     QString viscosidade;
75
76 };
77
78 #endif // CJANELAADICIONARTRECHOTUBULACAO_H

```

Fonte: produzido pelo autor.

Apresenta-se na listagem 21 a implementação da classe CJanelaAdicionarTrechoTubulacao.

frame

Listing 21: Arquivo de implementação da classe CJanelaAdicionarTrechoTubulacao

```

1 #include "CJanelaAdicionarTrechoTubulacao.h"
2 #include "ui_CJanelaAdicionarTrechoTubulacao.h"
3 #include <QMessageBox>
4
5 // Construtor: inicializa a interface
6 CJanelaAdicionarTrechoTubulacao::CJanelaAdicionarTrechoTubulacao(QWidget *parent)
7     : QDialog(parent)
8     , ui(new Ui::CJanelaAdicionarTrechoTubulacao)
9 {

```

```

10     ui->setupUi(this);
11 }
12
13 // Destruitor: libera memoria da interface
14 CJanelaAdicionarTrechoTubulacao::~CJanelaAdicionarTrechoTubulacao()
15 {
16     delete ui;
17 }
18
19 // Acao ao clicar em OK
20 void CJanelaAdicionarTrechoTubulacao::on_btnReturn_accepted()
21 {
22     // verifica se campos obrigatorios estao vazios
23     bool camposVazios =
24         ui->editTrecho->text().isEmpty() ||
25         ui->editProfInicial->text().isEmpty() ||
26         ui->editProfFinal->text().isEmpty() ||
27         ui->editDiametroExterno->text().isEmpty() ||
28         ui->editDiametroInterno->text().isEmpty() ||
29         ui->editCoefPoisson->text().isEmpty() ||
30         ui->editCoefExpansao->text().isEmpty() ||
31         ui->editModuloElasticidade->text().isEmpty() ||
32         ui->editPesoUnid->text().isEmpty() ||
33         ui->editNome->text().isEmpty() ||
34         ui->editDensidade->text().isEmpty() ||
35         ui->editViscosidade->text().isEmpty();
36
37     if (edit && camposVazios) {
38         QMessageBox::warning(this, "Erro", "Por 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 22 o arquivo de cabeçalho da classe CJanelaGraficoPressaoHidroestatica.

frame

Listing 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 hidrostatica x profundidade
9 Recebe os dados (profundidade e pressao) e plota o grafico usando QCustomPlot
10 Permite exportar o grafico em imagem
11 */
12
13 namespace Ui {
14 class CJanelaGraficoPressaoHidroestatica;
15 }
16
17 class CJanelaGraficoPressaoHidroestatica : public QDialog
18 {
19     Q_OBJECT
20
21 public:
22     explicit CJanelaGraficoPressaoHidroestatica(
23         const std::pair<std::vector<double>, std::vector<double>>& dados,
24         QWidget *parent = nullptr);
25     ~CJanelaGraficoPressaoHidroestatica();
26
27     // metodo para atualizar os dados do grafico
28     void PerfilHidrostatico(const std::pair<std::vector<double>, std::vector<double>>&
29                             novoPerfil);
30
31     // acao ao clicar no botao de exportar imagem
32     void on_BtnExportarGrafico_clicked();
33
34 private:
35     Ui::CJanelaGraficoPressaoHidroestatica *ui;
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 23 a implementação da classe CJanelaGraficoPressaoHidroestatica.

frame

Listing 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 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::~CJanelaGraficoPressaoHidroestatica()
22 {
23     delete ui;
24 }
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 pressao
36 // Parte da logica para segmentacao por inclinacao foi adaptada com auxilio de IA (
37 // ChatGPT)
38 void CJanelaGraficoPressaoHidroestatica::PlotarGraficoPressaoxProfundidade()
39 {
40     QVector<double> x(profundidades.begin(), profundidades.end());
41     QVector<double> y(pressoes.begin(), pressoes.end());
42
43     auto *plot = ui->customPlotPressaoMediaProfundidade;
44     plot->clearGraphs();
45
46     QVector<double> trechoX, trechoY;
47
48     // Funcao lambda para comparar inclinacao entre segmentos
49     auto isMesmaInclinacao = [] (double dy1, double dy2, double tolerancia) {
50         return std::abs(dy1 - dy2) < tolerancia;
51     };
52
53     double tolerancia = 1e-2;
54     double inclinacaoAnterior = (y[1] - y[0]) / (x[1] - x[0]);
55
56     trechoX.push_back(x[0]);
57     trechoY.push_back(y[0]);
58
59     for (int i = 1; i < x.size(); ++i) {
60         double inclinacaoAtual = (y[i] - y[i - 1]) / (x[i] - x[i - 1]);
61
62         // Se mudou muito a inclinacao, cria um novo trecho no grafico

```

```

62         if (!isMesmaInclinacao(inclinacaoAtual, inclinacaoAnterior, tolerancia)) {
63             int index = plot->graphCount();
64             plot->addGraph();
65             plot->graph(index)->setData(trechoX, trechoY);
66
67             // Uso de cor e estilo dinamico inspirado em sugestao de IA para melhorar
68             // visualizacao
69             QPen pen;
70             pen.setWidth(1);
71             pen.setColor(QColor::fromHsv((index * 70) % 360, 255, 200));
72             plot->graph(index)->setPen(pen);
73
74             QCPSscatterStyle estilo(QCPSscatterStyle::ssCircle, 4);
75             plot->graph(index)->setScatterStyle(estilo);
76             plot->graph(index)->setLineStyle(QCPGraph::lsLine);
77
78             plot->graph(index)->setName(QString("Fluido %1").arg(index + 1));
79
80             trechoX.clear();
81             trechoY.clear();
82         }
83
84         trechoX.push_back(x[i]);
85         trechoY.push_back(y[i]);
86         inclinacaoAnterior = inclinacaoAtual;
87     }
88
89     // adiciona ultimo trecho
90     int index = plot->graphCount();
91     plot->addGraph();
92     plot->graph(index)->setData(trechoX, trechoY);
93
94     QPen pen;
95     pen.setWidth(1);
96     pen.setColor(QColor::fromHsv((index * 70) % 360, 255, 200));
97     plot->graph(index)->setPen(pen);
98
99     QCPSscatterStyle estilo(QCPSscatterStyle::ssCircle, 4);
100    plot->graph(index)->setScatterStyle(estilo);
101    plot->graph(index)->setLineStyle(QCPGraph::lsLine);
102    plot->graph(index)->setName(QString("Fluido %1").arg(index + 1));
103
104    // Configura rotulos dos eixos
105    plot->xAxis->setLabel("Profundidade (ft)");
106    plot->yAxis->setLabel("Pressao Hidrostatica (psi)");
107
108    // Define o intervalo dos eixos com base nos dados
109    plot->xAxis->setRange(*std::min_element(x.begin(), x.end()),
110                           *std::max_element(x.begin(), x.end()));
111    plot->yAxis->setRange(*std::min_element(y.begin(), y.end()),
112                           *std::max_element(y.begin(), y.end()));
113
114    // Legenda posicionada no canto inferior direito
115    plot->legend->setVisible(true);
116    plot->axisRect()->setLayout()->setInsetAlignment(0, Qt::AlignRight | Qt::AlignBottom);
117    plot->legend->setBrush(QBrush(Qt::white));
118    plot->legend->setBorderPen(QPen(Qt::gray));
119    plot->legend->setFont(QFont("Arial", 9));

```

```

120     // Ativacao do zoom, arrastar e selecao com mouse (configuracao sugerida com base em
121     // IA)
122     plot->setInteraction(QCP::iRangeDrag, true);
123     plot->setInteraction(QCP::iRangeZoom, true);
124     plot->setInteraction(QCP::iSelectPlottables, true);
125
126     // Mostra tooltip com coordenadas ao passar o mouse (recurso inserido com apoio de IA
127     // )
128     //ERRO: linha abaixo n o compila
129     // connect(plot, &QCustomPlot::mouseMove, this, [=](QMouseEvent *event) {
130     QObject::connect(plot, &QCustomPlot::mouseMove, this, [=](QMouseEvent *event) { //-
131         //--- AQUI EST A CORRE O PRINCIPAL
132         double xVal = plot->xAxis->pixelToCoord(event->pos().x());
133         double yVal = plot->yAxis->pixelToCoord(event->pos().y());
134
135         // NOTA: Conforme discutido anteriormente, setToolTip pode n o ser ideal para
136         // tooltip din mico no gr fico QCustomPlot. Avalie se voc realmente quer
137         // um tooltip no *widget* QCustomPlot ou uma QLabel na barra de status.
138         plot->setToolTip(QString("Profundidade: %1 \nPress o: %2"))
139             .arg(xVal, 0, 'f', 2)
140             .arg(yVal, 0, 'f', 2));
141     });
142
143 // Botao de exportar imagem do grafico (logica de exportacao por QPixmap)
144 void CJanelaGraficoPressaoHidroestatica::on_BtnExportarGrafico_clicked()
145 {
146     QString fileName = QFileDialog::getSaveFileName(this, "Salvar grafico", "", "PNG (*.png);JPEG (*.jpg)");
147
148     if (!fileName.isEmpty()) {
149         QPixmap imagem = ui->customPlotPressaoMediaProfundidade->toPixmap(800, 600);
150         imagem.save(fileName);
151     }
152 }
```

Fonte: produzido pelo autor.

Apresenta-se na listagem 24 o arquivo de cabeçalho da classe CJanelaMenu.
frame

Listing 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 25 a implementação da classe CJanelaMenu.

frame

Listing 25: Arquivo de implementação da classe CJanelaMenu

```

1 #include "CJanelaMenu.h"
2 #include "ui_CJanelaMenu.h"
3
4 #include "CSimuladorReologico.h"
5 #include "CSimuladorPerdaTubulacao.h"
6
7 // Construtor da janela principal do menu
8 JanelaMenu::JanelaMenu(QWidget *parent)
9     : QMainWindow(parent)
10    , ui(new Ui::JanelaMenu)
11 {
12     ui->setupUi(this);
13
14     // Caixa de texto apenas para visualizacao, sem interacao
15     ui->textEdit->setReadOnly(true);
16     ui->textEdit->setMouseTracking(false);
17     ui->textEdit->setFocusPolicy(Qt::NoFocus);
18
19     // Configuracao da descricao do Modulo 1
20     ui->lbnModulo01->setAlignment(Qt::AlignCenter);
21
22     // Texto com HTML para deixar a explicacao mais visual
23     QString buttonText_01 =
24         "<b>Modulo 01 - Hidrulica de Perfuração</b><br>"
25         "<ul>"
26         "<li>Transporte de Cascalho.</li>"
27         "<li>Fluxo não Newtoniano na Coluna e Anular.</li>"
28         "</ul>";
29
30     ui->lbnModulo01->setText(buttonText_01);
31     ui->btnModulo01->setText(""); // Botao fica invisivel para dar lugar ao label
32     ui->lbnModulo01->setAttribute(Qt::WA_TransparentForMouseEvents); // Label nao
33     atrapalha clique

```

```

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

```

Fonte: produzido pelo autor.