

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

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

TRABALHO DE CONCLUSÃO DE CURSO

Versão 1:

NATHAN RANGEL MAGALHÃES  
THAUAN FERREIRA BARBOSA;

Versão 2:

NATHAN RANGEL MAGALHÃES

Orientador: André Duarte Bueno

MACAÉ - RJ

Maio - 2025

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

NATHAN RANGEL MAGALHÃES

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

Orientador: André Duarte Bueno, D.Sc.

Macaé - RJ  
Maio - 2025

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

# Resumo

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

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

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

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

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

# **Lista de Figuras**

1.1	Etapas para o desenvolvimento do software - <i>projeto de engenharia</i> . . . . .	15
2.1	Diagrama de caso de uso – caso de uso geral . . . . .	20
2.2	Diagrama de caso de uso específico: cálculo de pressão hidrostática e densidade . . . . .	21
2.3	Diagrama de caso de uso específico - cálculo da perda de pressão devido a perda por fricção no poço e no anular . . . . .	22
3.1	Diagrama de corpo livre, atuação de forças em um elemento de fluido . . . . .	29
3.2	Coluna composta por fluidos com diferentes características . . . . .	31
3.3	Fluxo laminar de fluido Newtoniano . . . . .	33
3.4	Diagrama de pacotes . . . . .	41
4.1	Diagrama de classes . . . . .	44
4.2	Diagrama de sequência . . . . .	47
4.3	Diagrama de comunicação . . . . .	48
4.4	Diagrama de máquina de estado . . . . .	49
4.5	Diagrama de atividades . . . . .	49
5.1	Diagrama de componentes . . . . .	51
6.1	Versão 0.1, interface do software . . . . .	53
6.2	Versão 0.2, interface do software . . . . .	54
6.3	Versão 1.0, interface do software . . . . .	55
6.4	Versão 1.1, interface do software . . . . .	56
6.5	Versão 1.1, Menu de interface do software . . . . .	58
6.6	Versão 1.1, interface do modulo 01 do software . . . . .	58
6.7	Versão 1.1, interface do modulo 02 software . . . . .	59
8.1	Solução gerada para calculo da pressão hidroestática . . . . .	150
8.2	Soluções geradas pelo código para modelo Newtoniano no poço e anular . .	152
8.3	Soluções geradas pelo código para modelo de Bingham no poço e anular .	154
8.4	Soluções geradas pelo código para modelo de lei de Potência no poço e anular . . . . .	156

8.5	Solução gerada para variação do comprimento do tudo devido a força pistão	157
8.6	Solução gerada para variação do comprimento do tudo devido ao efeito balão	159
8.7	Solução gerada para variação do comprimento do tudo devido ao efeito da temperatura	160
8.8	Opção de Salvar	161
8.9	Modelo do Arquivo .dat	161
8.10	Opção de importação de arquivos no software	162
9.1	Logo e documentação do <i>software</i>	165
9.2	Código fonte da classe CSimuladorPoco, no <i>Doxxygen</i>	165
10.1	Versão 1.1, Menu de interface do software	166
10.2	Versão 1.1, interface do modulo 01 do software	167
10.3	Versão 1.1, interface do modulo 02 software	167
10.4	Acesso às funcionalidades de configuração do poço e dos fluidos	169
10.5	Exemplo de estrutura de arquivo .dat para importação de dados	170
10.6	Exemplo de estrutura de arquivo .dat para importação de dados	171



## Lista de Tabelas

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



# Sumário

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

<b>4 AOO – Análise Orientada a Objeto</b>	<b>43</b>
4.1 Diagramas de classes . . . . .	43
4.1.1 Dicionário de Classes . . . . .	45
4.2 Diagrama de Sequência – Eventos e Mensagens . . . . .	46
4.2.1 Diagrama de Sequência Geral . . . . .	46
4.3 Diagrama de Comunicação – Colaboração . . . . .	48
4.4 Diagrama de Máquina de Estado . . . . .	48
4.5 Diagrama de Atividades . . . . .	49
<b>5 Projeto</b>	<b>50</b>
5.1 Projeto do sistema . . . . .	50
5.2 Diagrama de componentes . . . . .	51
<b>6 Ciclos de planejamento/detalhamento</b>	<b>52</b>
6.1 Versão Inicial – Interação via Terminal e Geração de Gráficos com Gnuplot	52
6.2 Versão 0.2 – Otimização de Entrada, Validação e Armazenamento Automático de Dados . . . . .	53
6.3 Versão 1.0 – Interface Gráfica, Amigável e Intuitiva Utilizando o Framework Qt Creator . . . . .	54
6.4 Versão 1.1 – Consolidação Visual, Barra de Tarefas e Automação de Processos	55
6.5 Versão 2.0 – Navegação por Módulos e Simulação Mecânica de Variação de Comprimento ( $\Delta L$ ) . . . . .	57
<b>7 Ciclos Construção - Implementação</b>	<b>60</b>
7.1 Código-Fonte (modelo) . . . . .	60
7.1.1 Código Qt (interface) . . . . .	131
<b>8 Resultados</b>	<b>148</b>
8.1 Metodologia de Validação . . . . .	148
8.2 Casos de Teste . . . . .	149
8.2.1 Validação da pressão Hidroestática . . . . .	149
8.2.2 Validação da perda de fricção pelo modelo Newtoniano no Poço e Anular . . . . .	150
8.2.3 Validação da perda de fricção pelo modelo Bingham no Poço e Anular . . . . .	152
8.2.4 Validação da perda de fricção pelo modelo Potência no Poço e Anular . . . . .	154
8.2.5 Validação da variação do comprimento do tubo devido a força pistão . . . . .	156
8.2.6 Validação da variação do comprimento do tubo devido ao efeito balão . . . . .	158
8.2.7 Validação da variação do comprimento do tubo devido ao efeito da temperatura . . . . .	159
8.2.8 Validação da importação e exportação do documento (“salvar como...”) . . . . .	160

8.3 Conclusão . . . . .	162
<b>9 Documentação</b>	<b>164</b>
9.1 Documentação do usuário . . . . .	164
9.2 Documentação do desenvolvedor . . . . .	164
<b>10 Manual do usuário</b>	<b>166</b>
10.1 Instalação . . . . .	166
10.1.1 Dependências . . . . .	166
10.2 Interface gráfica . . . . .	166
10.3 Funcionalidades . . . . .	168
10.3.1 Módulo 1 – Hidráulica de Perfuração . . . . .	168
<b>Referências Bibliográficas</b>	<b>172</b>

# Capítulo 1

## Introdução

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

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

### 1.1 Escopo do problema

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

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

### 1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:

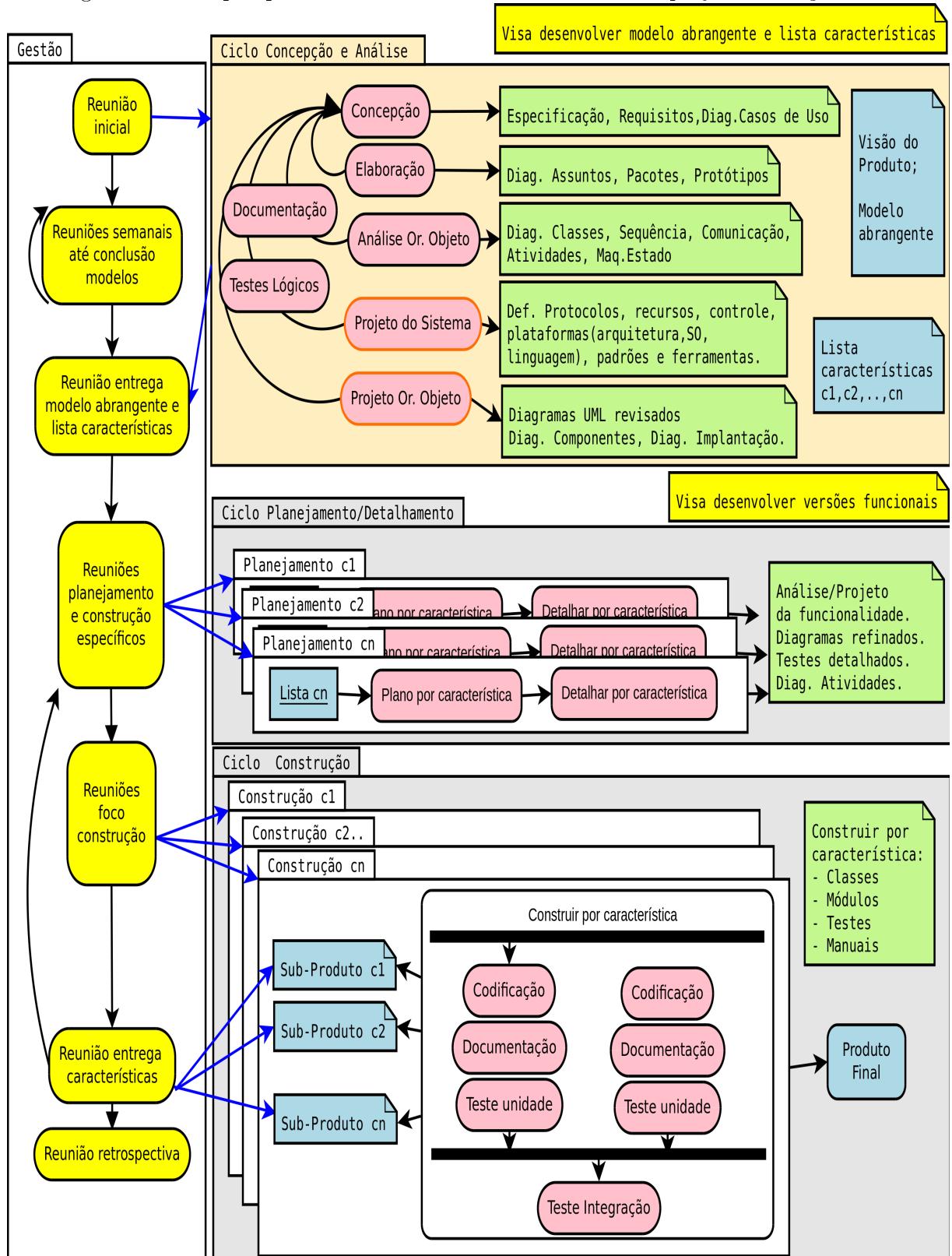
- Desenvolver um software capaz de analisar e calcular as principais equações que sustentam os fundamentos da disciplina de Engenharia de Poço, favorecendo a consolidação do conhecimento teórico adquirido em sala de aula.
- Objetivos específicos:
    - Modelar física e matematicamente os problemas abordados,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

### 1.3 Metodologia utilizada

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

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

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



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

# Capítulo 2

## Concepção

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

### 2.1 Nome do sistema/produto

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

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

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

A Tabela 2.1 apresenta as características do software desenvolvido.

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

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

## 2.2 Especificação

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

## 2.3 Requisitos

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

### 2.3.1 Requisitos Funcionais

Apresenta-se a seguir os requisitos funcionais

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

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

### 2.3.2 Requisitos Não Funcionais

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

## 2.4 Casos de Uso do Software

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

### **2.4.1 Diagrama de caso de uso geral**

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

**Tabela 2.2: Caso de uso 1**

Nome do caso de uso:	Simulação das propriedades de fluido e poço
Resumo/descrição:	Calcular as propriedades de fluido e poço para diferentes condições
Etapas:	<ol style="list-style-type: none"><li>1. Adicionar propriedades do fluido</li><li>2. Adicionar propriedades do poço</li><li>3. Incluir diferentes tipos de fluidos no poço</li><li>4. Calcular pressão hidrostática do poço</li><li>5. Calcular densidade do fluido</li><li>6. Calcular a queda de pressão devido a perdas por fricção</li><li>6. Plotar perfis de poço</li><li>7. Salvar dados em saída .dat</li></ol>

### **2.4.2 Diagrama de caso de uso específico**

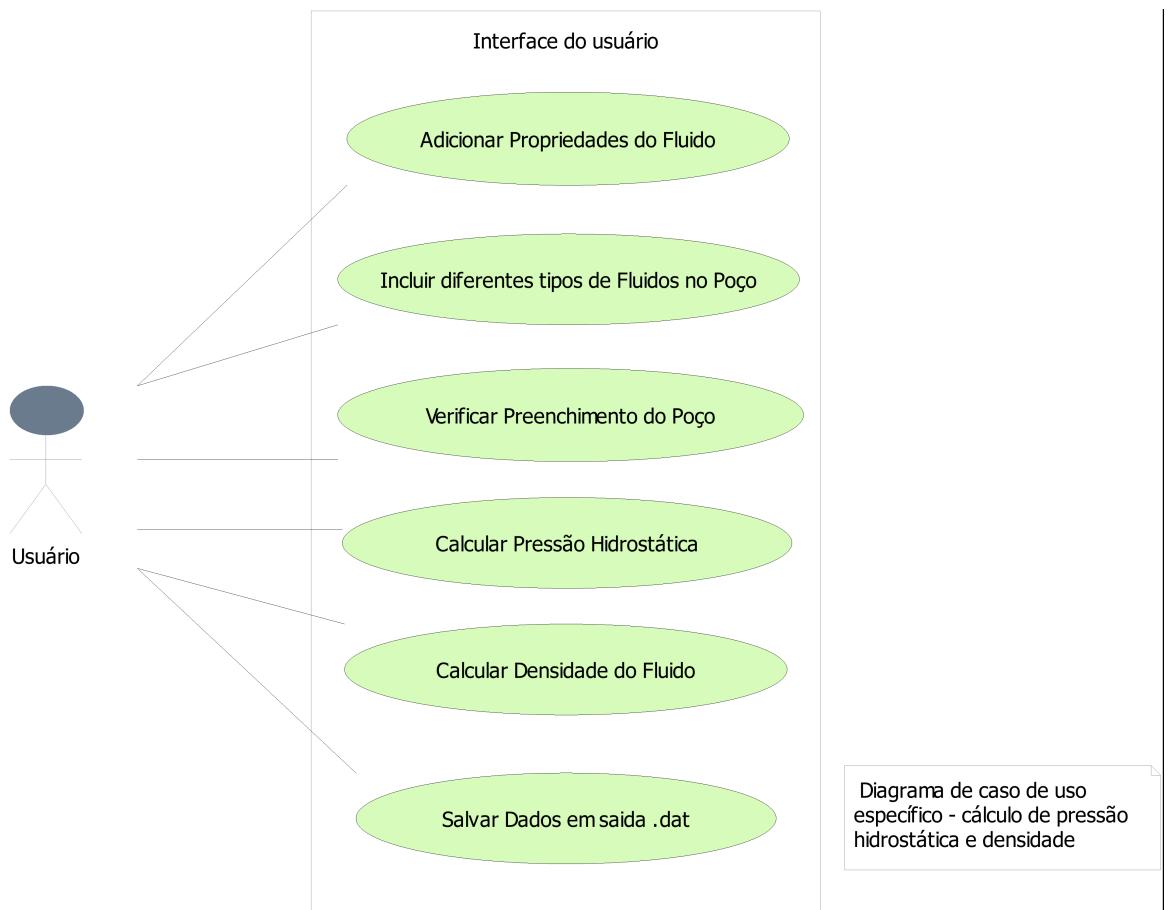
O caso de uso específico da Figura 2.2 mostra o cenário onde o usurário deseja calcular a pressão hidrostática e a densidade dos fluidos configurados no poço.

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



Fonte: Produzido pelo autor.

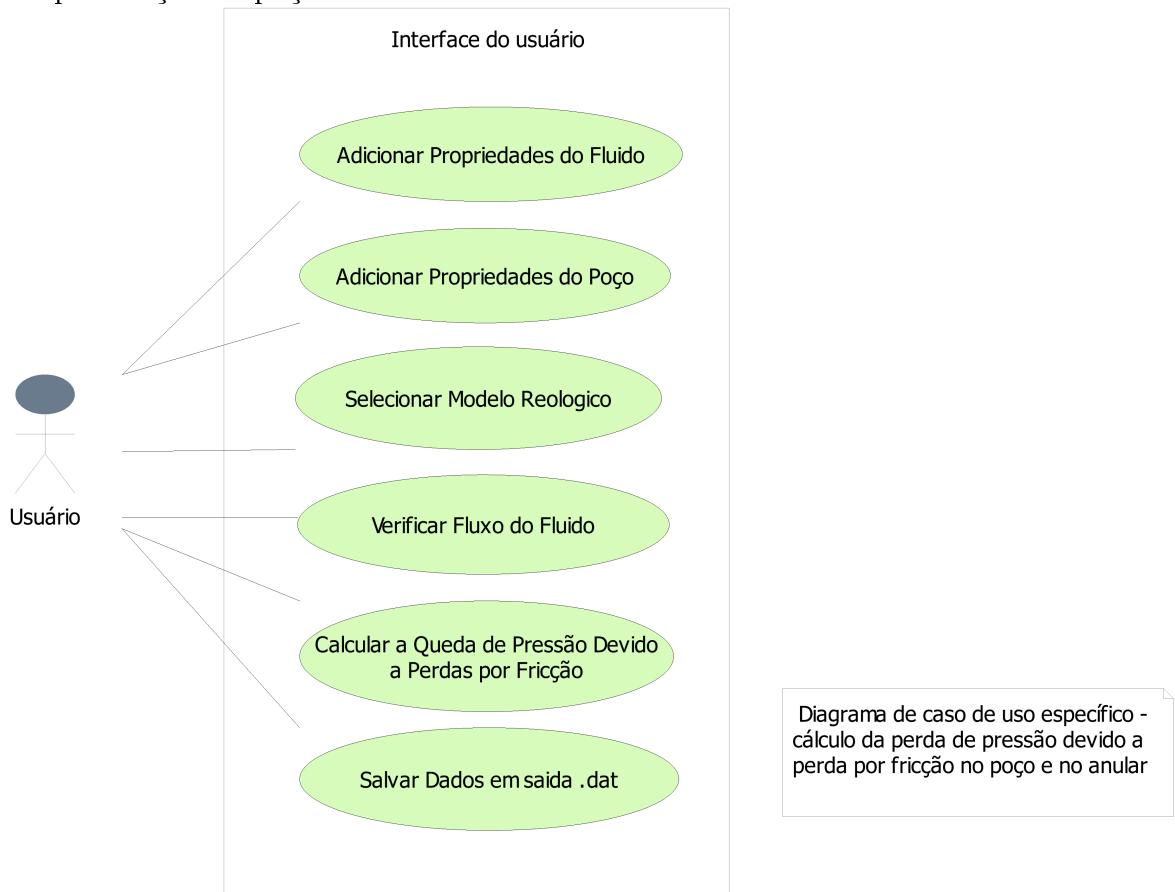
Figura 2.2: Diagrama de caso de uso específico: cálculo de pressão hidrostática e densidade



Fonte: Produzido pelo autor.

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

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



Fonte: Produzido pelo autor.

# Capítulo 3

## Elaboração

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

### 3.1 Análise de Domínio

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

Este projeto está vinculado a cinco conceitos essenciais:

1. Mecânica dos fluidos:

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

2. Mecânica das rochas:

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

### 3. Equações analíticas:

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

### 4. Programação:

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

### 5. Modelagem Gráfica:

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

## 3.2 Formulação Teórica

### 3.2.1 Ementa da disciplina

A seguir dados da ementa da disciplina cujo software atende.

- Dados básicos:

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

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

- Objetivos:

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

- Ementa resumida:

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

- Conteúdo programático:

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

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

### **3.2.2 Termos e Unidades**

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

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

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

### 3.2.3 Hidráulica de Perfuração

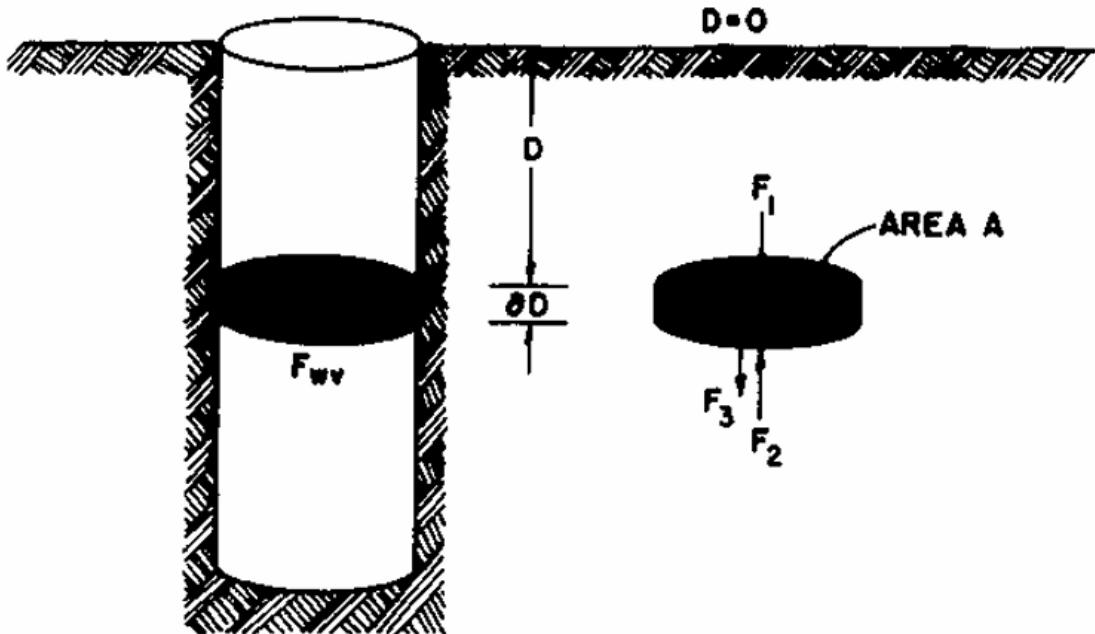
Na engenharia de perfuração, um fluido de perfuração possui três funções principais: transportar cascalho, prevenir o influxo de fluidos e manter a estabilidade do poço. Para cumprir essas funções, o fluido depende do seu escoamento na tubulação e das pressões associadas. Para que o engenheiro possa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

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

### 3.2.4 Pressão Hidrostática

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

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



Fonte Jr. et al. (1991)

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

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

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

### Fluidos incompressíveis

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

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

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

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

### Fluidos compressíveis

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

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

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

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

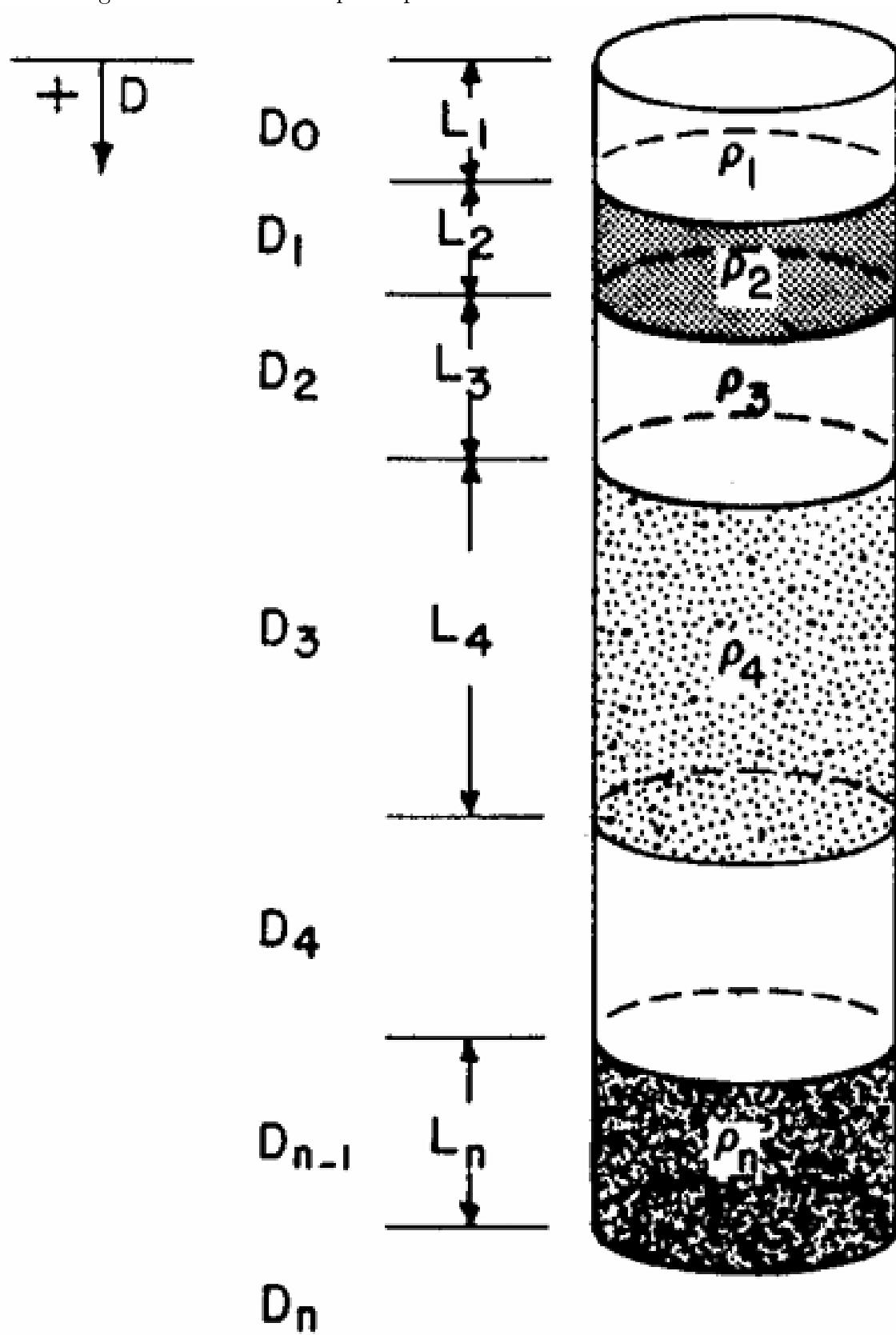
$$p = p_0 \exp \left( \frac{M \Delta Z}{1544 z T} \right) \quad (3.4)$$

Onde  $\Delta Z$  é a variação de profundidade [ $\text{ft}$ ].

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

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

Figura 3.2: Coluna composta por fluidos com diferentes características



Fonte Jr. et al. (1991)

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

ção:

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

Onde  $g$  é a gravidade [ $ft/s^2$ ].

### 3.2.6 Densidade Equivalente

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

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

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

### 3.2.7 Modelos Reológicos de Fluidos de Perfuração

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

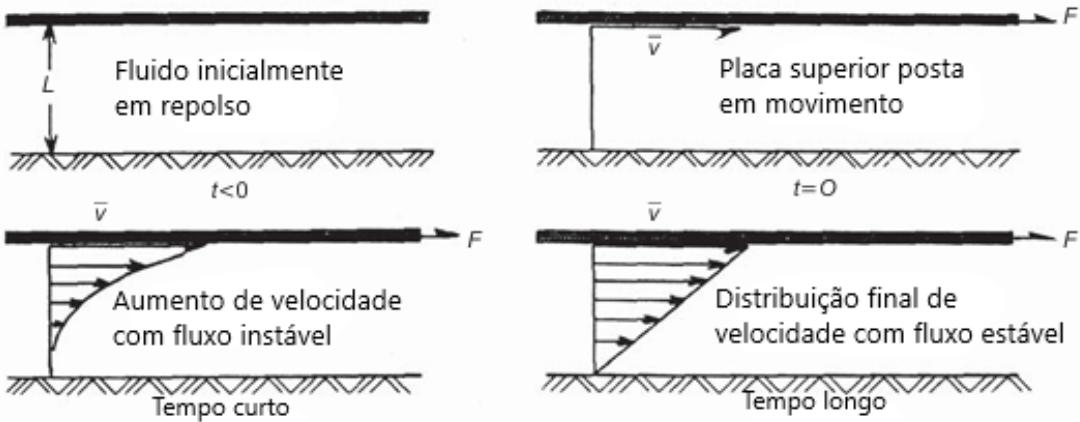
Na maioria dos casos, as propriedades elásticas dos fluidos de perfuração e seus efeitos durante o escoamento no poço são desprezíveis, sendo consideradas apenas as forças viscosas nos cálculos. Entretanto, com o avanço tecnológico, lamas cada vez mais complexas vêm sendo desenvolvidas, e, portanto,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

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

#### Visão geral dos modelos reológicos

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

Figura 3.3: Fluxo laminar de fluido Newtoniano



Adaptado de Mitchell & Miska (2011)

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

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

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

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

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

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

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

## Modelo de fluido Newtoniano

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

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

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

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

## Modelo de fluidos plásticos de Bingham

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

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

A principal característica de um plástico Bingham é a necessidade de um valor mínimo de tensão de cisalhamento para que o fluido comece a fluir, essa tensão mínima  $\tau_y$  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  $\mu_p$  é chamada de viscosidade plástica [ $cP$ ].

## Modelo fluidos de lei de potência

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

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

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

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

### 3.2.8 Perda de Pressão Friccional em um Tubo de Perfuração

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

#### Modelo de fluido Newtoniano

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

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

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

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

Onde  $q$  é a vazão do poço [ $gal/min$ ].

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

Para fluxo laminar:

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

Para o fluxo turbulento:

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

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

#### Fluidos plásticos de Bingham

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

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

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

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

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

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

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

Para fluxo laminar:

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

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

### Fluidos de lei de potência

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

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

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

Para fluxo laminar:

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

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

### 3.2.9 Perda de Pressão Friccional em um Anular

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

## Modelo de fluido Newtoniano

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

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

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

A velocidade média pode ser obtida pela equação:

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

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

Para fluxo laminar:

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

Para o fluxo turbulento:

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

## Fluidos plásticos de Bingham

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

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

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

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

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

Para fluxo laminar:

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

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

## Fluidos de lei de potência

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

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

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

Para fluxo laminar:

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

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

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

### 3.2.10 Variações de Carga Axial e Deslocamento em Colunas de Poço

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

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

#### Efeito da Variação de Temperatura

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

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

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

## Efeito Balão (Ballooning Effect)

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

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

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

## Variação Axial Devido à Força de Pistão ( $\Delta F$ )

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

A equação que representa esse deslocamento axial é:

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

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

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

## Força de Pistão Gerada por Packer

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

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

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

## Força de Pistão em Crossovers

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

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

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

### Efeitos de Injeção: Coluna Livre vs. Coluna Fixa

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

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

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

## 3.3 Identificação de Pacotes – Assuntos

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

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

### 3.4 Diagrama de Pacotes – Assuntos

O diagrama de pacotes é apresentado na Figura 3.4.

Figura 3.4: Diagrama de pacotes

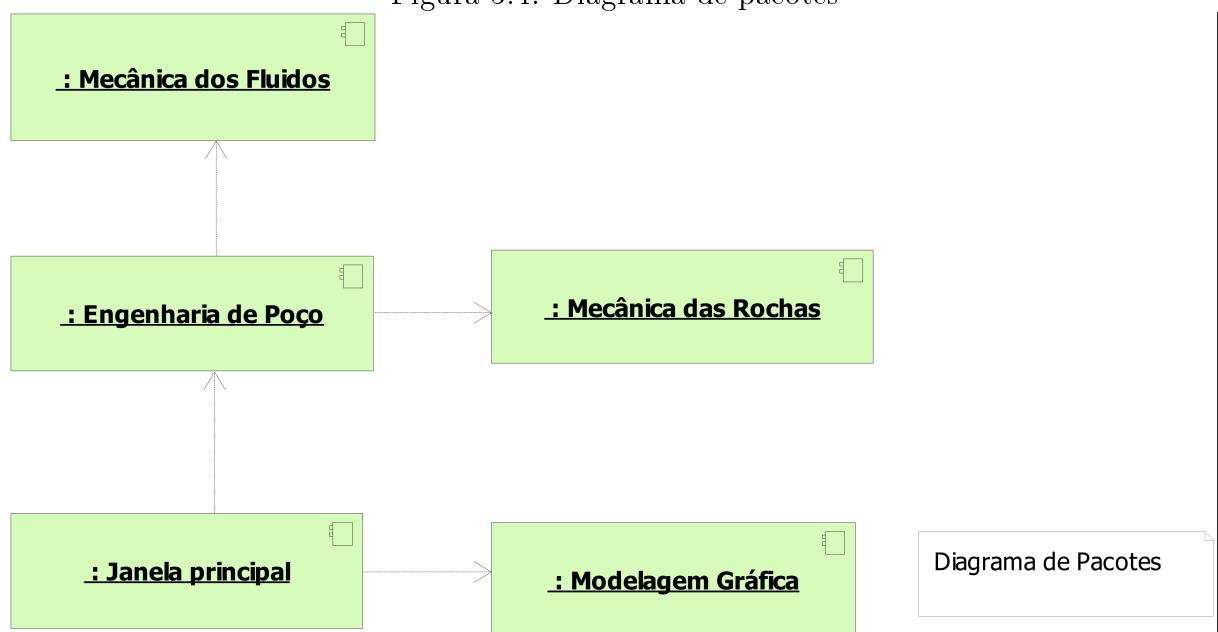


Diagrama de Pacotes

### 3 - Análise Orientada a Objeto

# Capítulo 4

## AOO – Análise Orientada a Objeto

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

### 4.1 Diagramas de classes

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

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

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

Figura 4.1: Diagrama de classes



Fonte: Produzido pelo autor

#### 4.1.1 Dicionário de Classes

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

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

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

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

## 4.2 Diagrama de Sequência – Eventos e Mensagens

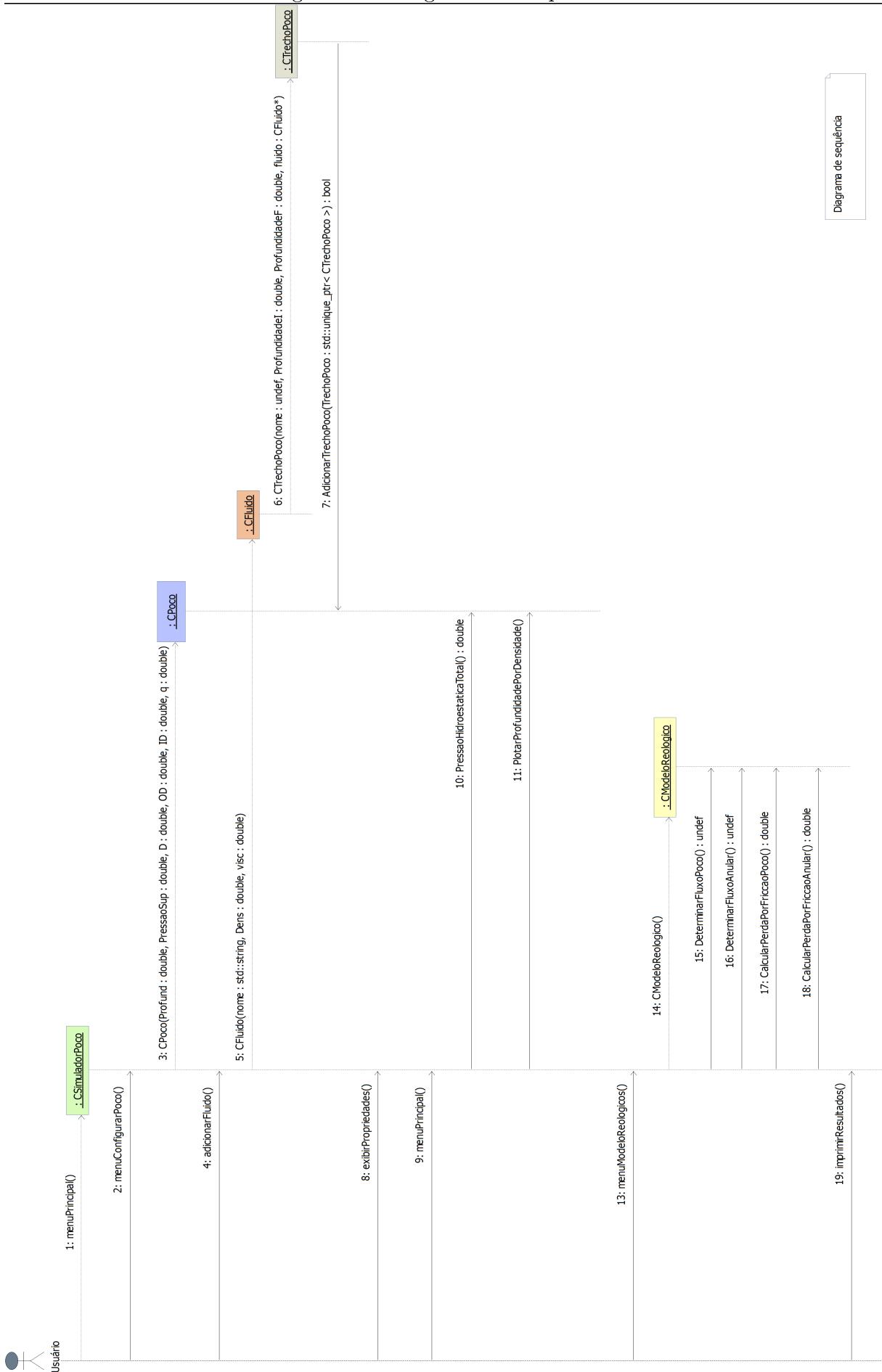
O diagrama de sequência descreve a interação entre os objetos do sistema e os elementos externos, evidenciando a ordem temporal das mensagens trocadas durante a execução de uma funcionalidade. Ele apresenta o fluxo de controle do sistema de forma cronológica, detalhando as chamadas de métodos e as respostas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

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

### 4.2.1 Diagrama de Sequência Geral

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

Figura 4.2: Diagrama de sequência

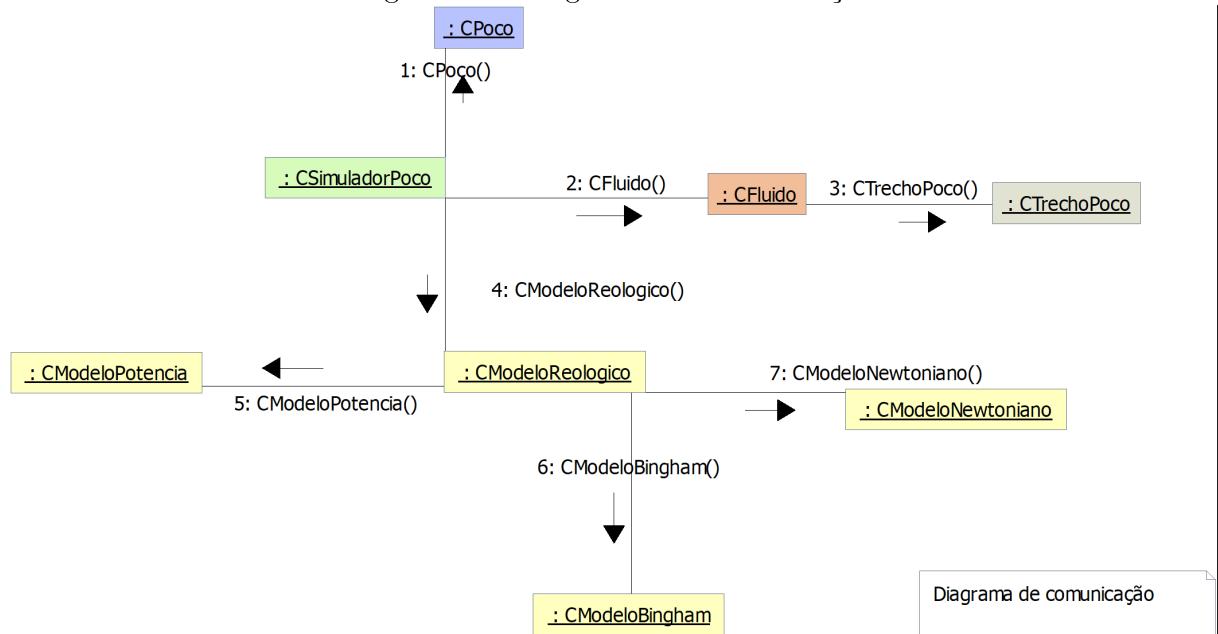


Fonte: Produzido pelo autor.

## 4.3 Diagrama de Comunicação – Colaboração

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

Figura 4.3: Diagrama de comunicação



Fonte: Produzido pelo autor.

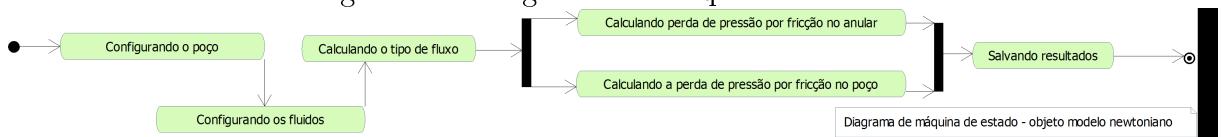
## 4.4 Diagrama de Máquina de Estado

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

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

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

Figura 4.4: Diagrama de máquina de estado



Fonte: Produzido pelo autor.

## 4.5 Diagrama de Atividades

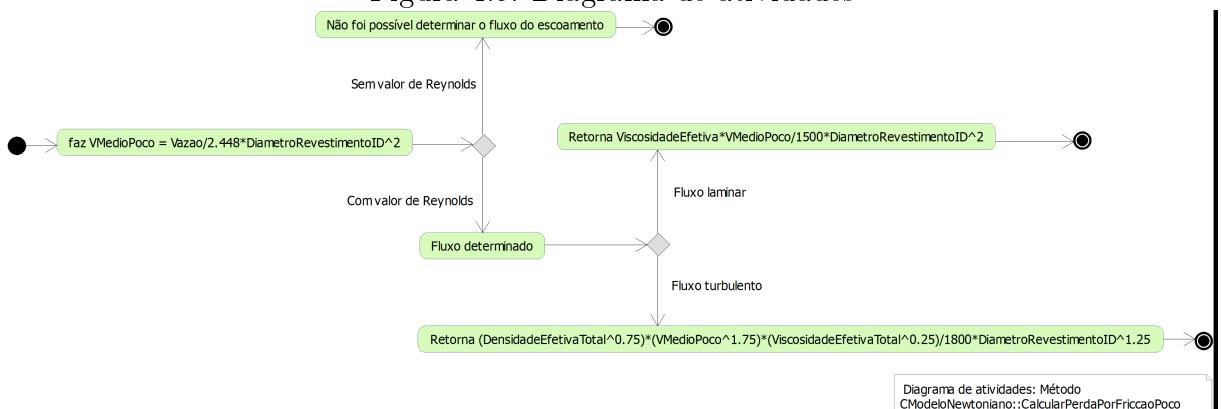
O diagrama de atividades apresentado descreve, em detalhe, a execução de uma atividade específica do sistema. No caso em questão, é representado o método **CalcularPerdaPorFriccaoPoco**, pertencente à classe **CModeloNewtoniano**.

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

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

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

Figura 4.5: Diagrama de atividades



Fonte: Produzido pelo autor.

# Capítulo 5

## Projeto

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

### 5.1 Projeto do sistema

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

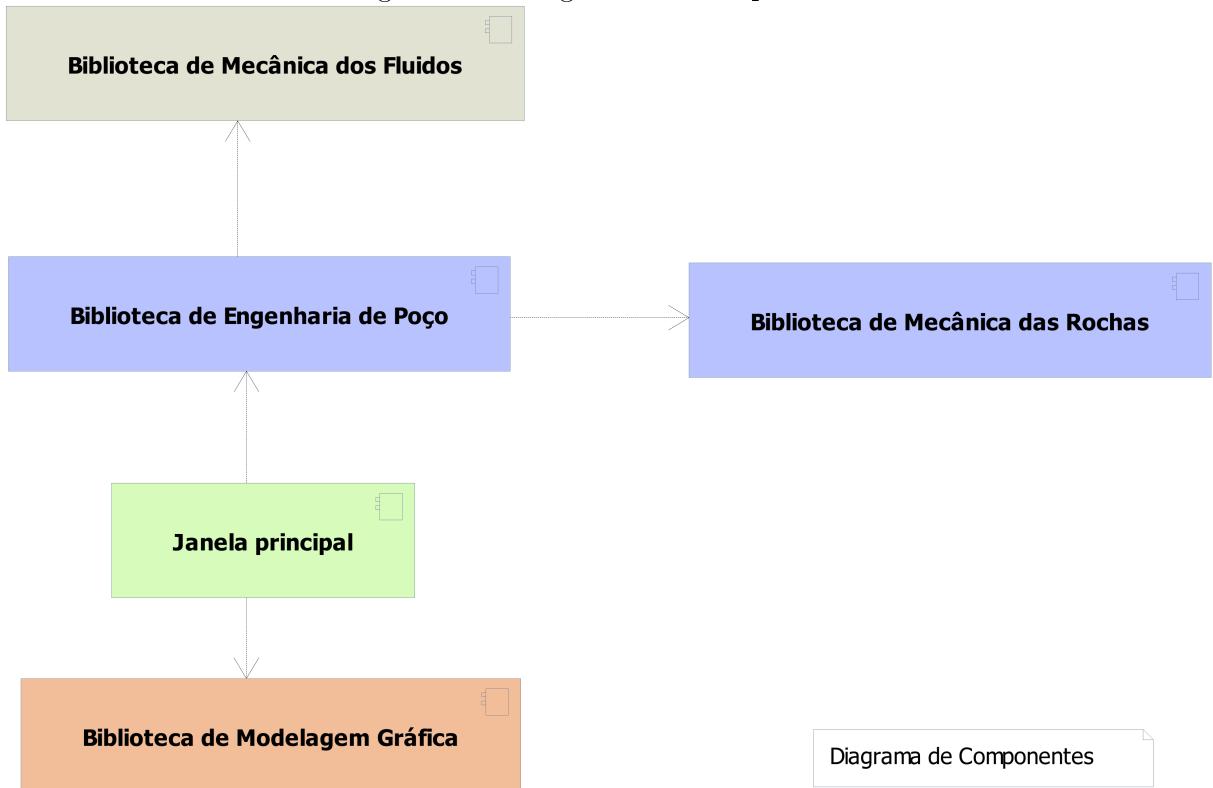
A linguagem escolhida foi o C++, em virtude de suas características que a tornam especialmente adequada para o desenvolvimento de aplicações técnicas e científicas. Os principais fatores que motivaram essa escolha incluem:

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

## 5.2 Diagrama de componentes

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

Figura 5.1: Diagrama de componentes



Fonte: Produzido pelo autor.

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

# Capítulo 6

## Ciclos de planejamento/detalhamento

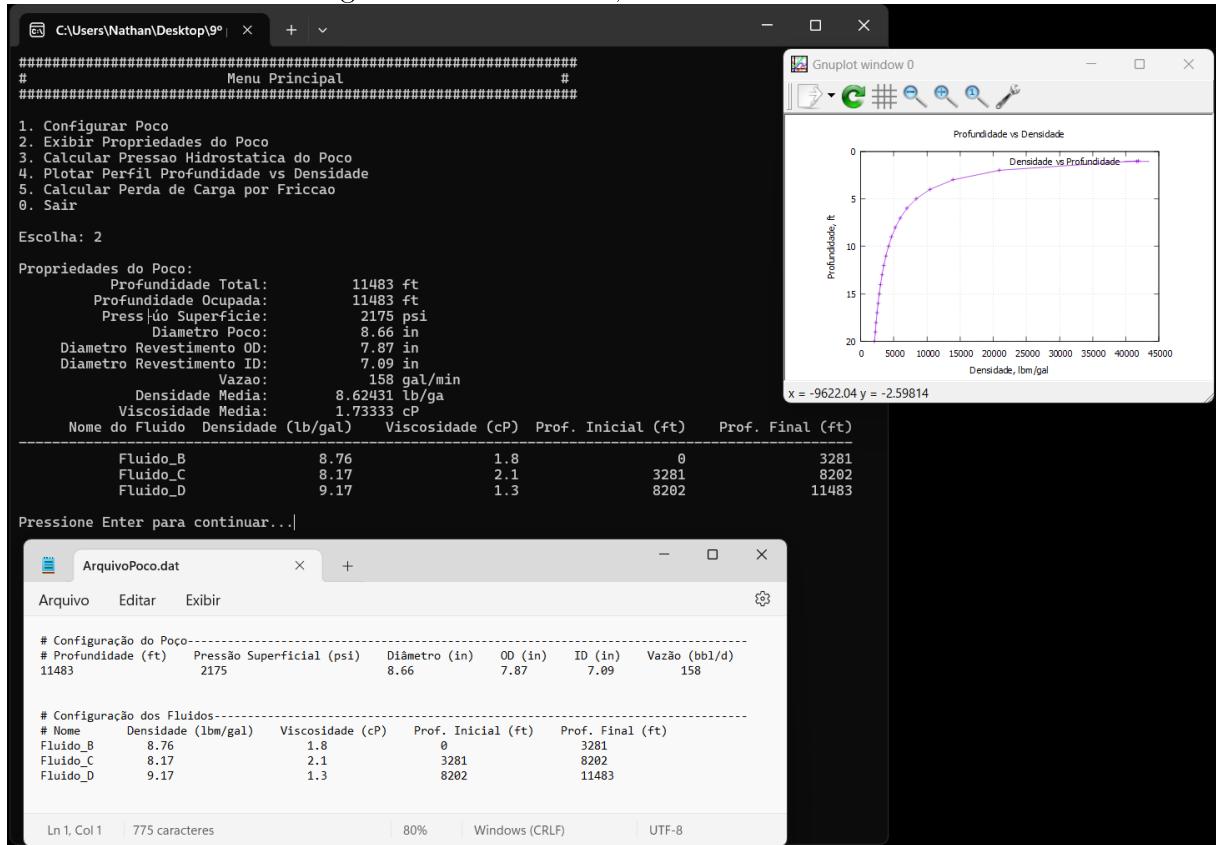
Apresenta-se neste capítulo as versões do software desenvolvido.

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

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

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

Figura 6.1: Versão 0.1, interface do software



Fonte: Produzido pelo autor.

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

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

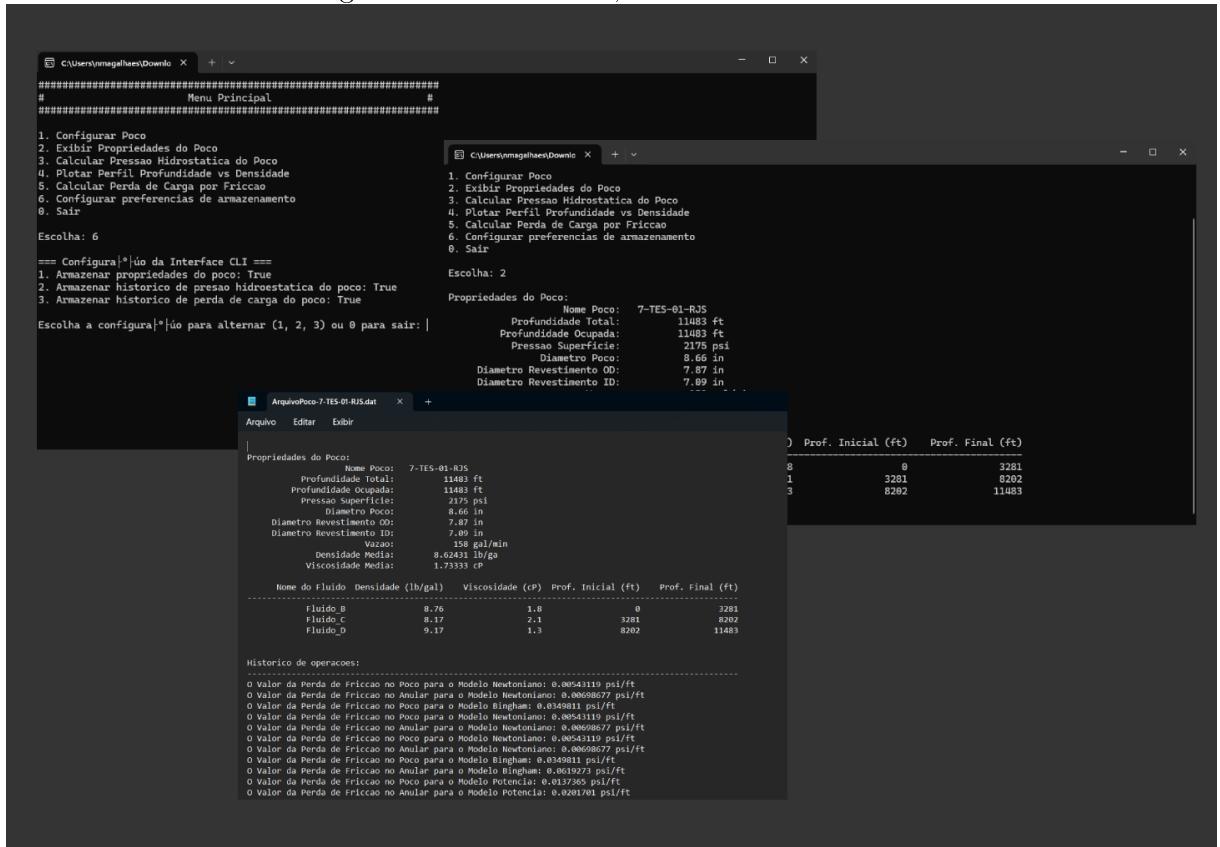
Principais aprimoramentos implementados:

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

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

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

Figura 6.2: Versão 0.2, interface do software



Fonte: Produzido pelo autor.

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

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

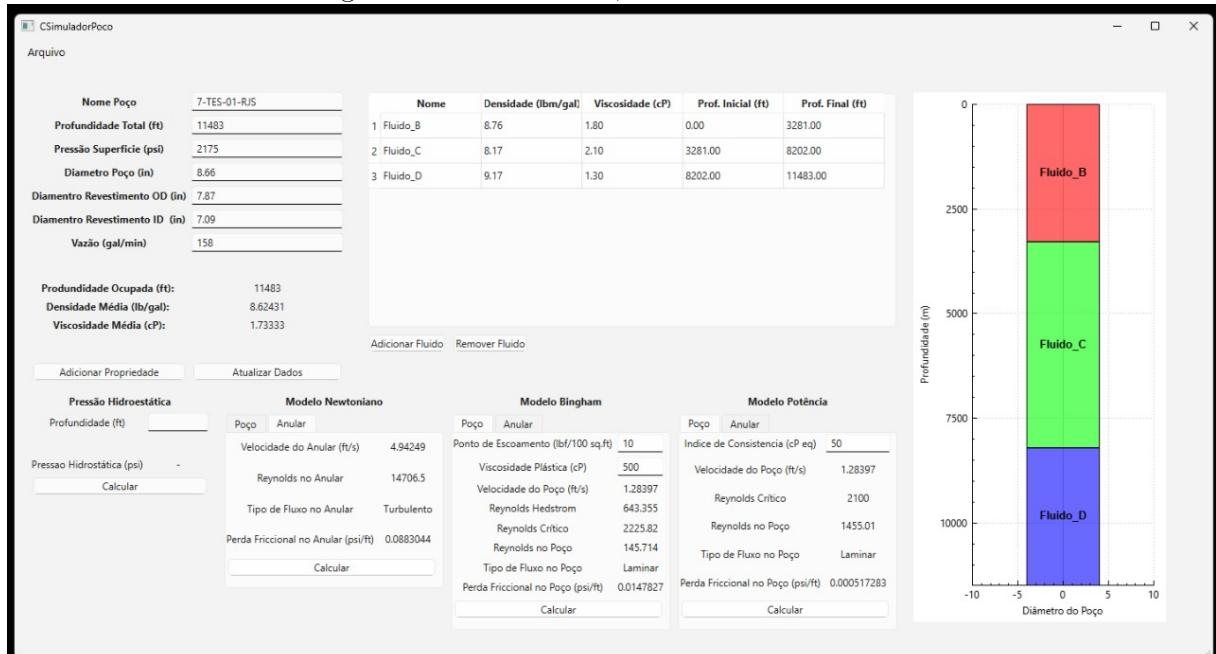
intuitiva.

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

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

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

Figura 6.3: Versão 1.0, interface do software



Fonte: Produzido pelo autor.

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

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

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

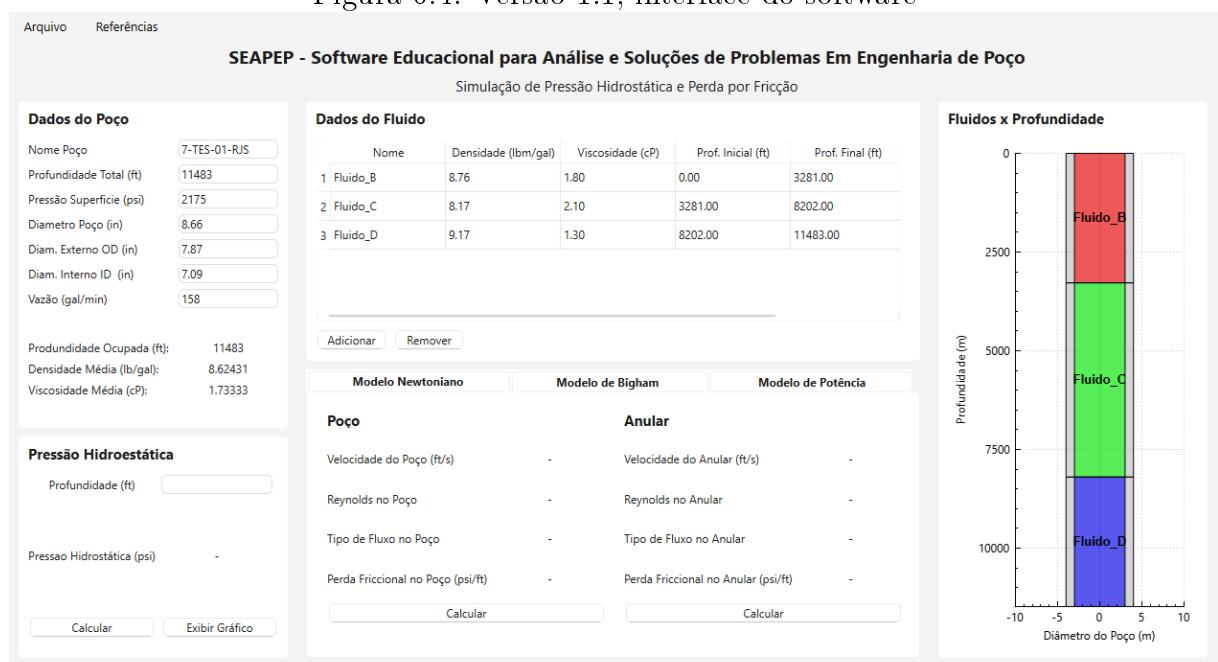
Além disso, foi implementada uma barra de tarefas com novas funcionalidades organizadas em menus acessíveis, incluindo:

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

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

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

Figura 6.4: Versão 1.1, interface do software



Fonte: Produzido pelo autor.

## 6.5 Versão 2.0 – Navegação por Módulos e Simulação Mecânica de Variação de Comprimento ( $\Delta L$ )

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

O Módulo 1 direciona para o simulador hidráulico de perfuração, que engloba todas as funcionalidades desenvolvidas até a versão 1.1, incluindo a simulação de escoamento, cálculo de pressão hidrostática e perdas por fricção com base em modelos reológicos. Esse módulo mantém a interface gráfica interativa e os recursos de visualização consolidados nas versões anteriores.

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

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

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

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

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

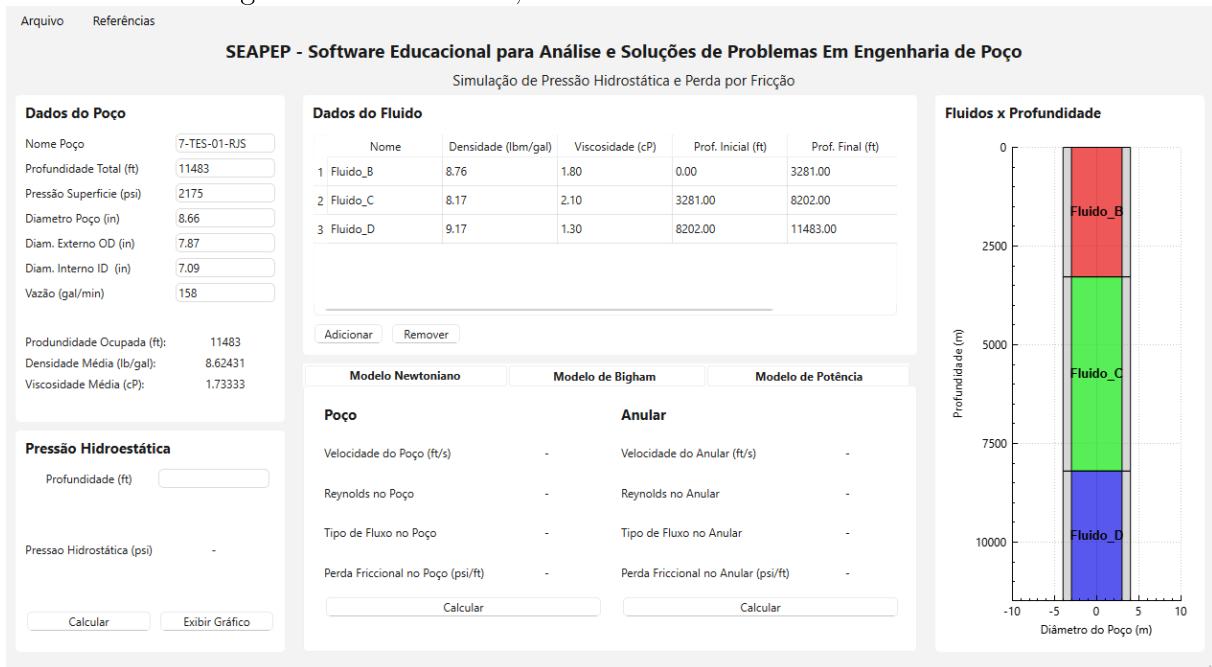
Com essa versão, o sistema passa a incorporar, além da análise hidráulica, uma abordagem mecânica de simulação, ampliando significativamente seu escopo didático e técnico na área de Engenharia de Poço.

Figura 6.5: Versão 1.1, Menu de interface do software



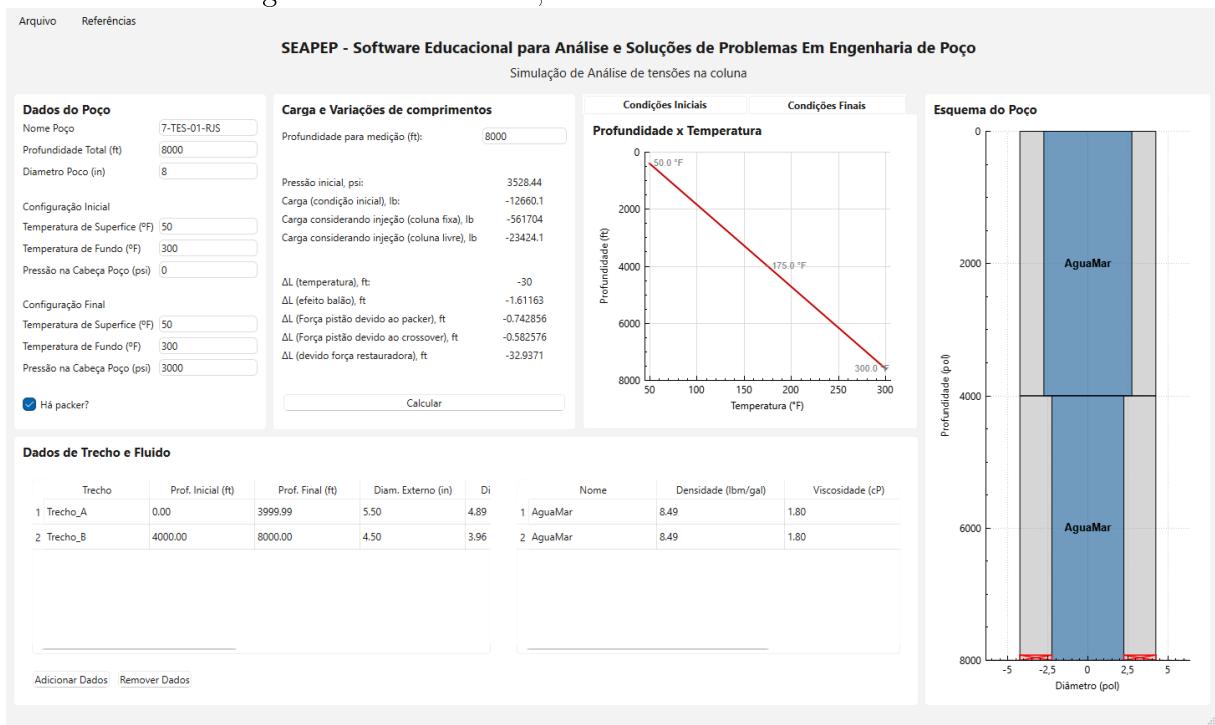
Fonte: Produzido pelo autor.

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



Fonte: Produzido pelo autor.

Figura 6.7: Versão 1.1, interface do modulo 02 software



Fonte: Produzido pelo autor.

# Capítulo 7

## Ciclos Construção - Implementação

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

### 7.1 Código-Fonte (modelo)

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

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

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

---

Listing 7.1: Arquivo de implementação da função main

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

```

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

---

Fonte: produzido pelo autor.

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

Listing 7.2: Arquivo de implementação da classe CModeloReologico

```

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

```

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

```

Fonte: produzido pelo autor.

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

---

Listing 7.3: Arquivo de implementação da classe CModeloReologico

---

```

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

```

```

3 #include <iomanip>
4
5 #include "CModeloReologico.h"
6
7 // Calcula o numero de Reynolds no poto usando a viscosidade total
8 // do fluido
9 double CModeloReologico::DeterminarReynoldsPoco() {
10    reynoldsPoco = (928 * poco->DensidadeEfetivaTotal() *
11                    vMediaPoco * poco->DiametroRevestimentoID()) /
12                    poco->ViscosidadeEfetivaTotal();
13
14    return reynoldsPoco;
15 }
16
17 // Mesmo calculo, mas permitindo passar a viscosidade como
18 // parametro
19 double CModeloReologico::DeterminarReynoldsPoco(double viscosidade)
20 {
21    reynoldsPoco = (928 * poco->DensidadeEfetivaTotal() *
22                    vMediaPoco * poco->DiametroRevestimentoID()) /
23                    viscosidade;
24
25    return reynoldsPoco;
26
27 }
28
29 // Calcula o numero de Reynolds no espaco anular
30 double CModeloReologico::DeterminarReynoldsAnular() {
31    reynoldsAnular = (757 * poco->DensidadeEfetivaTotal() *
32                       vMediaAnular *
33                           (poco->DiametroPoco() - poco->
34                            DiametroRevestimentoOD())) /
35                           poco->ViscosidadeEfetivaTotal();
36
37    return reynoldsAnular;
38 }
39
40 // Mesmo calculo para o anular, com viscosidade recebida
41 // externamente
42 double CModeloReologico::DeterminarReynoldsAnular(double
43 viscosidade) {
44    reynoldsAnular = (757 * poco->DensidadeEfetivaTotal() *
45                       vMediaAnular *
46                           (poco->DiametroPoco() - poco->
47                            DiametroRevestimentoOD())) /
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
378
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
478
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
978
979
979
980
981
982
983
984
985
986
987
987
988
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1407
1408
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1507
1508
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1607
1608
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1707
1708
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1766
1767
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1775
1775
1776
1776
1777
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1786
1787
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1795
1796
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1807
1808
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1816
1817
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1826
1827
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1836
1837
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1846
1847
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1857
1858
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1866
1867
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1875
1875
1876
1876
1877
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1886
1887
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1895
1896
1896
1897
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1907
1908
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1916
1917
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1926
1927
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1936
1937
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1946
1947
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1957
1958
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1966
1967
1967
1968
1968
1969
1969
1970
1971
1972
1973
1974
1975
1975
1976
1976
1977
1977
1978
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1986
1987
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1995
1996
1996
1997
1997
1998
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2007
2008
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2016
2017
2017
2018
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2026
2027
2027
2028
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2036
2037
2037
2038
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2046
2047
2047
2048
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2057
2058
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2066
2067
2067
2068
2068
2069
2069
2070
2071
2072
2073
2074
2075
2075
2076
2076
2077
2077
2078
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2086
2087
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2095
2096
2096
2097
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2106
2107
2107
2108
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2116
2117
2117
2118
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2126
2127
2127
2128
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2136
2137
2137
2138
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2146
2147
2147
2148
2148
2149
```

```

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

```

---

Fonte: produzido pelo autor.

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

Listing 7.4: Arquivo de implementação da classe CModeloReologico

```

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

```

```

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

```

```

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

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

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

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

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

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

```

---

Fonte: produzido pelo autor.

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

---

Listing 7.5: Arquivo de implementação da classe CModeloReológico

```

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

```

```

    CSimuladorPerdaTubulacao::EditarDadosPoco);

29
30
31     // iniciar com botões desativado
32     ui->btnAdicionarTrecho->setEnabled(false);
33     ui->btnRemoverTrecho->setEnabled(false);
34     ui->btnCalcularVariacoes->setEnabled(false);

35
36     connect(ui->tblFluidos, &QTableWidget::cellChanged, this, &
37             CSimuladorPerdaTubulacao::EditarLinhaTabela);
38     connect(ui->tblTrechos, &QTableWidget::cellChanged, this, &
39             CSimuladorPerdaTubulacao::EditarLinhaTabela);

40     // abrir janela no meio do monitor
41     QScreen *screen = QGuiApplication::primaryScreen();
42     QRect screenGeometry = screen->geometry();

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

45
46     this->move(x, y);

47
48     makePlotPoco();
49 }

50
51 CSimuladorPerdaTubulacao::~CSimuladorPerdaTubulacao()
52 {
53     delete ui;
54 }
55
56 void CSimuladorPerdaTubulacao::EditarDadosPoco() {
57     QString nome = ui->editNomePoco->text();
58     bool ok1, ok2, ok3, ok4, ok5, ok6, ok7, ok8;
59     double profund = ui->editProfundidadeTotal->text().toDouble(&
60                     ok1);
61     double diamPoco = ui->editDiametroPoco->text().toDouble(&ok2);
62     double pressao = ui->editPressaoSupInicial->text().toDouble(&
63                     ok3);
64     double pressaoFim = ui->editPressaoSupFinal->text().toDouble(&
65                     ok4);
66     double temperaturaSuperiorInicial = ui->
67         editTemperaturaSuperiorInicial->text().toDouble(&ok5);

```

```

64     double temperaturaFundoInicial = ui->
65         editTemperaturaFundoInicial->text().toDouble(&ok6);
66     double temperaturaSuperiorFinal = ui->
67         editTemperaturaSuperiorFinal->text().toDouble(&ok7);
68     double temperaturaFundoFinal = ui->editTemperaturaFundoFinal->
69         text().toDouble(&ok8);
70     bool haPacker = ui->checkBoxPacker->isChecked();
71
72
73     if (!nome.isEmpty() && ok1 && ok2 && ok3 && ok4 && ok5 && ok6
74       && ok7 && ok8) {
75         if (!poco) {
76             // Cria o p o o
77
78             poco = std::make_unique<CObjetoPoco>(
79                 CObjetoPoco::CriarParaModulo02(nome.toStdString(),
80                     profund, diamPoco, pressao, pressaoFim,
81                     temperaturaSuperiorInicial,
82                     temperaturaFundoInicial,
83                     temperaturaSuperiorFinal, temperaturaFundoFinal,
84                     haPacker)
85         );
86
87         ui->btnAdicionarTrecho->setEnabled(true);
88         ui->btnRemoverTrecho->setEnabled(true);
89         ui->btnCalcularVariacoes->setEnabled(true);
90
91         ui->statusbar->showMessage("Po o criado com Sucesso!")
92         ;
93     } else {
94         // Atualiza dados do po o j existente
95         poco->NomePoco(nome.toStdString());
96         poco->ProfundidadeTotal(profund);
97         poco->PressaoSuperficie(pressao);
98         poco->PressaoSuperficieFim(pressaoFim);
99         poco->TemperaturaTopoInicial(temperaturaSuperiorInicial
100           );
101         poco->TemperaturaFundoInicial(temperaturaFundoInicial);
102         poco->TemperaturaTopoFinal(temperaturaSuperiorFinal);
103         poco->TemperaturaFundoFinal(temperaturaFundoFinal);
104         poco->Packer(haPacker);
105
106         ui->statusbar->showMessage("Dados de Po o Atualizado
107           com Sucesso!");

```

```

94     }
95
96     AtualizarDados(); // Atualiza os dados calculados e a
97     interface
98 }
99
100 void CSimuladorPerdaTubulacao::on_btnAdicionarPropriedades_clicked
101 {
102     std::string nome;
103     double profundidade, diamPoco, pressaoSup, pressaoSupFim,
104         temperaturaSuperiorInicial, temperaturaFundoInicial,
105         temperaturaSuperiorFinal, temperaturaFundoFinal;
106     bool haPacker;
107
108     QString text;
109
110     text = ui->editNomePoco->text();
111     nome = text.toStdString();
112     text = ui->editPressaoSupInicial->text();
113     pressaoSup = text.toDouble();
114     text = ui->editPressaoSupFinal->text();
115     pressaoSupFim = text.toDouble();
116     text = ui->editProfundidadeTotal->text();
117     profundidade = text.toDouble();
118     text = ui->editDiametroPoco->text();
119     diamPoco = text.toDouble();
120     text = ui->editTemperaturaSuperiorInicial->text();
121     temperaturaSuperiorInicial = text.toDouble();
122     text = ui->editTemperaturaFundoInicial->text();
123     temperaturaFundoInicial = text.toDouble();
124     text = ui->editTemperaturaSuperiorFinal->text();
125     temperaturaSuperiorFinal = text.toDouble();
126     text = ui->editTemperaturaFundoFinal->text();
127     temperaturaFundoFinal = text.toDouble();
128     haPacker = ui->checkBoxPacker->isChecked();
129
130     if (poco) {
131         QMessageBox::StandardButton resposta = QMessageBox::
132             question(

```

```

131         this,
132         " ",
133         "Ao confirmar, todos os fluidos ser\u00e3o deletados! Tem\u00e1
134         certeza?",
135         QMessageBox::Yes | QMessageBox::No
136     );
137
138     if (resposta == QMessageBox::Yes) {
139         poco = std::make_unique<CObjetoPoco>(
140             CObjetoPoco::CriarParaModulo02(nome, profundidade,
141             diamPoco, pressaoSup, pressaoSupFim,
142             temperaturaSuperiorInicial,
143             temperaturaFundoInicial,
144             temperaturaSuperiorFinal, temperaturaFundoFinal,
145             haPacker)
146         );
147     }
148     AtualizarDados();
149 } else {
150     poco = std::make_unique<CObjetoPoco>(
151         CObjetoPoco::CriarParaModulo02(nome, profundidade,
152             diamPoco, pressaoSup, pressaoSupFim,
153             temperaturaSuperiorInicial, temperaturaFundoInicial,
154             temperaturaSuperiorFinal, temperaturaFundoFinal,
155             haPacker)
156     );
157     makePlotTemperatura(temperaturaSuperiorInicial,
158     temperaturaFundoInicial, profundidade, ui->
159     customPlotTemperaturaInicial);
160     makePlotTemperatura(temperaturaSuperiorFinal,
161     temperaturaFundoFinal, profundidade, ui->
162     customPlotTemperaturaFinal);
163     makePlotPoco();
164 }
165 void CSimuladorPerdaTubulacao::AtualizarDados()
166 {
167     ui->tblFluidos->blockSignals(true);
168     ui->tblTreichos->blockSignals(true);

```

```

159
160     if (poco){
161         // Atualiza os valores dos QLineEdit's com os dados do
162         // objeto poco
163         ui->editNomePoco->setText(QString::fromStdString(poco->
164             NomePoco));           // Profundidade total do po o
165         ui->editProfundidadeTotal->setText(QString::number(poco->
166             ProfundidadeTotal));           // Profundidade total do
167             po o
168         ui->editDiametroPoco->setText(QString::number(poco->
169             DiametroPoco));
170         ui->editPressaoSupInicial->setText(QString::number(poco->
171             PressaoSuperficie)); // Profundidade ocupada
172         ui->editPressaoSupFinal->setText(QString::number(poco->
173             PressaoSuperficieFim));
174         ui->editTemperaturaSuperiorInicial->setText(QString::number(
175             (poco->TemperaturaTopoInicial())));
176             // Di metro do po o
177         ui->editTemperaturaFundoInicial->setText(QString::number(
178             poco->TemperaturaFundoInicial()));
179             // Di metro externo do revestimento (OD)
180         ui->editTemperaturaSuperiorFinal->setText(QString::number(
181             poco->TemperaturaTopoFinal()));
182             // Di metro interno do revestimento (ID)
183         ui->editTemperaturaFundoFinal->setText(QString::number(poco
184             ->TemperaturaFundoFinal()));
185             // Vaz o do fluido no po o
186         if (poco->Packer() == true) {
187             ui->checkBoxPacker->setChecked(true);
188         } else {
189             ui->checkBoxPacker->setChecked(false);
190         }
191
192
193
194
195
196
197
198
199         // Atualizar QTableWidget com os dados dos trechos
200         ui->tblTrechos->setRowCount(static_cast<int>(poco->Trechos
201             ().size()));
202         ui->tblFluidos->setRowCount(static_cast<int>(poco->Trechos
203             ().size()));
204         int row = 0;
205         for (const auto& trecho : poco->Trechos()) {

```

```

184
185     ui->tblTrechos->setItem(row, 0, new QTableWidgetItem(
186         QString::fromStdString(trecho->Nome())));
187     ui->tblTrechos->setItem(row, 1, new QTableWidgetItem(
188         QString::number(trecho->ProfundidadeInicial(), 'f',
189         2)));
190     ui->tblTrechos->setItem(row, 2, new QTableWidgetItem(
191         QString::number(trecho->ProfundidadeFinal(), 'f',
192         2)));
193     ui->tblTrechos->setItem(row, 3, new QTableWidgetItem(
194         QString::number(trecho->DiametroExterno(), 'f',
195         2)));
196     ui->tblTrechos->setItem(row, 4, new QTableWidgetItem(
197         QString::number(trecho->DiametroInterno(), 'f',
198         2)));
199     ui->tblTrechos->setItem(row, 5, new QTableWidgetItem(
200         QString::number(trecho->CoeficientePoisson(), 'f',
201         2)));
202     ui->tblTrechos->setItem(row, 6, new QTableWidgetItem(
203         QString::number(trecho->CoeficienteExpancaoTermica(),
204         'f', 8)));
205     ui->tblTrechos->setItem(row, 7, new QTableWidgetItem(
206         QString::number(trecho->ModuloElasticidade(), 'f',
207         2)));
208     ui->tblTrechos->setItem(row, 8, new QTableWidgetItem(
209         QString::number(trecho->PesoUnidade(), 'f',
210         2)));
211
212     ui->tblFluidos->setItem(row, 0, new QTableWidgetItem(
213         QString::fromStdString(trecho->Fluido()->Nome())));
214     ui->tblFluidos->setItem(row, 1, new QTableWidgetItem(
215         QString::number(trecho->Fluido()->Densidade(), 'f',
216         2)));
217     ui->tblFluidos->setItem(row, 2, new QTableWidgetItem(
218         QString::number(trecho->Fluido()->Viscosidade(), 'f',
219         2)));
220
221     ++row;
222 }
223 }
224 makePlotTemperatura(poco->TemperaturaTopoInicial(), poco->
225 TemperaturaFundoInicial(), poco->ProfundidadeTotal(), ui->
226 customPlotTemperaturaInicial);

```

```

203     makePlotTemperatura(poco->TemperaturaTopoFinal(), poco->
204         TemperaturaFundoFinal(), poco->ProfundidadeTotal(), ui->
205         customPlotTemperaturaFinal);
206     makePlotPoco();
207
208 }
209
210 void CSimuladorPerdaTubulacao::on_btnAdicionarTrecho_clicked()
211 {
212     ui->tblTreichos->setEditTriggers(QAbstractItemView::NoEditTriggers);
213     ui->tblFluidos->setEditTriggers(QAbstractItemView::NoEditTriggers);
214
215     if (!poco) {
216         QMessageBox::warning(this, "Erro", "As propriedades do p o o
217             precisam est   preenchida!");
218     }
219     else{
220         CJanelaAdicionarTrechoTubulacao JanelaTrecho;
221         JanelaTrecho.exec();
222
223         if (JanelaTrecho.Trecho() != "" &&
224             JanelaTrecho.ProfundidadeInicial() != "" &&
225             JanelaTrecho.ProfundidadeFinal() != "" &&
226             JanelaTrecho.DiametroExterno() != "" &&
227             JanelaTrecho.DiametroInterno() != "" &&
228             JanelaTrecho.CoefficientePoisson() != "" &&
229             JanelaTrecho.CoefficienteExpansaoTermica() != "" &&
230             JanelaTrecho.ModuloElasticidade() != "" &&
231             JanelaTrecho.PesoUnidade() != "" &&
232             JanelaTrecho.NomeFluido() != "" &&
233             JanelaTrecho.Densidade() != "" &&
234             JanelaTrecho.Viscosidade() != ""){
235
236             int numLinhas = ui->tblTreichos->rowCount();
237             ui->tblTreichos->insertRow(numLinhas);

```

```

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

262     ui->tblFluidos->insertRow(numLinhas);
263     ui->tblFluidos->setItem(numLinhas, 0, new
264         QTableWidgetItem(JanelaTrecho.NomeFluido()));
265     ui->tblFluidos->setItem(numLinhas, 1, new
266         QTableWidgetItem(JanelaTrecho.Densidade()));
267     ui->tblFluidos->setItem(numLinhas, 2, new
268         QTableWidgetItem(JanelaTrecho.Viscosidade()));

269     std::string NomeTrecho = JanelaTrecho.Trecho().
270         toStdString();
271     double profundInicial = JanelaTrecho.
272         ProfundidadeInicial().toDouble();
273     double profundFinal = JanelaTrecho.ProfundidadeFinal().
274         toDouble();
275     double diametroExterno = JanelaTrecho.DiametroExterno()
276         .toDouble();
277     double diametroInterno = JanelaTrecho.DiametroInterno()
278         .toDouble();
279     double coeficientePoisson = JanelaTrecho.

```

```

261             CoeficientePoisson().toDouble();
262             double coeficienteExpansaoTermica = JanelaTrecho.
263                 CoeficienteExpansaoTermica().toDouble();
264             double moduloElasticidade = JanelaTrecho.
265                 ModuloElasticidade().toDouble();
266             double pesoUnidade = JanelaTrecho.PesoUnidade().
267                 toDouble();

268             std::string nome = JanelaTrecho.NomeFluido().
269                 toStdString();
270             double densidade = JanelaTrecho.Densidade().toDouble();
271             double viscosidade = JanelaTrecho.Viscosidade().
272                 toDouble();

273             auto fluido = std::make_unique<CFluido>(nome, densidade,
274                 , viscosidade);
275             auto trechoPoco = std::make_unique<CTrechoPoco>(
276                 NomeTrecho, profundInicial, profundFinal, std::move(
277                     fluido), diametroExterno, diametroInterno,
278                 coeficientePoisson, coeficienteExpansaoTermica,
279                 moduloElasticidade, pesoUnidade);
280             poco->AdicionarTrechoPoco(std::move(trechoPoco));
281
282             AtualizarDados();
283         }
284     }
285 }
286
287 void CSimuladorPerdaTubulacao::makePlotTemperatura(double
288 TempInicial, double TempFinal, double profundidade, QCustomPlot*
289 plot)
290 {
291     // Limpa graficos anteriores
292     plot->clearItems();
293     plot->clearPlottables();
294
295     // Configura eixos
296     plot->xAxis->setLabel("Temperatura ( F )");
297     plot->yAxis->setLabel("Profundidade (ft)");
298     plot->yAxis->setRangeReversed(true); // profundidade cresce pra
299         baixo
300
301 }
```

```

289     // Adiciona "respiro" nos extremos
290     double margemProfundidade = profundidade * 0.05; // 5% de
291     margem visual
292
293     // Define pontos
294     double tempMeio = (TempInicial + TempFinal) / 2.0;
295     QVector<double> temperaturas = {TempInicial, tempMeio,
296                                     TempFinal};
297     QVector<double> profundidades = {0.0 + margemProfundidade,
298                                     profundidade / 2.0, profundidade - margemProfundidade};
299
300     // Ajuste de range
301     double tempMin = *std::min_element(temperaturas.begin(),
302                                         temperaturas.end());
303     double tempMax = *std::max_element(temperaturas.begin(),
304                                         temperaturas.end());
305
306     plot->xAxis->setRange(tempMin - 5, tempMax + 5);
307     plot->yAxis->setRange(0.0, profundidade); // Mantem 0 a
308                                         profundidade total, o respiro e visual apenas
309
310     // Grid leve
311     plot->xAxis->grid()->setVisible(true);
312     plot->yAxis->grid()->setVisible(true);
313     plot->xAxis->grid()->setPen(QPen(QColor(220, 220, 220)));
314     plot->yAxis->grid()->setPen(QPen(QColor(220, 220, 220)));
315
316     // Linha do perfil
317     QCPIGraph *perfilTemp = plot->addGraph();
318     perfilTemp->setData(temperaturas, profundidades);
319     perfilTemp->setPen(QPen(QColor(200, 0, 0), 2));
320
321     // Marcar e rotular os 3 pontos (topo, meio e fundo)
322     for (int i = 0; i < temperaturas.size(); ++i) {
323         QCPIItemEllipse *ponto = new QCPIItemEllipse(plot);
324         ponto->topLeft->setCoords(temperaturas[i] - 2,
325                                     profundidades[i] - 20);
326         ponto->bottomRight->setCoords(temperaturas[i] + 2,
327                                         profundidades[i] + 20);
328         ponto->setPen(Qt::NoPen);
329         ponto->setBrush(QBrush(Qt::darkGray));
330
331
332

```

```

323     QCPIItemText *rotulo = new QCPIItemText(plot);
324     rotulo->position->setCoords(temperaturas[i] + 4,
325         profundidades[i]);
326     rotulo->setText(QString::number(temperaturas[i], 'f', 1) +
327         " ° F ");
328     rotulo->setFont(QFont("Arial", 8));
329     rotulo->setColor(Qt::darkGray);
330     // Ajusta alinhamento do rotulo do ultimo ponto para dentro
331     // do grafico
332     if (i == temperaturas.size() - 1)
333         rotulo->setPositionAlignment(Qt::AlignRight | Qt::
334             AlignVCenter); // alinha para a esquerda do ponto
335     final
336     else
337         rotulo->setPositionAlignment(Qt::AlignLeft | Qt::
338             AlignVCenter);
339 }
340
341 // Limpeza estatica
342 plot->legend->setVisible(false);
343 plot->setBackground(Qt::white);
344 plot->xAxis->setBasePen(QPen(Qt::black));
345 plot->yAxis->setBasePen(QPen(Qt::black));
346 plot->xAxis->setTickPen(QPen(Qt::black));
347 plot->yAxis->setTickPen(QPen(Qt::black));
348 plot->xAxis->setTickLabelColor(Qt::black);
349 plot->yAxis->setTickLabelColor(Qt::black);
350
351
352 void CSimuladorPerdaTubulacao::on_btnRemoverTrecho_clicked()
353 {
354     // pega qual linha ta selecionada em cada tabela
355     int linhaSelecionadaTrecho = ui->tblTrechos->currentRow();
356     int linhaSelecionadaFluido = ui->tblFluidos->currentRow();
357
358     // se a selecao for feita na tabela de trecho, faz a remocao

```

```

359     if (linhaSelecionadaTrecho >= 0) {
360         QMessageBox::StandardButton resposta = QMessageBox::
361             question(
362                 this,
363                 "Confirma o",
364                 "Deseja\u00e1 remover\u00e1 trecho\u00e9 ou\u00e3 fluido\u00e1 associado?",
365                 QMessageBox::Yes | QMessageBox::No
366             );
367
368         if (resposta == QMessageBox::Yes) {
369             // pega o nome do trecho pela tabela (pra remover do
370             // objeto poco)
371             QString nomeTrecho = ui->tblTrechos->item(
372                 linhaSelecionadaTrecho, 0)->text();
373
374             // remove a mesma linha das duas tabelas
375             ui->tblTrechos->removeRow(linhaSelecionadaTrecho);
376             ui->tblFluidos->removeRow(linhaSelecionadaTrecho);
377
378             // remove o trecho (e junto o fluido) do objeto poco
379             poco->RemoverTrechoPoco(nomeTrecho.toStdString());
380
381             // atualiza visualmente os dados
382             AtualizarDados();
383             ui->statusbar->showMessage("Trecho\u00e1 removido\u00e1 com\u00e1 sucesso
384             !");
385         }
386
387     } else if (linhaSelecionadaFluido >= 0) {
388         // se clicou so na tabela de fluido, avisa que deve usar a
389         // outra
390         QMessageBox::warning(this, "Aviso", "A\u00e1 remo\u00e1 o\u00e1 deve ser\u00e1
391         feita\u00e1 pela\u00e1 tabela\u00e1 de\u00e1 trechos.");
392     } else {
393         // nenhuma linha foi selecionada
394         QMessageBox::warning(this, "Erro", "Nenhuma\u00e1 linha\u00e1 foi\u00e1
395         selecionada.");
396     }
397
398 }
399
400
401 void CSimuladorPerdaTubulacao::makePlotPoco()

```

```

394 {
395     ui->customPlotPoco->clearItems();
396     ui->customPlotPoco->xAxis->setLabel("Di metro (pol)");
397     ui->customPlotPoco->yAxis->setLabel("Profundidade (pol)");
398     ui->customPlotPoco->yAxis->setRangeReversed(true);
399
400     if (!poco || poco->Trechos().empty())
401         return;
402
403     // 1. Profundidade máxima e maior di metro externo
404     double profundidadeMaxima = 0.0;
405     double maiorDiametroExterno = 0.0;
406     for (const auto& trecho : poco->Trechos()) {
407         profundidadeMaxima = std::max(profundidadeMaxima, trecho->
408             ProfundadeFinal());
409         maiorDiametroExterno = std::max(maiorDiametroExterno,
410             trecho->DiametroExterno());
411     }
412
413     // 2. Buraco = maior di metro externo + 3 polegadas
414     double diametroBuraco = maiorDiametroExterno + 3.0;
415
416     // 3. Ajuste visual no eixo X: 1.5 vezes o furo
417     double larguraGrafico = diametroBuraco * 1.5;
418     ui->customPlotPoco->xAxis->setRange(-larguraGrafico / 2.0,
419         larguraGrafico / 2.0);
420     ui->customPlotPoco->yAxis->setRange(0, profundidadeMaxima);
421
422     // 4. Cores dos fluidos
423     QMap<QString, QColor> mapaCores;
424     QVector<QColor> coresDisponiveis = {
425         QColor(70, 130, 180, 180), QColor(255, 0, 0, 150),
426         QColor(0, 255, 0, 150), QColor(0, 0, 255, 150),
427         QColor(255, 165, 0, 150), QColor(128, 0, 128, 150)
428     };
429     int corIndex = 0;
430
431     // 5. Desenho dos trechos
432     for (const auto& trecho : poco->Trechos()) {
433         double z1 = trecho->ProfundidadeInicial();
434         double z2 = trecho->ProfundadeFinal();
435         double dExt = trecho->DiametroExterno();

```

```

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

```

```

469     profundidadePacker = poco->Trechos().back()->
470         ProfundidadeFinal();
471 }
472
473     double alturaPacker = std::max(profundidadeMaxima * 0.01, 12.0)
474         ;
475     double zTop, zBottom;
476
477     // Se o packer estiver exatamente na profundidade máxima,
478     // desenha ele todo acima
479     if (std::abs(profundidadePacker - profundidadeMaxima) < 1e-3) {
480         zBottom = profundidadePacker;
481         zTop = profundidadePacker - alturaPacker;
482     } else {
483         zTop = profundidadePacker - alturaPacker / 2.0;
484         zBottom = profundidadePacker + alturaPacker / 2.0;
485     }
486
487     // Encontrar o trecho correspondente à profundidade do packer
488     double diametroNoPacker = 0.0;
489     for (const auto& trecho : poco->Trechos()) {
490         if (profundidadePacker >= trecho->ProfundidadeInicial() &&
491             profundidadePacker <= trecho->ProfundidadeFinal()) {
492             diametroNoPacker = trecho->DiametroExterno();
493             break;
494         }
495     }
496
497     // Se não achou trecho correspondente, usa o maior conhecido
498     // como fallback
499     if (diametroNoPacker == 0.0)
500         diametroNoPacker = maiorDiametroExterno;
501
502     // Coordenadas horizontais para os quadrados laterais
503     double xEsq1 = -diametroBuraco / 2.0;
504     double xEsq2 = -diametroNoPacker / 2.0;
505     double xDir1 = diametroNoPacker / 2.0;
506     double xDir2 = diametroBuraco / 2.0;
507
508     // === Quadrado esquerdo ===
509     QCPIItemRect* rectPackerEsq = new QCPIItemRect(ui->
510         customPlotPoco);

```

```

506     rectPackerEsq->topLeft->setCoords(xEsq1, zTop);
507     rectPackerEsq->bottomRight->setCoords(xEsq2, zBottom);
508     rectPackerEsq->setPen(QPen(Qt::red, 1.5));
509     rectPackerEsq->setBrush(Qt::NoBrush);

510
511 // === Quadrado direito ===
512 QCPIItemRect* rectPackerDir = new QCPIItemRect(ui->
513     customPlotPoco);
514     rectPackerDir->topLeft->setCoords(xDir1, zTop);
515     rectPackerDir->bottomRight->setCoords(xDir2, zBottom);
516     rectPackerDir->setPen(QPen(Qt::red, 1.5));
517     rectPackerDir->setBrush(Qt::NoBrush);

518
519 // === X vermelho esquerdo ===
520 QCPIItemLine* linha1Esq = new QCPIItemLine(ui->customPlotPoco
521     );
522     linha1Esq->start->setCoords(xEsq1, zTop);
523     linha1Esq->end->setCoords(xEsq2, zBottom);
524     linha1Esq->setPen(QPen(Qt::red, 1.5));

525
526 QCPIItemLine* linha2Esq = new QCPIItemLine(ui->customPlotPoco
527     );
528     linha2Esq->start->setCoords(xEsq2, zTop);
529     linha2Esq->end->setCoords(xEsq1, zBottom);
530     linha2Esq->setPen(QPen(Qt::red, 1.5));

531
532 // === X vermelho direito ===
533 QCPIItemLine* linha1Dir = new QCPIItemLine(ui->customPlotPoco
534     );
535     linha1Dir->start->setCoords(xDir1, zTop);
536     linha1Dir->end->setCoords(xDir2, zBottom);
537     linha1Dir->setPen(QPen(Qt::red, 1.5));

538
539 QCPIItemLine* linha2Dir = new QCPIItemLine(ui->customPlotPoco
540     );
541     linha2Dir->start->setCoords(xDir2, zTop);
542     linha2Dir->end->setCoords(xDir1, zBottom);
543     linha2Dir->setPen(QPen(Qt::red, 1.5));

544 }

545
546 // 7. Linha tracejada indicando profundidade de medi o , se
547 v lida

```

```

542     bool ok = false;
543     double profundidadeMedicao = ui->editProfundidadeMedicao->text
544         ().toDouble(&ok);
545
546     // So desenha se a conversao foi bem-sucedida e o valor for
547     // maior que zero
548     if (ok && profundidadeMedicao > 0.0) {
549         QCPItemLine* linhaMedicao = new QCPItemLine(ui->
550             customPlotPoco);
551         linhaMedicao->start->setCoords(-larguraGrafico / 2.0,
552             profundidadeMedicao);
553         linhaMedicao->end->setCoords(larguraGrafico / 2.0,
554             profundidadeMedicao);
555
556         // Define o estilo como linha tracejada preta
557         QPen pen(Qt::black);
558         pen.setStyle(Qt::DashLine);
559         pen.setWidthF(1.5);
560         linhaMedicao->setPen(pen);
561     }
562
563     ui->customPlotPoco->replot();
564 }
565
566 void CSimuladorPerdaTubulacao::on_btnCalcularVariacoes_clicked()
567 {
568
569     double pressaoCabeca = ui->editPressaoSupFinal->text().toDouble
570         ();
571
572     QString profundidadeStr = ui->editProfundidadeMedicao->text();
573     double profundidade = profundidadeStr.toDouble();
574
575     ui->lbnPressaoHidroestatica->setText(QString::number( (poco->
576         PressaoHidroestaticaNoPonto(profundidade)) ));
577     ui->lbnCargaInicial->setText(QString::number(poco->Carga(
578         profundidade, true)));
579
580     ui->lbnTituloDeltaTemperatura->setText(QString::number(poco->
581         DeltaTemperatura(profundidade)));
582     ui->lbnCargaInjecaoColunaFixa->setText(QString::number(poco->
583         CargaInjecao(profundidade)));

```

```

574     ui ->lbnCargaInjecaoColunaLivre ->setText(QString::number(poco ->
575         Carga(profundidade, false)));
576     ui ->lbnDeltaLPistaoPacker ->setText(QString::number(poco ->
577         DeltaLPistaoPacker(profundidade)));
578     ui ->lbnTituloDeltaLBalao ->setText(QString::number(poco ->
579         DeltaLEfeitoBalao(profundidade)));
580     ui ->lbnDeltaLPistaoCrossover ->setText(QString::number(poco ->
581         DeltaLPistaoCrossover(profundidade)));
582     ui ->lbnDeltaLForcaRestauradora ->setText(QString::number(poco ->
583         DeltaLForcaRestauradora(profundidade)));
584 }
585
586
587
588 void CSimuladorPerdaTubulacao::on_actionArquivo_Dat_triggered()
589 {
590     QString caminhoDoArquivo = QFileDialog::getOpenFileName(
591         this,
592         "Selecione um arquivo",
593         "",
594         "Todos os arquivos (*.*)"
595     );
596
597     std::string caminhoDoArquivoStr = caminhoDoArquivo.toStdString()
598         ();
599     std::ifstream file(caminhoDoArquivoStr);
600
601     if (!file.is_open()) {
602         ui ->statusbar ->showMessage("Falha ao abrir o arquivo!");
603         return;
604     }
605
606     std::string linha;
607     bool lendoTreichos = false;
608
609     while (std::getline(file, linha)) {
610         if (linha.find("Configuracao dos Fluidos") != std::string::npos) {
611             lendoTreichos = true;
612             continue;
613         }
614
615         if (linha.empty() || linha[0] == '#') {

```

```

609         continue;
610     }
611
612     if (!lendoTrechos) {
613         // Leitura dos dados do p o o
614         std::istringstream iss(linha);
615         std::string nome;
616         double profundidade, diamPoco, pressaoSup,
617             pressaoSupFim;
618         double temperaturaSuperiorInicial,
619             temperaturaFundoInicial;
620         double temperaturaSuperiorFinal, temperaturaFundoFinal;
621         std::string strHaPacker;
622
623         if (iss >> nome >> profundidade >> diamPoco >>
624             pressaoSup >> pressaoSupFim
625             >> temperaturaSuperiorInicial >>
626             temperaturaFundoInicial
627             >> temperaturaSuperiorFinal >>
628             temperaturaFundoFinal >> strHaPacker) {
629
630             ui->btnAdicionarTrecho->setEnabled(true);
631             ui->btnRemoverTrecho->setEnabled(true);
632             ui->btnCalcularVariacoes->setEnabled(true);
633
634             bool haPacker = (strHaPacker == "true" ||
635                               strHaPacker == "1");
636
637             poco = std::make_unique<CObjetoPoco>(
638                 CObjetoPoco::CriarParaModulo02(nome,
639                     profundidade, diamPoco, pressaoSup,
640                     pressaoSupFim,
641                     temperaturaSuperiorInicial
642                     ,
643                     temperaturaFundoInicial
644                     ,
645                     );

```

```

640                                     temperaturaSuperiorFinal
641                                     ,
642                                     temperaturaFundoFinal
643                                     , haPacker)
644                               );
645 } else {
646     std::cerr << "Erro ao ler linha de po o : " <<
647     linha << std::endl;
648 }
649 } else {
650     // Leitura dos dados dos trechos
651     std::istringstream iss(linha);
652     std::string nomeTrecho, nomeFluido;
653     double profundInicial, profundFinal;
654     double diametroExterno, diametroInterno;
655     double coeficientePoisson, coeficienteExpansaoTermica;
656     double moduloElasticidade, pesoUnidade;
657     double densidade, viscosidade;
658
659     if (iss >> nomeTrecho >> profundInicial >> profundFinal
660         >> diametroExterno >> diametroInterno
661         >> coeficientePoisson >> coeficienteExpansaoTermica
662         >> moduloElasticidade >> pesoUnidade
663         >> nomeFluido >> densidade >> viscosidade) {
664
665         auto fluido = std::make_unique<CFluido>(nomeFluido,
666                                         densidade, viscosidade);
667         auto trechoPoco = std::make_unique<CTrechoPoco>(
668             nomeTrecho,
669             profundInicial, profundFinal, std::move(fluido)
670             ,
671             diametroExterno, diametroInterno,
672             coeficientePoisson, coeficienteExpansaoTermica,
673             moduloElasticidade, pesoUnidade
674         );
675
676         if (!poco->AdicionarTrechoPoco(std::move(trechoPoco
677             )))) {
678             std::cerr << "Falha ao adicionar trecho ao
679             po o .\n";
680         }
681     }
682 }
```

```

673         } else {
674             std::cerr << "Erro ao ler o trecho:" << linha << std
675             ::endl;
676         }
677     }
678
679     file.close();
680     AtualizarDados();
681     ui->statusbar->showMessage("Dados importados com sucesso!");
682 }
683
684
685 void CSimuladorPerdaTubulacao::on_actionNova_Simula_o_triggered()
686 {
687     QMessageBox::StandardButton resposta = QMessageBox::question(
688         this,
689         "",
690         "Tem certeza que deseja iniciar uma nova simulação?",
691         QMessageBox::Yes | QMessageBox::No
692     );
693
694     if (resposta == QMessageBox::Yes) {
695         CSimuladorPerdaTubulacao *newWindow = new
696             CSimuladorPerdaTubulacao();
697         newWindow->show();
698         this->close();
699     }
700
701
702 void CSimuladorPerdaTubulacao::
703     on_actionExportar_Como_Imagem_triggered()
704 {
705     QString fileName = QFileDialog::getSaveFileName(this, "Salvar
706         imagem", "", "PNG (*.png);;JPEG (*.jpg)");
707
708     if (!fileName.isEmpty()) {
709         QPixmap pixmap = this->grab();
710         pixmap.save(fileName);
711     }
712 }
```

```

711
712
713 void CSimuladorPerdaTubulacao::on_actionSobre_o_SEEP_triggered()
714 {
715     CJanelaSobreSoftware janelaSobre;
716     janelaSobre.setWindowTitle("Sobre o Software");
717     janelaSobre.exec();
718 }
719
720
721 void CSimuladorPerdaTubulacao::SalvarArquivo(bool salvarComo)
722 {
723     QString caminho;
724
725     // Se for salvarComo ou ainda não tiver caminho, abrir o
726     // dia logo
727     if (salvarComo || CaminhoArquivo().isEmpty()) {
728         caminho = QFileDialog::getSaveFileName(this, "Salvar"
729             " Arquivo", "", "Arquivo DAT (*.dat)");
730         if (caminho.isEmpty()) return; // usuário cancelou
731         CaminhoArquivo(caminho);
732         NomeArquivo(QFileInfo(caminho).fileName());
733     } else {
734         caminho = CaminhoArquivo(); // salva direto
735     }
736
737     QFile arquivo(caminho);
738     if (!arquivo.open(QIODevice::WriteOnly | QIODevice::Text)) {
739         QMessageBox::warning(this, "Erro", "Não foi possível salvar"
740             " o arquivo.");
741         return;
742     }
743     QTextStream out(&arquivo);
744
745     out << "# Configuração do Poco"
746     -----
747     n";
748     out << "#"
749         << QString("Nome").leftJustified(35, ' ')
750         << QString("Profundidade (ft)").leftJustified(35, ' ')
751         << QString("Diâmetro do Poco (ft)").leftJustified(35, ' ')

```

```

748     << QString("Pressao\u2022Sup.\u2022Inicial\u2022(psi)").leftJustified(35,
749         '\u2022')
750     << QString("Pressao\u2022Sup.\u2022Final\u2022(psi)").leftJustified(35, '\u2022'
751         )
752     << QString("Temp\u2022Sup.\u2022Inicial\u2022( F )").leftJustified(35, '\u2022'
753         )
754     << QString("Temp\u2022Fund.\u2022Inicial\u2022( F )").leftJustified(35, '\u2022'
755         )
756     << QString("Prof\u2022Packer\u2022(ft)").leftJustified(35, '\u2022')
757     << "\n";
758
759     out << " \u2022"
760     << ui->editNomePoco->text().leftJustified(35, '\u2022')
761     << ui->editProfundidadeTotal->text().leftJustified(35, '\u2022')
762     << ui->editDiametroPoco->text().leftJustified(35, '\u2022')
763     << ui->editPressaoSupInicial->text().leftJustified(35, '\u2022')
764     << ui->editPressaoSupFinal->text().leftJustified(35, '\u2022')
765     << ui->editTemperaturaSuperiorInicial->text().leftJustified
766         (35, '\u2022')
767     << ui->editTemperaturaFundoInicial->text().leftJustified
768         (35, '\u2022')
769     << ui->editTemperaturaSuperiorFinal->text().leftJustified
770         (35, '\u2022')
771     << ui->editTemperaturaFundoFinal->text().leftJustified(35,
772         '\u2022');
773
774     // Checkbox de saida
775     if (ui->checkBoxPacker->isChecked()) {
776         out << "true";
777     } else {
778         out << "false";
779     }
780
781     out << "\n";
782
783     // Escreve os dados dos fluidos
784     out << "\n\n\n#\u2022Configuracao\u2022dos\u2022Fluidos\u2022
785
786     -----
787     \n";

```

```

780     out << "#_"
781     << QString("Nome_Trecho").leftJustified(25, ' ')
782     << QString("Prof._Inicial_(ft)").leftJustified(25, ' ')
783     << QString("Prof._Final_(ft)").leftJustified(25, ' ')
784     << QString("Diam._externo_(in)").leftJustified(25, ' ')
785     << QString("Diam._interno_(in)").leftJustified(25, ' ')
786     << QString("Coef._Poisson").leftJustified(25, ' ')
787     << QString("Coef._Exp._Term.(1/F)").leftJustified(25, ' ')
788     << QString("Mod._Elast.(psi)").leftJustified(25, ' ')
789     << QString("Peso/unid_(lb/ft)").leftJustified(25, ' ')
790     << QString("Nome_fluido").leftJustified(25, ' ')
791     << QString("Densidade_(lbf/gal)").leftJustified(25, ' ')
792     << QString("Viscosidade_(cP)").leftJustified(25, ' ')
793     << "\n";
794
795     int linhas = ui->tblFluidos->rowCount();
796     for (int i = 0; i < linhas; ++i) {
797         out << "##" // recuo
798         << ui->tblTrechos->item(i, 0)->text().leftJustified(25,
799                                         ' ')
800         << ui->tblTrechos->item(i, 1)->text().leftJustified(25,
801                                         ' ')
802         << ui->tblTrechos->item(i, 2)->text().leftJustified(25,
803                                         ' ')
804         << ui->tblTrechos->item(i, 3)->text().leftJustified(25,
805                                         ' ')
806         << ui->tblTrechos->item(i, 4)->text().leftJustified(25,
807                                         ' ')
808         << ui->tblTrechos->item(i, 5)->text().leftJustified(25,
809                                         ' ')
810         << ui->tblTrechos->item(i, 6)->text().leftJustified(25,
811                                         ' ')
812         << ui->tblTrechos->item(i, 7)->text().leftJustified(25,
813                                         ' ')
814         << ui->tblTrechos->item(i, 8)->text().leftJustified(25,
815                                         ' ')
816         << ui->tblFluidos->item(i, 0)->text().leftJustified(25,
817                                         ' ')
818         << ui->tblFluidos->item(i, 1)->text().leftJustified(25,
819                                         ' ')
820         << ui->tblFluidos->item(i, 2)->text().leftJustified(25,
821                                         ' ')

```

```

810             << "\n";
811     }
812
813     arquivo.close();
814
815     // Atualiza o caminho salvo apenas se for novo
816     if (CaminhoArquivo().isEmpty())
817     {
818         CaminhoArquivo(caminho);
819         NomeArquivo(QFileInfo(caminho).fileName());
820     }
821
822     QMessageBox::information(this, "Salvo", "Arquivo salvo com
823     sucesso!");
824
825 void CSimuladorPerdaTubulacao::on_actionSalvar_triggered()
826 {
827     SalvarArquivo(false); // salvar direto
828 }
829
830 void CSimuladorPerdaTubulacao::on_actionSalvar_como_triggered()
831 {
832     SalvarArquivo(true); // for ar abrir QFileDialog
833 }
834
835 // essa funcao edita os dados do fluido e do trecho com base na
836 // linha da tabela de fluidos
837 void CSimuladorPerdaTubulacao::EditarLinhaTabela(int row)
838 {
839     // garante que o ndice da linha v lido
840     if (row < 0 || row >= poco->Trechos().size()) {
841         return;
842     }
843
844     // obtem o trecho e fluido da mesma linha
845     CTrechoPoco* trecho = poco->Trechos().at(row);
846     CFluido* fluido = trecho->Fluido();
847
848     if (!fluido) return;
849
850     trecho->Nome(ui->tblTrechos->item(row, 0)->text().toString()

```

```

    );
850     trecho->ProfundidadeInicial(ui->tblTrechos->item(row, 1)->text
851         () .toDouble());
851     trecho->ProfundidadeFinal(ui->tblTrechos->item(row, 2)->text() .
852         toDouble());
852     trecho->DiametroExterno(ui->tblTrechos->item(row, 3)->text() .
853         toDouble());
853     trecho->DiametroInterno(ui->tblTrechos->item(row, 4)->text() .
854         toDouble());
854     trecho->CoeficientePoisson(ui->tblTrechos->item(row, 5)->text() .
855         toDouble());
855     trecho->CoeficienteExpancaoTermica(ui->tblTrechos->item(row, 6)
856         ->text() .toDouble());
856     trecho->ModuloEslasticidade(ui->tblTrechos->item(row, 7)->text
857         () .toDouble());
857     trecho->PesoUnidade(ui->tblTrechos->item(row, 8)->text() .
858         toDouble());

858
859     fluido->Nome(ui->tblFluidos->item(row, 0)->text() .toStdString()
860         );
860     fluido->Densidade(ui->tblFluidos->item(row, 1)->text() .toDouble
861         ());
861     fluido->Viscosidade(ui->tblFluidos->item(row, 2)->text() .
862         toDouble());

862
863     // atualiza valores globais do simulador
864     AtualizarDados();
865
866     ui->statusbar->showMessage("Fluido e profundidades atualizados com sucesso!");
867 }

```

---

Fonte: produzido pelo autor.

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

Listing 7.6: Arquivo de implementação da classe CObjetoPoco

```

1 #ifndef COBJETOPOCO_H
2 #define COBJETOPOCO_H
3
4 #include <vector>
5 #include <memory>
6 #include "CTrechoTubulacao.h"
7

```

```

8 // Classe CObjetoPoco representa o objeto principal que armazena os
   dados do po e permite a execu o de c lculos e
   simula es

9
10 class CObjetoPoco {
11 protected:
12     std::string nomePoco;
13     double profundidadeFinal = 0.0;
14     double profundidadeOcupada = 0.0;
15     double pressaoSuperficie = 0.0;
16     double diametroPoco = 0.0;
17     double diametroRevestimentoOD = 0.0;
18     double diametroRevestimentoID = 0.0;
19     double vazao = 0.0;
20     std::vector<std::unique_ptr<CTrechoPoco>> trechos;

21
22     double pressaoSuperficieFim = 0.0;
23     double temperaturaTopoInicial = 0.0;
24     double temperaturaFundoInicial = 0.0;
25     double temperaturaTopoFinal = 0.0;
26     double temperaturaFundoFinal = 0.0;
27     double packer = true;

28
29 public:
30     // Construtor e destrutor padr o
31     CObjetoPoco() = default;
32     ~CObjetoPoco() = default;

33
34     // Evita c pia (pra evitar duplica o desnecess ria 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     // M todos de cria o est ticos, separando claramente os
       dados exigidos por cada m dulo
43     static CObjetoPoco CriarParaModulo01(std::string Nome, double
       Profund, double PressaoSup, double D, double OD, double ID,
       double q);

```

```

44     static CObjetoPoco CriarParaModulo02(std::string Nome, double
45         Profund, double diamPoco, double PressaoSup, double
46         PressaoSupFinal, double TempTopoInicial, double
47         TempFundoInicial, double TempTopoFinal, double
48         TempFundoFinal, bool haPacker);
49
50     // Getters para acessar os atributos de forma segura
51     std::string NomePoco() const { return nomePoco; }
52     double ProfundidadeTotal() const { return profundidadeFinal; }
53     double ProfundidadeOcupada() const { return profundidadeOcupada
54         ; }
55     double PressaoSuperficie() const { return pressaoSuperficie; }
56     double PressaoSuperficieFim() const { return
57         pressaoSuperficieFim; }
58     double DiametroPoco() const { return diametroPoco; }
59     double DiametroRevestimentoOD() const { return
60         diametroRevestimentoOD; }
61     double DiametroRevestimentoID() const { return
62         diametroRevestimentoID; }
63     double Vazao() const { return vazao; }
64     double TemperaturaTopoInicial() const { return
65         temperaturaTopoInicial; }
66     double TemperaturaFundoInicial() const { return
67         temperaturaFundoInicial; }
68     double TemperaturaTopoFinal() const { return
69         temperaturaTopoFinal; }
70     double TemperaturaFundoFinal() const { return
71         temperaturaFundoFinal; }
72     bool Packer() const { return packer; }
73
74     // Retorna os trechos adicionados ao poço (em forma de
75     // ponteiros brutos)
76     std::vector<CTrechoPoco*> Trechos() const;
77
78     // Setters permitem modificar os atributos do poço
79     void NomePoco(std::string Nome) { nomePoco = Nome; }
80     void ProfundidadeTotal(double Profundidade) { profundidadeFinal
81         = Profundidade; }
82     void ProfundidadeOcupada(double Profundidade) {
83         profundidadeOcupada = Profundidade; }
84     void PressaoSuperficie(double PressaoSuperior) {
85         pressaoSuperficie = PressaoSuperior; }

```

```

70     void PressaoSuperficieFim(double PressaoSuperior) {
71         pressaoSuperficieFim = PressaoSuperior; }
72     void DiametroPoco(double D) { diametroPoco = D; }
73     void DiametroRevestimentoOD(double DiametroExterno) {
74         diametroRevestimentoOD = DiametroExterno; }
75     void DiametroRevestimentoID(double DiametroInterno) {
76         diametroRevestimentoID = DiametroInterno; }
77     void Vazao(double q) { vazao = q; }
78     void TemperaturaTopoInicial(double temperatura) {
79         temperaturaTopoInicial = temperatura; }
80     void TemperaturaFundoInicial(double temperatura) {
81         temperaturaFundoInicial = temperatura; }
82     void TemperaturaTopoFinal(double temperatura) {
83         temperaturaTopoFinal = temperatura; }
84     void TemperaturaFundoFinal(double temperatura) {
85         temperaturaFundoFinal = temperatura; }
86     void Packer(bool haPacker) { packer = haPacker; }

87
88 // M etodos de c lculo
89
90     double PressaoHidroestaticaTotal() const;
91     double PressaoHidroestaticaNoPonto(double profundidade) const;
92     double DensidadeEfetivaTotal() const;
93     double ViscosidadeEfetivaTotal() const;
94     bool VerificarPreenchimentoColuna();
95     double Carga(double profundidade, bool inicio) const;
96     double DeltaLTemperatura(double profundidade) const;
97     double DeltaLEfeitoBalao(double profundidade) const;
98     double VariacaoCargaDevidoCrossover(double profundidade, bool
99         deCimaParaBaixo, bool inicio) const;
100    double VariacaoCargaEfeitoPistao(double profundidade, double ID
101        , double OD) const;
102    double DeltaLPistaoPacker(double profundidade) const;
103    double DeltaLPistaoCrossover(double profundidade) const;
104    double DeltaLForcaRestauradora(double profundidade) const;
105    double CargaInjecao(double profundidade) const;
106    double TemperaturaNoPonto(double profundidade, double T_topo,
107        double T_Fundo) const;

108
109    bool AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco>
110        trechoParaAdicionar); // Fun o para adicionar um novo
111        trecho ao p o o

```

```

100     void RemoverFluidoPoco(const std::string& nomeFluido); // Remove trecho do po o com base no nome do fluido
101     void RemoverTrechoPoco(const std::string& nomeTrecho);
102
103     // M todos que retornam dados para gr ficos
104     std::pair<std::vector<double>, std::vector<double>>
105         PlotarProfundidadePorPressao();
106     std::pair<std::vector<double>, std::vector<double>>
107         PlotarProfundidadePorPressaoMedia();
108 };
109
110 #endif

```

---

Fonte: produzido pelo autor.

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

Listing 7.7: Arquivo de implementação da classe CObjetoPoco

```

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

```

```

        ;
20     objetoPoco.diametroRevestimentoID = diametroRevestimentoInterno
        ;
21     objetoPoco.vazao = vazaoDoPoco;
22
23     return objetoPoco;
24 }
25
26
27 CObjetoPoco CObjetoPoco::CriarParaModulo02(std::string nomeDoPoco,
28     double profundidadeFinalDoPoco, double diamPoco, double
29     pressaoNaSuperficie, double pressaoNaSuperficieFim,
30     double
31     temperaturaTopoInicial
32     , double
33     temperaturaFundoInicial
34     ,
35     double
36     temperaturaTopoFinal,
37     double
38     temperaturaFundoFinal
39     , bool haPacker) {
40
41     CObjetoPoco objetoPoco;
42
43     objetoPoco.nomePoco = nomeDoPoco;
44     objetoPoco.profundidadeFinal = profundidadeFinalDoPoco;
45     objetoPoco.diametroPoco = diamPoco;
46     objetoPoco.pressaoSuperficie = pressaoNaSuperficie;
47     objetoPoco.pressaoSuperficieFim = pressaoNaSuperficieFim;
48     objetoPoco.temperaturaTopoInicial = temperaturaTopoInicial;
49     objetoPoco.temperaturaFundoInicial = temperaturaFundoInicial;
50     objetoPoco.temperaturaTopoFinal = temperaturaTopoFinal;
51     objetoPoco.temperaturaFundoFinal = temperaturaFundoFinal;
52     objetoPoco.packer = haPacker;
53
54     return objetoPoco;
55 }
56
57
58 std::vector<CTrechoPoco*> CObjetoPoco::Treichos() const {
59     std::vector<CTrechoPoco*> vetorDePonteirosParaTreichos;
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1096
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2189
2190
2191
2192
2193
2194
2195
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215

```

```

50     // percorre todos os trechos armazenados no po o
51     for (const auto& trechoUnico : trechos) {
52         // adiciona o ponteiro cru (n o nico ) de cada trecho ao
53         // vetor de sa da
54         vetorDePonteirosParaTrechos.push_back(trechoUnico.get());
55     }
56
57     // retorna o vetor contendo os ponteiros dos trechos
58     return vetorDePonteirosParaTrechos;
59 }
60
61 bool CObjetoPoco::AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco>
62                                         trechoParaAdicionar) {
63     // calcula o comprimento do trecho com base nas profundidades
64     // inicial e final
65     double comprimentoDoTrecho = trechoParaAdicionar->
66         ProfundidadeFinal() - trechoParaAdicionar->
67         ProfundidadeInicial();
68
69     // move o trecho para dentro do vetor principal do po o
70     trechos.push_back(std::move(trechoParaAdicionar));
71
72
73
74 void CObjetoPoco::RemoverFluidoPoco(const std::string& nomeFluido)
75 {
76     for (auto it = trechos.begin(); it != trechos.end();) {
77         if ((*it)->Fluido()->Nome() == nomeFluido) {
78             double comprimento = (*it)->ProfundidadeFinal() - (*it)
79                         ->ProfundidadeInicial();
80             it = trechos.erase(it);
81             profundidadeOcupada -= comprimento;
82         } else {
83             ++it;
84         }
85     }

```

```

84 }
85
86 void CObjetoPoco::RemoverTrechoPoco(const std::string& nomeTrecho)
87 {
88     for (auto it = trechos.begin(); it != trechos.end();) {
89         if ((*it)->Nome() == nomeTrecho) {
90             double comprimento = (*it)->ProfundidadeFinal() - (*it)
91                         ->ProfundidadeInicial();
92             it = trechos.erase(it);
93             profundidadeOcupada -= comprimento;
94         } else {
95             ++it;
96         }
97     }
98 }
99
100
101 // soma a pressao hidroestatica de todos os trechos do po o
102 for (const auto& trechoAtual : trechos) {
103     pressaoTotalHidroestatica += trechoAtual->
104         PressaoHidroestatica();
105 }
106
107 // adiciona a pressao da superficie para obter a pressao total
108 return pressaoTotalHidroestatica + pressaoSuperficie;
109
110
111 double CObjetoPoco::PressaoHidroestaticaNoPonto(double
112     profundidadeDesejada) const {
113     double pressaoAcumulada = pressaoSuperficie;
114     double profundidadeJaCalculada = 0.0;
115
116     // percorre cada trecho verificando se ele contem a
117     // profundidade desejada
118     for (const auto& trechoAtual : trechos) {
119         double comprimentoDoTrecho = trechoAtual->ProfundidadeFinal
120             () - trechoAtual->ProfundidadeInicial();
121
122         // se a profundidade estiver dentro do trecho atual
123         if (profundidadeDesejada <= profundidadeJaCalculada +
124             comprimentoDoTrecho && profundidadeDesejada <
125                 profundidadeJaCalculada + comprimentoDoTrecho)
126             profundidadeJaCalculada += comprimentoDoTrecho;
127     }
128
129     return profundidadeJaCalculada;
130 }

```

```

        comprimentoDoTrecho) {
120    double profundidadeDentroDoTrecho =
121        profundidadeDesejada - profundidadeJaCalculada;
122
123    // adiciona somente a parte proporcional da pressao no
124    // trecho
125    pressaoAcumulada += trechoAtual->PressaoHidroestatica(
126        profundidadeDentroDoTrecho);
127
128    break;
129} else {
130    // adiciona a pressao total do trecho completo
131    pressaoAcumulada += trechoAtual->PressaoHidroestatica()
132        ;
133    profundidadeJaCalculada += comprimentoDoTrecho;
134}
135
136    return pressaoAcumulada;
137}
138
139bool CObjetoPoco::VerificarPreenchimentoColuna() {
140    double profundidadeNaoPreenchida = profundidadeFinal -
141        profundidadeOcupada;
142
143    if (profundidadeNaoPreenchida > 0) {
144        std::cout << "Uma coluna de " << profundidadeNaoPreenchida
145        << " litros fluido precisa ser adicionada!" << std::endl;
146        return false; // coluna incompleta
147    } else {
148        std::cout << "A coluna de fluidos equivale a profundidade "
149        << profundidadeTotal << " litros!" << std::endl;
150        return true; // coluna completamente preenchida
151    }
152}
153
154double CObjetoPoco::DensidadeEfetivaTotal() const {
155    double somaDensidadePonderada = 0.0;
156    double comprimentoTotalDaColuna = 0.0;
157
158    for (const auto& trechoAtual : trechos) {
159        double comprimentoTrecho = trechoAtual->ProfundidadeFinal()

```

```

        - trechoAtual->ProfundidadeInicial();

154
155    // multiplica a densidade equivalente pelo comprimento do
156    // trecho para ponderar
157    somaDensidadePonderada += trechoAtual->DensidadeEquivalente()
158    () * comprimentoTrecho;
159    comprimentoTotalDaColuna += comprimentoTrecho;
160}
161}

162

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

176 std::pair<std::vector<double>, std::vector<double>> CObjetoPoco::
177 PlotarProfundidadePorPressaoMedia() {
178     std::vector<double> vetorDeProfundidades;
179     std::vector<double> vetorDePressoes;
180
181     // percorre cada metro ao longo da profundidade total do po o
182     for (double profundidadeAtual = 0.0; profundidadeAtual <=
183         ProfundidadeTotal(); profundidadeAtual += 1.0) {
184         vetorDeProfundidades.push_back(profundidadeAtual);
185
186         // calcula a pressao no ponto atual usando a funcao
187         // apropriada
188         double pressaoNoPonto = PressaoHidroestaticaNoPonto(
189             profundidadeAtual);
190         vetorDePressoes.push_back(pressaoNoPonto);
191     }

```

```

187
188     // retorna as duas listas para serem usadas na geracao do
189     // grafico
190     return std::make_pair(vetorDeProfundidades, vetorDePressoes);
191 }
192 double CObjetoPoco::Carga(double profundidade, bool inicio) const {
193     double cargaTotal = 0.0;
194     double pi = 3.141592653589793;
195
196     if (trechos.empty())
197         return 0.0;
198
199     // 1. Pressao no fundo do poco (base)
200     double profundidadeBase = 0.0;
201     double OD_base = 0.0;
202     double ID_base = 0.0;
203
204     for (const auto& trecho : trechos) {
205         if (trecho->ProfundidadeFinal() > profundidadeBase) {
206             profundidadeBase = trecho->ProfundidadeFinal();
207             OD_base = trecho->DiametroExterno();
208             ID_base = trecho->DiametroInterno();
209         }
210     }
211
212     // 2. Carga por pressao no fundo (negativa)
213     double pressaoBase = PressaoHidroestaticaNoPonto(
214         profundidadeBase);
215     if (inicio == false){
216         pressaoBase = PressaoHidroestaticaNoPonto(profundidadeBase)
217             + PressaoSuperficieFim();
218     }
219
220     double areaBase = (pi / 4.0) * (OD_base * OD_base - ID_base *
221         ID_base);
222     double cargaPressao = -1.0 * pressaoBase * areaBase;
223     cargaTotal += cargaPressao;
224
225     // 3. Soma pesos de trechos abaixo da profundidade
226     for (const auto& trecho : trechos) {
227         double z_i = trecho->ProfundidadeInicial();

```

```

225     double z_f = trecho->ProfundidadeFinal();
226     double pesoUnit = trecho->PesoUnidade();
227
228     // Caso 1: trecho totalmente abaixo da profundidade
229     if (z_i >= profundidade) {
230         double L_total = z_f - z_i;
231         double cargaPeso = L_total * pesoUnit;
232         cargaTotal += cargaPeso;
233     }
234
235     // Caso 2: profundidade dentro do trecho
236     else if (profundidade > z_i && profundidade < z_f) {
237         double L_parcial = z_f - profundidade;
238         double cargaPeso = L_parcial * pesoUnit;
239         cargaTotal += cargaPeso;
240     }
241
242 }
243
244
245 // 4. considerando as Cargas por efeito pisto
246 if (inicio){
247     cargaTotal += VariacaoCargaDevidoCrossover(profundidade ,
248             false , true);
249 }
250 else{
251     cargaTotal += VariacaoCargaDevidoCrossover(profundidade ,
252             false , false);
253 }
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,
271       TemperaturaTopoInicial(),
272       TemperaturaFundoInicial()) +
273     TemperaturaNoPonto(z_f,
274       TemperaturaTopoInicial(),
275       TemperaturaFundoInicial())
276   ) / 2.0;
277
278
279   double tMedioFinal = (
280     TemperaturaNoPonto(z_i,
281       TemperaturaTopoFinal(),
282       TemperaturaFundoFinal()) +
283     TemperaturaNoPonto(z_f,
284       TemperaturaTopoFinal(),
285       TemperaturaFundoFinal())
286   ) / 2.0;
287
288   // Variacao de temperatura final - inicial
289   double deltaT = tMedioFinal - tMedioInicial;
290
291   // Caso raro onde nao houve variacao de temperatura entre
292   // as duas condicoes
293   // A gente for a um deltaT usando a temperatura no ponto
294   // zero como referencia
295   if (deltaT == 0) {
296     double media = (
297       TemperaturaNoPonto(z_i,
298         TemperaturaTopoFinal(),
299         TemperaturaFundoFinal()) +
300       TemperaturaNoPonto(z_f,
301         TemperaturaTopoFinal(),
302         TemperaturaFundoFinal())
303     ) / 2.0;
304
305     deltaT = TemperaturaNoPonto(0, TemperaturaTopoFinal(),
306       TemperaturaFundoFinal()) - media;

```

```

291     }
292
293     double L = 0.0;
294
295     // Se a profundidade estiver dentro do trecho, calcula o
296     // 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,
301     // considera L total
302     else if (z_f <= profundidade) {
303         L = z_f - z_i;
304     }
305     // Soma o deltaL do trecho na variavel acumuladora
306     deltaLTotal += ct * L * deltaT;
307 }
308
309 }
310
311
312
313
314 double CObjetoPoco::TemperaturaNoPonto(double profundidade, double
315     T_topo, double T_Fundo) const {
316
317     double inclinacao = (T_Fundo - T_topo) / ProfundidadeOcupada();
318     double temperatura = T_topo + inclinacao * profundidade;
319     return temperatura;
320 }
321
322 double CObjetoPoco::VariacaoCargaDevidoCrossover(double
323     profundidade, bool deCimaParaBaixo, bool inicio) const {
324
325     double pi = 3.141592653589793;
326     double variacao_total = 0.0;
327
328     // tolerancia para comparacoes de ponto flutuante
329     constexpr double tolerancia = 1e-5;
330
331 }
```

```

329     if (trechos.size() < 2)
330         return 0.0;
331
332     // percorre todas as interfaces entre os trechos consecutivos
333     for (size_t i = 1; i < trechos.size(); ++i) {
334         const auto& trecho_acima = trechos[i - 1];
335         const auto& trecho_abixo = trechos[i];
336
337         double z_interface = trecho_abixo->ProfundidadeInicial();
338
339         // define se a interface deve ser considerada com base na
340         // direcao da analise
341         bool considerar = false;
342         if (!deCimaParaBaixo && z_interface >= profundidade -
343             tolerancia)
344             considerar = true;
345         if ( deCimaParaBaixo && z_interface <= profundidade +
346             tolerancia)
347             considerar = true;
348
349         if (!considerar)
350             continue;
351
352         // coleta diametros internos e externos dos dois trechos
353         double ID_acima = trecho_acima->DiametroInterno();
354         double ID_abixo = trecho_abixo->DiametroInterno();
355         double OD_acima = trecho_acima->DiametroExterno();
356         double OD_abixo = trecho_abixo->DiametroExterno();
357
358         // se nao houver mudanca de diametro, nao ha efeito pistao
359         if (std::abs(ID_acima - ID_abixo) < 1e-6 &&
360             std::abs(OD_acima - OD_abixo) < 1e-6) {
361             continue;
362         }
363
364         // calcula diferenca de area interna e externa
365         double Ai = (pi / 4.0) * (ID_acima * ID_acima - ID_abixo *
366             ID_abixo);
367         double Ao = (pi / 4.0) * (OD_acima * OD_acima - OD_abixo *
368             OD_abixo);
369
370         // 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 =
373             // pressao hidrostatica
374             carga_crossover = pressao * (Ai - Ao);
375         } else {
376             // no fim da coluna: pressao interna = pressao
377             // hidrostatica + pressao superficial final
378             carga_crossover = (pressao - (pressao +
379             PressaoSuperficieFim())) * Ai;
380         }
381     } else {
382         // sem packer: pressao interna e externa sao iguais
383         carga_crossover = pressao * (Ai - Ao);
384     }
385
386     variacao_total += carga_crossover;
387 }
388
389
390 #include <QDebug>
391 double CObjetoPoco::VariacaoCargaEfeitoPistao(double profundidade,
392                                                 double ID, double OD) const {
393     const double pi = 3.141592653589793;
394
395     // rea da parede interna do tubo
396     double Ain = (pi / 4.0) * (OD * OD - ID * ID);
397
398     double Aout = Ain;
399
400     // Press es internas
401     double Pin_inicio = PressaoHidroestaticaNoPonto(profundidade) +
402     PressaoSuperficie();

```

```

403     double Pin_fim = PressaoHidroestaticaNoPonto(profundidade) +
404         PressaoSuperficieFim();
405
406     // Press es 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 mudanca de diametro externo entre os
434     // trechos
435     bool possuiDiferencaOD = false;
436     double primeiroOD = trechos[0]->DiametroExterno();
437     for (const auto& trecho : trechos) {
438         if (std::abs(trecho->DiametroExterno() - primeiroOD) > 1e
439             -6) {
440             possuiDiferencaOD = true;
441             break;
442         }
443     }

```

```

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->ModuloEslasticidade();
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 mudanca de OD: usa ID e OD do trecho
468             CargaPistao = VariacaoCargaEfeitoPistao(z_i, ID, OD);
469         } else {
470             // todos os trechos iguais: usa ID e o diametro total
471             // do poco
472             CargaPistao = VariacaoCargaEfeitoPistao(z_i, ID,
473                                         DiametroPoco());
474
475         double deltaL_trecho = (CargaPistao * L) / (E * area_anular
476             );
477         deltaL_total += deltaL_trecho;
478     }
479
480     return deltaL_total;

```

```

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 CargaPistão = VariacaoCargaEfeitoPistão(z_i, 0, ID);
514         double deltaL_trecho = 2 * v * (CargaPistão * (L)) / (E *
515             area_anular); // o x12      uma unidade de conversão
516             // para in
517             deltaL_total += deltaL_trecho;
518     }
519
520     return deltaL_total;
521 }

```

```

521
522 double CObjetoPoco::DeltaLPistaoCrossover(double profundidade)
523     const {
524         double pi = 3.141592653589793;
525         double deltaL_total = 0.0;
526
527         for (size_t i = 1; i < trechos.size(); ++i) {
528             const auto& trechoAcima = trechos[i - 1];
529             const auto& trechoAbaixo = trechos[i];
530
531             // Profundidade onde ocorre a mudanca de trecho
532             double z_interface = trechoAbaixo->ProfundidadeInicial();
533
534             // Ignora se a interface estiver abaixo da profundidade
535             // analisada
536             if (z_interface > profundidade)
537                 continue;
538
539             // Coleta os diametros externos para comparar se ha mudanca
540             double OD_acima = trechoAcima->DiametroExterno();
541             double ID_acima = trechoAcima->DiametroInterno();
542             double OD_abaixo = trechoAbaixo->DiametroExterno();
543
544             // Se os diametros forem quase iguais, nao ha efeito pistao
545             // relevante
546             if (std::abs(OD_acima - OD_abaixo) < 1e-6)
547                 continue;
548
549             // Area da secao transversal da coluna acima (quem vai
550             // deformar)
551             double area_secao = (pi / 4.0) * (OD_acima * OD_acima -
552                                         ID_acima * ID_acima);
553
554             // Obtem o modulo de elasticidade medio entre os dois
555             // trechos
556             double E1 = trechoAcima->ModuloElasticidade();
557             double E2 = trechoAbaixo->ModuloElasticidade();
558             double E_medio = (E1 + E2) / 2.0;
559
560             if (E_medio <= 0.0 || area_secao <= 0.0)
561                 continue;
562
563             if (deltaL_total <= 0.0)
564                 continue;
565
566             if (deltaL_total >= 0.0)
567                 continue;
568
569             if (deltaL_total <= 0.0)
570                 continue;
571
572             if (deltaL_total >= 0.0)
573                 continue;
574
575             if (deltaL_total <= 0.0)
576                 continue;
577
578             if (deltaL_total >= 0.0)
579                 continue;
580
581             if (deltaL_total <= 0.0)
582                 continue;
583
584             if (deltaL_total >= 0.0)
585                 continue;
586
587             if (deltaL_total <= 0.0)
588                 continue;
589
590             if (deltaL_total >= 0.0)
591                 continue;
592
593             if (deltaL_total <= 0.0)
594                 continue;
595
596             if (deltaL_total >= 0.0)
597                 continue;
598
599             if (deltaL_total <= 0.0)
600                 continue;
601
602             if (deltaL_total >= 0.0)
603                 continue;
604
605             if (deltaL_total <= 0.0)
606                 continue;
607
608             if (deltaL_total >= 0.0)
609                 continue;
610
611             if (deltaL_total <= 0.0)
612                 continue;
613
614             if (deltaL_total >= 0.0)
615                 continue;
616
617             if (deltaL_total <= 0.0)
618                 continue;
619
620             if (deltaL_total >= 0.0)
621                 continue;
622
623             if (deltaL_total <= 0.0)
624                 continue;
625
626             if (deltaL_total >= 0.0)
627                 continue;
628
629             if (deltaL_total <= 0.0)
630                 continue;
631
632             if (deltaL_total >= 0.0)
633                 continue;
634
635             if (deltaL_total <= 0.0)
636                 continue;
637
638             if (deltaL_total >= 0.0)
639                 continue;
640
641             if (deltaL_total <= 0.0)
642                 continue;
643
644             if (deltaL_total >= 0.0)
645                 continue;
646
647             if (deltaL_total <= 0.0)
648                 continue;
649
650             if (deltaL_total >= 0.0)
651                 continue;
652
653             if (deltaL_total <= 0.0)
654                 continue;
655
656             if (deltaL_total >= 0.0)
657                 continue;
658
659             if (deltaL_total <= 0.0)
660                 continue;
661
662             if (deltaL_total >= 0.0)
663                 continue;
664
665             if (deltaL_total <= 0.0)
666                 continue;
667
668             if (deltaL_total >= 0.0)
669                 continue;
670
671             if (deltaL_total <= 0.0)
672                 continue;
673
674             if (deltaL_total >= 0.0)
675                 continue;
676
677             if (deltaL_total <= 0.0)
678                 continue;
679
680             if (deltaL_total >= 0.0)
681                 continue;
682
683             if (deltaL_total <= 0.0)
684                 continue;
685
686             if (deltaL_total >= 0.0)
687                 continue;
688
689             if (deltaL_total <= 0.0)
690                 continue;
691
692             if (deltaL_total >= 0.0)
693                 continue;
694
695             if (deltaL_total <= 0.0)
696                 continue;
697
698             if (deltaL_total >= 0.0)
699                 continue;
700
701             if (deltaL_total <= 0.0)
702                 continue;
703
704             if (deltaL_total >= 0.0)
705                 continue;
706
707             if (deltaL_total <= 0.0)
708                 continue;
709
710             if (deltaL_total >= 0.0)
711                 continue;
712
713             if (deltaL_total <= 0.0)
714                 continue;
715
716             if (deltaL_total >= 0.0)
717                 continue;
718
719             if (deltaL_total <= 0.0)
720                 continue;
721
722             if (deltaL_total >= 0.0)
723                 continue;
724
725             if (deltaL_total <= 0.0)
726                 continue;
727
728             if (deltaL_total >= 0.0)
729                 continue;
730
731             if (deltaL_total <= 0.0)
732                 continue;
733
734             if (deltaL_total >= 0.0)
735                 continue;
736
737             if (deltaL_total <= 0.0)
738                 continue;
739
740             if (deltaL_total >= 0.0)
741                 continue;
742
743             if (deltaL_total <= 0.0)
744                 continue;
745
746             if (deltaL_total >= 0.0)
747                 continue;
748
749             if (deltaL_total <= 0.0)
750                 continue;
751
752             if (deltaL_total >= 0.0)
753                 continue;
754
755             if (deltaL_total <= 0.0)
756                 continue;
757
758             if (deltaL_total >= 0.0)
759                 continue;
760
761             if (deltaL_total <= 0.0)
762                 continue;
763
764             if (deltaL_total >= 0.0)
765                 continue;
766
767             if (deltaL_total <= 0.0)
768                 continue;
769
770             if (deltaL_total >= 0.0)
771                 continue;
772
773             if (deltaL_total <= 0.0)
774                 continue;
775
776             if (deltaL_total >= 0.0)
777                 continue;
778
779             if (deltaL_total <= 0.0)
780                 continue;
781
782             if (deltaL_total >= 0.0)
783                 continue;
784
785             if (deltaL_total <= 0.0)
786                 continue;
787
788             if (deltaL_total >= 0.0)
789                 continue;
790
791             if (deltaL_total <= 0.0)
792                 continue;
793
794             if (deltaL_total >= 0.0)
795                 continue;
796
797             if (deltaL_total <= 0.0)
798                 continue;
799
800             if (deltaL_total >= 0.0)
801                 continue;
802
803             if (deltaL_total <= 0.0)
804                 continue;
805
806             if (deltaL_total >= 0.0)
807                 continue;
808
809             if (deltaL_total <= 0.0)
810                 continue;
811
812             if (deltaL_total >= 0.0)
813                 continue;
814
815             if (deltaL_total <= 0.0)
816                 continue;
817
818             if (deltaL_total >= 0.0)
819                 continue;
820
821             if (deltaL_total <= 0.0)
822                 continue;
823
824             if (deltaL_total >= 0.0)
825                 continue;
826
827             if (deltaL_total <= 0.0)
828                 continue;
829
830             if (deltaL_total >= 0.0)
831                 continue;
832
833             if (deltaL_total <= 0.0)
834                 continue;
835
836             if (deltaL_total >= 0.0)
837                 continue;
838
839             if (deltaL_total <= 0.0)
840                 continue;
841
842             if (deltaL_total >= 0.0)
843                 continue;
844
845             if (deltaL_total <= 0.0)
846                 continue;
847
848             if (deltaL_total >= 0.0)
849                 continue;
850
851             if (deltaL_total <= 0.0)
852                 continue;
853
854             if (deltaL_total >= 0.0)
855                 continue;
856
857             if (deltaL_total <= 0.0)
858                 continue;
859
860             if (deltaL_total >= 0.0)
861                 continue;
862
863             if (deltaL_total <= 0.0)
864                 continue;
865
866             if (deltaL_total >= 0.0)
867                 continue;
868
869             if (deltaL_total <= 0.0)
870                 continue;
871
872             if (deltaL_total >= 0.0)
873                 continue;
874
875             if (deltaL_total <= 0.0)
876                 continue;
877
878             if (deltaL_total >= 0.0)
879                 continue;
880
881             if (deltaL_total <= 0.0)
882                 continue;
883
884             if (deltaL_total >= 0.0)
885                 continue;
886
887             if (deltaL_total <= 0.0)
888                 continue;
889
890             if (deltaL_total >= 0.0)
891                 continue;
892
893             if (deltaL_total <= 0.0)
894                 continue;
895
896             if (deltaL_total >= 0.0)
897                 continue;
898
899             if (deltaL_total <= 0.0)
900                 continue;
901
902             if (deltaL_total >= 0.0)
903                 continue;
904
905             if (deltaL_total <= 0.0)
906                 continue;
907
908             if (deltaL_total >= 0.0)
909                 continue;
910
911             if (deltaL_total <= 0.0)
912                 continue;
913
914             if (deltaL_total >= 0.0)
915                 continue;
916
917             if (deltaL_total <= 0.0)
918                 continue;
919
920             if (deltaL_total >= 0.0)
921                 continue;
922
923             if (deltaL_total <= 0.0)
924                 continue;
925
926             if (deltaL_total >= 0.0)
927                 continue;
928
929             if (deltaL_total <= 0.0)
930                 continue;
931
932             if (deltaL_total >= 0.0)
933                 continue;
934
935             if (deltaL_total <= 0.0)
936                 continue;
937
938             if (deltaL_total >= 0.0)
939                 continue;
940
941             if (deltaL_total <= 0.0)
942                 continue;
943
944             if (deltaL_total >= 0.0)
945                 continue;
946
947             if (deltaL_total <= 0.0)
948                 continue;
949
950             if (deltaL_total >= 0.0)
951                 continue;
952
953             if (deltaL_total <= 0.0)
954                 continue;
955
956             if (deltaL_total >= 0.0)
957                 continue;
958
959             if (deltaL_total <= 0.0)
960                 continue;
961
962             if (deltaL_total >= 0.0)
963                 continue;
964
965             if (deltaL_total <= 0.0)
966                 continue;
967
968             if (deltaL_total >= 0.0)
969                 continue;
970
971             if (deltaL_total <= 0.0)
972                 continue;
973
974             if (deltaL_total >= 0.0)
975                 continue;
976
977             if (deltaL_total <= 0.0)
978                 continue;
979
980             if (deltaL_total >= 0.0)
981                 continue;
982
983             if (deltaL_total <= 0.0)
984                 continue;
985
986             if (deltaL_total >= 0.0)
987                 continue;
988
989             if (deltaL_total <= 0.0)
990                 continue;
991
992             if (deltaL_total >= 0.0)
993                 continue;
994
995             if (deltaL_total <= 0.0)
996                 continue;
997
998             if (deltaL_total >= 0.0)
999                 continue;
1000
1001             if (deltaL_total <= 0.0)
1002                 continue;
1003
1004             if (deltaL_total >= 0.0)
1005                 continue;
1006
1007             if (deltaL_total <= 0.0)
1008                 continue;
1009
1010             if (deltaL_total >= 0.0)
1011                 continue;
1012
1013             if (deltaL_total <= 0.0)
1014                 continue;
1015
1016             if (deltaL_total >= 0.0)
1017                 continue;
1018
1019             if (deltaL_total <= 0.0)
1020                 continue;
1021
1022             if (deltaL_total >= 0.0)
1023                 continue;
1024
1025             if (deltaL_total <= 0.0)
1026                 continue;
1027
1028             if (deltaL_total >= 0.0)
1029                 continue;
1030
1031             if (deltaL_total <= 0.0)
1032                 continue;
1033
1034             if (deltaL_total >= 0.0)
1035                 continue;
1036
1037             if (deltaL_total <= 0.0)
1038                 continue;
1039
1040             if (deltaL_total >= 0.0)
1041                 continue;
1042
1043             if (deltaL_total <= 0.0)
1044                 continue;
1045
1046             if (deltaL_total >= 0.0)
1047                 continue;
1048
1049             if (deltaL_total <= 0.0)
1050                 continue;
1051
1052             if (deltaL_total >= 0.0)
1053                 continue;
1054
1055             if (deltaL_total <= 0.0)
1056                 continue;
1057
1058             if (deltaL_total >= 0.0)
1059                 continue;
1060
1061             if (deltaL_total <= 0.0)
1062                 continue;
1063
1064             if (deltaL_total >= 0.0)
1065                 continue;
1066
1067             if (deltaL_total <= 0.0)
1068                 continue;
1069
1070             if (deltaL_total >= 0.0)
1071                 continue;
1072
1073             if (deltaL_total <= 0.0)
1074                 continue;
1075
1076             if (deltaL_total >= 0.0)
1077                 continue;
1078
1079             if (deltaL_total <= 0.0)
1080                 continue;
1081
1082             if (deltaL_total >= 0.0)
1083                 continue;
1084
1085             if (deltaL_total <= 0.0)
1086                 continue;
1087
1088             if (deltaL_total >= 0.0)
1089                 continue;
1090
1091             if (deltaL_total <= 0.0)
1092                 continue;
1093
1094             if (deltaL_total >= 0.0)
1095                 continue;
1096
1097             if (deltaL_total <= 0.0)
1098                 continue;
1099
1100             if (deltaL_total >= 0.0)
1101                 continue;
1102
1103             if (deltaL_total <= 0.0)
1104                 continue;
1105
1106             if (deltaL_total >= 0.0)
1107                 continue;
1108
1109             if (deltaL_total <= 0.0)
1110                 continue;
1111
1112             if (deltaL_total >= 0.0)
1113                 continue;
1114
1115             if (deltaL_total <= 0.0)
1116                 continue;
1117
1118             if (deltaL_total >= 0.0)
1119                 continue;
1120
1121             if (deltaL_total <= 0.0)
1122                 continue;
1123
1124             if (deltaL_total >= 0.0)
1125                 continue;
1126
1127             if (deltaL_total <= 0.0)
1128                 continue;
1129
1130             if (deltaL_total >= 0.0)
1131                 continue;
1132
1133             if (deltaL_total <= 0.0)
1134                 continue;
1135
1136             if (deltaL_total >= 0.0)
1137                 continue;
1138
1139             if (deltaL_total <= 0.0)
1140                 continue;
1141
1142             if (deltaL_total >= 0.0)
1143                 continue;
1144
1145             if (deltaL_total <= 0.0)
1146                 continue;
1147
1148             if (deltaL_total >= 0.0)
1149                 continue;
1150
1151             if (deltaL_total <= 0.0)
1152                 continue;
1153
1154             if (deltaL_total >= 0.0)
1155                 continue;
1156
1157             if (deltaL_total <= 0.0)
1158                 continue;
1159
1160             if (deltaL_total >= 0.0)
1161                 continue;
1162
1163             if (deltaL_total <= 0.0)
1164                 continue;
1165
1166             if (deltaL_total >= 0.0)
1167                 continue;
1168
1169             if (deltaL_total <= 0.0)
1170                 continue;
1171
1172             if (deltaL_total >= 0.0)
1173                 continue;
1174
1175             if (deltaL_total <= 0.0)
1176                 continue;
1177
1178             if (deltaL_total >= 0.0)
1179                 continue;
1180
1181             if (deltaL_total <= 0.0)
1182                 continue;
1183
1184             if (deltaL_total >= 0.0)
1185                 continue;
1186
1187             if (deltaL_total <= 0.0)
1188                 continue;
1189
1190             if (deltaL_total >= 0.0)
1191                 continue;
1192
1193             if (deltaL_total <= 0.0)
1194                 continue;
1195
1196             if (deltaL_total >= 0.0)
1197                 continue;
1198
1199             if (deltaL_total <= 0.0)
1200                 continue;
1201
1202             if (deltaL_total >= 0.0)
1203                 continue;
1204
1205             if (deltaL_total <= 0.0)
1206                 continue;
1207
1208             if (deltaL_total >= 0.0)
1209                 continue;
1210
1211             if (deltaL_total <= 0.0)
1212                 continue;
1213
1214             if (deltaL_total >= 0.0)
1215                 continue;
1216
1217             if (deltaL_total <= 0.0)
1218                 continue;
1219
1220             if (deltaL_total >= 0.0)
1221                 continue;
1222
1223             if (deltaL_total <= 0.0)
1224                 continue;
1225
1226             if (deltaL_total >= 0.0)
1227                 continue;
1228
1229             if (deltaL_total <= 0.0)
1230                 continue;
1231
1232             if (deltaL_total >= 0.0)
1233                 continue;
1234
1235             if (deltaL_total <= 0.0)
1236                 continue;
1237
1238             if (deltaL_total >= 0.0)
1239                 continue;
1240
1241             if (deltaL_total <= 0.0)
1242                 continue;
1243
1244             if (deltaL_total >= 0.0)
1245                 continue;
1246
1247             if (deltaL_total <= 0.0)
1248                 continue;
1249
1250             if (deltaL_total >= 0.0)
1251                 continue;
1252
1253             if (deltaL_total <= 0.0)
1254                 continue;
1255
1256             if (deltaL_total >= 0.0)
1257                 continue;
1258
1259             if (deltaL_total <= 0.0)
1260                 continue;
1261
1262             if (deltaL_total >= 0.0)
1263                 continue;
1264
1265             if (deltaL_total <= 0.0)
1266                 continue;
1267
1268             if (deltaL_total >= 0.0)
1269                 continue;
1270
1271             if (deltaL_total <= 0.0)
1272                 continue;
1273
1274             if (deltaL_total >= 0.0)
1275                 continue;
1276
1277             if (deltaL_total <= 0.0)
1278                 continue;
1279
1280             if (deltaL_total >= 0.0)
1281                 continue;
1282
1283             if (deltaL_total <= 0.0)
1284                 continue;
1285
1286             if (deltaL_total >= 0.0)
1287                 continue;
1288
1289             if (deltaL_total <= 0.0)
1290                 continue;
1291
1292             if (deltaL_total >= 0.0)
1293                 continue;
1294
1295             if (deltaL_total <= 0.0)
1296                 continue;
1297
1298             if (deltaL_total >= 0.0)
1299                 continue;
1300
1301             if (deltaL_total <= 0.0)
1302                 continue;
1303
1304             if (deltaL_total >= 0.0)
1305                 continue;
1306
1307             if (deltaL_total <= 0.0)
1308                 continue;
1309
1310             if (deltaL_total >= 0.0)
1311                 continue;
1312
1313             if (deltaL_total <= 0.0)
1314                 continue;
1315
1316             if (deltaL_total >= 0.0)
1317                 continue;
1318
1319             if (deltaL_total <= 0.0)
1320                 continue;
1321
1322             if (deltaL_total >= 0.0)
1323                 continue;
1324
1325             if (deltaL_total <= 0.0)
1326                 continue;
1327
1328             if (deltaL_total >= 0.0)
1329                 continue;
1330
1331             if (deltaL_total <= 0.0)
1332                 continue;
1333
1334             if (deltaL_total >= 0.0)
1335                 continue;
1336
1337             if (deltaL_total <= 0.0)
1338                 continue;
1339
1340             if (deltaL_total >= 0.0)
1341                 continue;
1342
1343             if (deltaL_total <= 0.0)
1344                 continue;
1345
1346             if (deltaL_total >= 0.0)
1347                 continue;
1348
1349             if (deltaL_total <= 0.0)
1350                 continue;
1351
1352             if (deltaL_total >= 0.0)
1353                 continue;
1354
1355             if (deltaL_total <= 0.0)
1356                 continue;
1357
1358             if (deltaL_total >= 0.0)
1359                 continue;
1360
1361             if (deltaL_total <= 0.0)
1362                 continue;
1363
1364             if (deltaL_total >= 0.0)
1365                 continue;
1366
1367             if (deltaL_total <= 0.0)
1368                 continue;
1369
1370             if (deltaL_total >= 0.0)
1371                 continue;
1372
1373             if (deltaL_total <= 0.0)
1374                 continue;
1375
1376             if (deltaL_total >= 0.0)
1377                 continue;
1378
1379             if (deltaL_total <= 0.0)
1380                 continue;
1381
1382             if (deltaL_total >= 0.0)
1383                 continue;
1384
1385             if (deltaL_total <= 0.0)
1386                 continue;
1387
1388             if (deltaL_total >= 0.0)
1389                 continue;
1390
1391             if (deltaL_total <= 0.0)
1392                 continue;
1393
1394             if (deltaL_total >= 0.0)
1395                 continue;
1396
1397             if (deltaL_total <= 0.0)
1398                 continue;
1399
1400             if (deltaL_total >= 0.0)
1401                 continue;
1402
1403             if (deltaL_total <= 0.0)
1404                 continue;
1405
1406             if (deltaL_total >= 0.0)
1407                 continue;
1408
1409             if (deltaL_total <= 0.0)
1410                 continue;
1411
1412             if (deltaL_total >= 0.0)
1413                 continue;
1414
1415             if (deltaL_total <= 0.0)
1416                 continue;
1417
1418             if (deltaL_total >= 0.0)
1419                 continue;
1420
1421             if (deltaL_total <= 0.0)
1422                 continue;
1423
1424             if (deltaL_total >= 0.0)
1425                 continue;
1426
1427             if (deltaL_total <= 0.0)
1428                 continue;
1429
1430             if (deltaL_total >= 0.0)
1431                 continue;
1432
1433             if (deltaL_total <= 0.0)
1434                 continue;
1435
1436             if (deltaL_total >= 0.0)
1437                 continue;
1438
1439             if (deltaL_total <= 0.0)
1440                 continue;
1441
1442             if (deltaL_total >= 0.0)
1443                 continue;
1444
1445             if (deltaL_total <= 0.0)
1446                 continue;
1447
1448             if (deltaL_total >= 0.0)
1449                 continue;
1450
1451             if (deltaL_total <= 0.0)
1452                 continue;
1453
1454             if (deltaL_total >= 0.0)
1455                 continue;
1456
1457             if (deltaL_total <= 0.0)
1458                 continue;
1459
1460             if (deltaL_total >= 0.0)
1461                 continue;
1462
1463             if (deltaL_total <= 0.0)
1464                 continue;
1465
1466             if (deltaL_total >= 0.0)
1467                 continue;
1468
1469             if (deltaL_total <= 0.0)
1470                 continue;
1471
1472             if (deltaL_total >= 0.0)
1473                 continue;
1474
1475             if (deltaL_total <= 0.0)
1476                 continue;
1477
1478             if (deltaL_total >= 0.0)
1479                 continue;
1480
1481             if (deltaL_total <= 0.0)
1482                 continue;
1483
1484             if (deltaL_total >= 0.0)
1485                 continue;
1486
1487             if (deltaL_total <= 0.0)
1488                 continue;
1489
1490             if (deltaL_total >= 0.0)
1491                 continue;
1492
1493             if (deltaL_total <= 0.0)
1494                 continue;
1495
1496             if (deltaL_total >= 0.0)
1497                 continue;
1498
1499             if (deltaL_total <= 0.0)
1500                 continue;
1501
1502             if (deltaL_total >= 0.0)
1503                 continue;
1504
1505             if (deltaL_total <= 0.0)
1506                 continue;
1507
1508             if (deltaL_total >= 0.0)
1509                 continue;
1510
1511             if (deltaL_total <= 0.0)
1512                 continue;
1513
1514             if (deltaL_total >= 0.0)
1515                 continue;
1516
1517             if (deltaL_total <= 0.0)
1518                 continue;
1519
1520             if (deltaL_total >= 0.0)
1521                 continue;
1522
1523             if (deltaL_total <= 0.0)
1524                 continue;
1525
1526             if (deltaL_total >= 0.0)
1527                 continue;
1528
1529             if (deltaL_total <= 0.0)
1530                 continue;
1531
1532             if (deltaL_total >= 0.0)
1533                 continue;
1534
1535             if (deltaL_total <= 0.0)
1536                 continue;
1537
1538             if (deltaL_total >= 0.0)
1539                 continue;
1540
1541             if (deltaL_total <= 0.0)
1542                 continue;
1543
1544             if (deltaL_total >= 0.0)
1545                 continue;
1546
1547             if (deltaL_total <= 0.0)
1548                 continue;
1549
1550             if (deltaL_total >= 0.0)
1551                 continue;
1552
1553             if (deltaL_total <= 0.0)
1554                 continue;
1555
1556             if (deltaL_total >= 0.0)
1557                 continue;
1558
1559             if (deltaL_total <= 0.0)
1560                 continue;
1561
1562             if (deltaL_total >= 0.0)
1563                 continue;
1564
1565             if (deltaL_total <= 0.0)
1566                 continue;
1567
1568             if (deltaL_total >= 0.0)
1569                 continue;
1570
1571             if (deltaL_total <= 0.0)
1572                 continue;
1573
1574             if (deltaL_total >= 0.0)
1575                 continue;
1576
1577             if (deltaL_total <= 0.0)
1578                 continue;
1579
1580             if (deltaL_total >= 0.0)
1581                 continue;
1582
1583             if (deltaL_total <= 0.0)
1584                 continue;
1585
1586             if (deltaL_total >= 0.0)
1587                 continue;
1588
1589             if (deltaL_total <= 0.0)
1590                 continue;
1591
1592             if (deltaL_total >= 0.0)
1593                 continue;
1594
1595             if (deltaL_total <= 0.0)
1596                 continue;
1597
1598             if (deltaL_total >= 0.0)
1599                 continue;
1600
1601             if (deltaL_total <= 0.0)
1602                 continue;
1603
1604             if (deltaL_total >= 0.0)
1605                 continue;
1606
1607             if (deltaL_total <= 0.0)
1608                 continue;
1609
1610             if (deltaL_total >= 0.0)
1611                 continue;
1612
1613             if (deltaL_total <= 0.0)
1614                 continue;
1615
1616             if (deltaL_total >= 0.0)
1617                 continue;
1618
1619             if (deltaL_total <= 0.0)
1620                 continue;
1621
1622             if (deltaL_total >= 0.0)
1623                 continue;
1624
1625             if (deltaL_total <=
```

```

557     // Carga causada pelo efeito pistao na interface
558     double cargaPistao = VariacaoCargaEfeitoPistao(z_interface,
559             OD_abajo, OD_acima);
560
561     // L = comprimento da coluna acima da interface
562     double L = z_interface;
563
564     // Deformacao axial provocada pela carga
565     double deltaL = (cargaPistao * L) / (E_medio * area_secao);
566     deltaL_total += deltaL;
567 }
568
569     return deltaL_total;
570 }

571 double CObjetoPoco::DeltaLForcaRestauradora(double profundidade)
572 const{
573
574     double deltaLTemperatura = DeltaLTemperatura(profundidade);
575     double deltaLBalao = DeltaLEfeitoBalao(profundidade);
576     double deltaLPacker = DeltaLPistaoPacker(profundidade);
577     double deltaLCrossover = DeltaLPistaoCrossover(profundidade);
578
579     return deltaLTemperatura + deltaLBalao + deltaLPacker +
580         deltaLCrossover;
581 }

582 double CObjetoPoco::CargaInjecao(double profundidade) const {
583     const double pi = 3.141592653589793;
584     double denominador = 0.0;
585
586     // Deformacao total desejada
587     double deltaL = DeltaLForcaRestauradora(profundidade);
588
589     for (const auto& trecho : trechos) {
590         double z_i = trecho->ProfundidadeInicial();
591         double z_f = trecho->ProfundidadeFinal();
592
593         // Pula trechos abaixo da profundidade desejada
594         if (z_i > profundidade)
595             continue;

```

```

596
597     double limiteSuperior = std::min(z_f, profundidade);
598     double comprimentoUtil = limiteSuperior - z_i;
599
600     if (comprimentoUtil <= 0.0)
601         continue;
602
603     double E = trecho->ModuloElasticidade();
604     double OD = trecho->DiametroExterno();
605     double ID = trecho->DiametroInterno();
606     double area = (pi / 4.0) * (OD * OD - ID * ID);
607
608     // soma das flexibilidades: L / (E*A)
609     denominador += comprimentoUtil / (E * area);
610 }
611
612 if (denominador == 0.0)
613     return 0.0;
614
615 // pressao = deformacao total / soma das flexibilidades
616 return deltaL / denominador;
617 }

```

---

Fonte: produzido pelo autor.

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

Listing 7.8: Arquivo de implementação da classe CFluido

---

```

1 #ifndef CFLUIDO_H
2 #define CFLUIDO_H
3
4 #include <string>
5
6 /*
7 Classe que representa um fluido qualquer do sistema
8 Aqui sao armazenados nome, densidade e viscosidade
9 Essas informacoes sao usadas nos calculos de pressao, perda de
   carga, etc
10 */
11
12 class CFluido {
13 protected:
14     std::string nomeFluido;           // nome do fluido (ex: oleo ou
                                     agua)

```

```

15     double densidadeFluido = 0.0;      // densidade em lbm/gal
16     double viscosidadeFluido = 0.0;    // viscosidade em cP
17
18 public:
19     CFluido() {}
20     ~CFluido() {}
21
22     // Construtor para inicializar com todos os dados
23     CFluido(std::string nome, double densidade, double viscosidade)
24         : nomeFluido(nome), densidadeFluido(densidade),
25           viscosidadeFluido(viscosidade) {}
26
27     // getters
28     std::string Nome() const { return nomeFluido; }
29     double Densidade() const { return densidadeFluido; }
30     double Viscosidade() const { return viscosidadeFluido; }
31
32     // setters
33     void Nome(const std::string& novoNome) { nomeFluido = novoNome;
34     }
35     void Densidade(double novaDensidade) { densidadeFluido =
36         novaDensidade; }
37     void Viscosidade(double novaViscosidade) { viscosidadeFluido =
38         novaViscosidade; }
39 };
40
41 #endif

```

---

Fonte: produzido pelo autor.

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

Listing 7.9: Arquivo de implementação da classe CFluido

```

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

```

---

Fonte: produzido pelo autor.

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

Listing 7.10: Arquivo de implementação da classe CModeloReologico

```

1 #ifndef CMODELOREOLOGICO_H
2 #define CMODELOREOLOGICO_H
3
4 #include <string>
5 #include "CObjetoPoco.h"
6
7 /*
8 Classe base abstrata para os modelos reologicos
9 Define as propriedades e metodos que devem ser implementados pelos
   modelos concretos
10 Serve como interface para modelos como newtoniano, Bingham e lei da
    potencia
11 */
12
13 class CModeloReológico {
14
15 protected:
16     // Propriedades relacionadas ao escoamento e calculos
17     double fatorFriccaoPoco = 0.0;
18     double fatorFriccaoAnular = 0.0;
19     double reynoldsPoco = 0.0;
20     double reynoldsAnular = 0.0;
21     double vMediaPoco = 0.0;
22     double vMediaAnular = 0.0;
23
24     // Tipo de fluxo (laminar ou turbulento)
25     std::string fluxoPoco;
26     std::string fluxoAnular;
27
28     // Objeto que contem as propriedades do poco
29     CObjetoPoco* poco;
30
31 public:
32     // Construtores
33     CModeloReológico() {}
34     virtual ~CModeloReológico() {}
35     CModeloReológico(CObjetoPoco* poco) : poco(poco) {}
36
37     // Getters
38     double FatorFriccaoPoco() const { return fatorFriccaoPoco; }
39     double FatorFriccaoAnular() const { return fatorFriccaoAnular;
      }

```

```

40     double ReynoldsPoco() const { return reynoldsPoco; }
41     double ReynoldsAnular() const { return reynoldsAnular; }
42     double vMediaPoco() const { return vMediaPoco; }
43     double vMediaAnular() const { return vMediaAnular; }
44     std::string FluxoPoco() const { return fluxoPoco; }
45     std::string FluxoAnular() const { return fluxoAnular; }
46
47     // M etodos para determinar fatores e velocidades
48     double DeterminarFatorFriccao(double re, double n);
49     double DeterminarReynoldsPoco();
50     double DeterminarReynoldsPoco(double viscosidade);
51     double DeterminarReynoldsAnular();
52     double DeterminarReynoldsAnular(double viscosidade);
53     double DeterminarVelocidadeMediaPoco();
54     double DeterminarVelocidadeMediaAnular();
55
56     // M etodos puros (devem ser implementados pelas classes filhas
57     )
58     virtual std::string DeterminarFluxoPoco() = 0;
59     virtual std::string DeterminarFluxoAnular() = 0;
60     virtual double CalcularPerdaPorFriccaoPoco() = 0;
61     virtual double CalcularPerdaPorFriccaoAnular() = 0;
62 };
63 #endif // CMODELOREOLOGICO_H

```

---

Fonte: produzido pelo autor.

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

Listing 7.11: Arquivo de implementação da classe CModeloReológico

```

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
8 // do fluido
9 double CModeloReologico::DeterminarReynoldsPoco() {
10     reynoldsPoco = (928 * poco->DensidadeEfetivaTotal() *
11                     vMediaPoco * poco->DiâmetroRevestimentoID()) /
12                     poco->ViscosidadeEfetivaTotal();
13     return reynoldsPoco;

```

```

12 }
13
14 // Mesmo calculo, mas permitindo passar a viscosidade como
15 // parametro
15 double CModeloReologico::DeterminarReynoldsPoco(double viscosidade)
16 {
17     reynoldsPoco = (928 * poco->DensidadeEfetivaTotal() *
18                     vMediaPoco * poco->DiametroRevestimentoID()) /
19                     viscosidade;
20     return reynoldsPoco;
21
22 }
23
24 // Calcula o numero de Reynolds no espaco anular
23 double CModeloReologico::DeterminarReynoldsAnular() {
24     reynoldsAnular = (757 * poco->DensidadeEfetivaTotal() *
25                     vMediaAnular *
26                     (poco->DiametroPoco() - poco->
27                     DiametroRevestimentoOD())) /
28                     poco->ViscosidadeEfetivaTotal();
29     return reynoldsAnular;
30 }
31
32 // Mesmo calculo para o anular, com viscosidade recebida
32 // externamente
31 double CModeloReologico::DeterminarReynoldsAnular(double
32     viscosidade) {
33     reynoldsAnular = (757 * poco->DensidadeEfetivaTotal() *
34                     vMediaAnular *
35                     (poco->DiametroPoco() - poco->
36                     DiametroRevestimentoOD())) /
37                     viscosidade;
38     return reynoldsAnular;
39 }
40
41 // Calcula a velocidade media do fluido no interior da coluna (
41 // poco)
42 double CModeloReologico::DeterminarVelocidadeMediaPoco() {
43     vMediaPoco = poco->Vazao() / (2.448 * std::pow(poco->
44         DiametroRevestimentoID(), 2));
45     return vMediaPoco;
46 }

```

```

43
44 // Calcula a velocidade m dia no espaço anular entre a coluna e o
   revestimento
45 double CModeloReologico::DeterminarVelocidadeMediaAnular() {
46     vMediaAnular = poco->Vazao() /
47         (2.448 * (std::pow(poco->DiametroPoco(), 2) -
48             std::pow(poco->DiametroRevestimentoOD(), 2)));
49     ;
50
51     return vMediaAnular;
52 }

```

---

Fonte: produzido pelo autor.

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

Listing 7.12: Arquivo de implementação da classe CModeloNewtoniano

---

```

1 #ifndef CMODELONEWTONIANO_H
2 #define CMODELONEWTONIANO_H
3
4 #include "CModeloReologico.h"
5
6 /*
7 Classe que representa o modelo reológico newtoniano
8 Nesse caso, a viscosidade é constante e independe da taxa de
   deformação
9 A classe herda de CModeloReologico e implementa os métodos
   específicos 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 poço
20     CModeloNewtoniano(CObjetoPoco* poco) : CModeloReologico(poco) {
21         DeterminarFluxoPoco();
22         DeterminarFluxoAnular();
23     }
24
25     // Métodos obrigatórios sobrescritos do modelo base
26     std::string DeterminarFluxoPoco() override;

```

```

27     std::string DeterminarFluxoAnular() override;
28     double CalcularPerdaPorFriccaoPoco() override;
29     double CalcularPerdaPorFriccaoAnular() override;
30 };
31
32 #endif // CMODELONEWTONIANO_H

```

---

Fonte: produzido pelo autor.

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

**Listing 7.13:** Arquivo de implementação da classe CModeloNewtoniano

```

1 #include "CModeloNewtoniano.h"
2 #include <cmath>
3
4 // Determina o tipo de fluxo no poco com base no numero de Reynolds
5 std::string CModeloNewtoniano::DeterminarFluxoPoco() {
6     DeterminarVelocidadeMediaPoco();
7     DeterminarReynoldsPoco();
8
9     // Valor limite de 2100 usado para separar regime laminar e
10    turbulent
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
15 // Reynolds
15 std::string CModeloNewtoniano::DeterminarFluxoAnular() {
16     DeterminarVelocidadeMediaAnular();
17     DeterminarReynoldsAnular();
18
19     fluxoAnular = (reynoldsAnular <= 2100) ? "Laminar" : "
20         Turbulento";
21     return fluxoAnular;
21}
22
23 // Calcula a perda de carga por friccao no poco para regime laminar
24 // ou turbulent
24 double CModeloNewtoniano::CalcularPerdaPorFriccaoPoco() {
25     if (fluxoPoco.empty()) {
26         DeterminarFluxoPoco();
27     }
28

```

```

29     if (fluxoPoco == "Laminar") {
30         return (poco->ViscosidadeEfetivaTotal() * vMediaPoco) /
31             (1500 * std::pow(poco->DiametroRevestimentoID(), 2))
32             ;
33     } else {
34         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std
35             ::pow(vMediaPoco, 1.75) *
36             std::pow(poco->ViscosidadeEfetivaTotal(), 0.25)) /
37             (1800 * std::pow(poco->DiametroRevestimentoID(),
38             1.25));
39 }
40
41 // Calcula a perda de carga por friccao no espaco anular
42 double CModeloNewtoniano::CalcularPerdaPorFriccaoAnular() {
43     if (fluxoAnular.empty()) {
44         DeterminarFluxoAnular();
45     }
46
47     double diametroAnular = poco->DiametroPoco() - poco->
48         DiametroRevestimentoOD();
49
50     if (fluxoAnular == "Laminar") {
51         return (poco->ViscosidadeEfetivaTotal() * vMediaAnular) /
52             (1000 * std::pow(diametroAnular, 2));
53     } else {
54         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std
55             ::pow(vMediaAnular, 1.75) *
56             std::pow(poco->ViscosidadeEfetivaTotal(), 0.25)) /
57             (1396 * std::pow(diametroAnular, 1.25));
58     }
59 }

```

---

Fonte: produzido pelo autor.

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

Listing 7.14: Arquivo de implementação da classe CModeloBingham

```

1 #ifndef CMODELOBINGHAM_H
2 #define CMODELOBINGHAM_H
3
4 #include "CModeloReologico.h"
5
6 /*

```

```

7 Classe que representa o modelo reologico de Bingham
8 Esse modelo e usado para fluidos com um ponto de escoamento
    definido
9 A implementacao baseia-se na heranca da classe CModeloReologico
10 */
11
12 class CModeloBingham : public CModeloReologico {
13
14 protected:
15     // Parametros especificos do modelo de Bingham
16     double viscosidadePlastica = 0.0;
17     double pontoDeEscoamento = 0.0;
18
19     // Valores criticos 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,
37                     double pontoDeEscoamento)
38         : CModeloReologico(poco),
39           viscosidadePlastica(viscosidadePlastica),
40           pontoDeEscoamento(pontoDeEscoamento)
41     {
42         DeterminarFluxoPoco();
43         DeterminarFluxoAnular();
44     }
45
46     // Getters
47     double ViscosidadePlastica() const { return viscosidadePlastica;

```

```

        ;
    }

47     double PontoDeEscoamento() const { return pontoDeEscoamento; }
48     double ReynoldsCriticoPoco() const { return reynoldsCriticoPoco
        ; }
49     double ReynoldsCriticoAnular() const { return
        reynoldsCriticoAnular; }
50     double ReynoldsHedstronPoco() const { return
        reynoldsHedstronPoco; }
51     double ReynoldsHedstronAnular() const { return
        reynoldsHedstronAnular; }

52
53 // Setters
54 void ViscosidadePlastica(double valor) { viscosidadePlastica =
    valor; }
55 void PontoDeEscoamento(double valor) { pontoDeEscoamento =
    valor; }

56
57 // Metodos especificos do modelo de Bingham
58     double DeterminarReynoldsCritico(double hedstron);
59     double DeterminarReynoldsHedstronPoco();
60     double DeterminarReynoldsHedstronAnular();
61     std::string DeterminarFluxoPoco() override;
62     std::string DeterminarFluxoAnular() override;
63     double CalcularPerdaPorFriccaoPoco() override;
64     double CalcularPerdaPorFriccaoAnular() override;
65 };
66
67 #endif // CMODELOBINGHAM_H

```

---

Fonte: produzido pelo autor.

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

**Listing 7.15:** Arquivo de implementação da classe CModeloBingham

---

```

1 #include "CModeloBingham.h"
2 #include <cmath>
3 #include <QApplication>
4 #include <QFileDialog>
5 #include <QString>
6 #include <QMessageBox>
7
8 // Determina o valor de Reynolds critico com base no numero de
// Hedstron
9 // Utiliza metodo numerico de Newton-Raphson para resolver a

```

```

equacao implicita

10 double CModeloBingham::DeterminarReynoldsCritico(double hedstron) {
11     // f(x) = ((x / (1 - x)^3) * 16800) - Hedstron
12     auto func = [hedstron](double x) {
13         return ((x / std::pow(1 - x, 3)) * 16800) - hedstron;
14     };
15
16     // Derivada de f(x)
17     auto derivadaFunc = [] (double x) {
18         return (16800 * (1 + x) / std::pow(1 - x, 4));
19     };
20
21     // Metodo de Newton-Raphson para aproximar a raiz
22     auto newtonRaphson = [&] (double x) {
23         for (int i = 0; i < 10000; ++i) {
24             x = x - func(x) / derivadaFunc(x);
25             if (fabs(func(x)) < 1e-2) break;
26         }
27         return x;
28     };
29
30     // Chute inicial
31     double y = newtonRaphson(0.99);
32
33     // Retorna o Reynolds critico com base em y encontrado
34     return ((1 - ((4.0 / 3.0) * y) + ((1.0 / 3.0) * std::pow(y, 4)))
35         * hedstron) / (8 * y);
36
37 // Calcula o numero de Hedstron para o escoamento no poco
38 double CModeloBingham::DeterminarReynoldsHedstronPoco() {
39     reynoldsHedstronPoco =
40         (37100 * poco->DensidadeEfetivaTotal() * pontoDeEscoamento
41         *
42         std::pow(poco->DiametroRevestimentoID(), 2)) /
43         std::pow(viscosidadePlastica, 2);
44
45     return reynoldsHedstronPoco;
46
47 // Calcula o numero de Hedstron para o escoamento no espaco anular
48 double CModeloBingham::DeterminarReynoldsHedstronAnular() {

```

```

49     double diametroAnular = poco->DiametroPoco() - poco->
50         DiametroRevestimentoOD();
51
52     reynoldsHedstronAnular =
53         (24700 * poco->DensidadeEfetivaTotal() * pontoDeEscoamento
54             *
55             std::pow(diametroAnular, 2)) /
56             std::pow(viscosidadePlastica, 2);
57
58     return reynoldsHedstronAnular;
59 }
60
61 // Determina o tipo de fluxo no poto (laminar ou turbulento)
62 std::string CModeloBingham::DeterminarFluxoPoco() {
63     DeterminarVelocidadeMediaPoco();
64     DeterminarReynoldsPoco(viscosidadePlastica);
65     DeterminarReynoldsHedstronPoco();
66
67     reynoldsCriticoPoco = DeterminarReynoldsCritico(
68         reynoldsHedstronPoco);
69
70     fluxoPoco = (reynoldsPoco <= reynoldsCriticoPoco) ? "Laminar" :
71         "Turbulento";
72     return fluxoPoco;
73 }
74
75 // Determina o tipo de fluxo no espaco anular
76 std::string CModeloBingham::DeterminarFluxoAnular() {
77     DeterminarVelocidadeMediaAnular();
78     DeterminarReynoldsAnular(viscosidadePlastica);
79     DeterminarReynoldsHedstronAnular();
80
81     reynoldsCriticoAnular = DeterminarReynoldsCritico(
82         reynoldsHedstronAnular);
83
84     fluxoAnular = (reynoldsAnular <= reynoldsCriticoAnular) ? "
85         Laminar" : "Turbulento";
86     return fluxoAnular;
87 }
88
89 // Calcula a perda de carga por friccao no poto
90 double CModeloBingham::CalcularPerdaPorFriccaoPoco() {

```

```

85     if (fluxoPoco == "Laminar") {
86         return ((viscosidadePlastica * vMediaPoco) /
87                 (1500 * std::pow(poco->DiametroRevestimentoID(), 2)
88                  )) +
89                 (pontoDeEscoamento / (225 * poco->
90                           DiametroRevestimentoID())));
91     } else {
92         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std
93                 ::pow(vMediaPoco, 1.75) *
94                 std::pow(viscosidadePlastica, 0.25)) /
95                 (1800 * std::pow(poco->DiametroRevestimentoID(),
96                         1.25));
97     }
98 }
99
100 // Calcula a perda de carga por friccao no espaco anular
101 double CModeloBingham::CalcularPerdaPorFriccaoAnular() {
102     double diametroAnular = poco->DiametroPoco() - poco->
103             DiametroRevestimentoOD();
104
105     if (fluxoAnular == "Laminar") {
106         return ((viscosidadePlastica * vMediaAnular) /
107                 (1000 * std::pow(diametroAnular, 2))) +
108                 (pontoDeEscoamento / (200 * diametroAnular));
109     } else {
110         return (std::pow(poco->DensidadeEfetivaTotal(), 0.75) * std
111                 ::pow(vMediaAnular, 1.75) *
112                 std::pow(viscosidadePlastica, 0.25)) /
113                 (1396 * std::pow(diametroAnular, 1.25));
114     }
115 }

```

---

Fonte: produzido pelo autor.

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

Listing 7.16: Arquivo de implementação da classe CModeloPotencia

```

1 #ifndef CMODELOPOTENCIA_H
2 #define CMODELOPOTENCIA_H
3
4 #include "CModeloReologico.h"
5
6 /*
7 Classe que representa o modelo reológico da Lei da Potencia

```

```

8 Esse modelo e aplicado a fluidos pseudoplastico ou dilatante (n
9   diferente de 1)
10 A classe herda de CModeloReologico e implementa os metodos
11   especificos do modelo
12 */
13
14 class CModeloPotencia : public CModeloReologico {
15
16 protected:
17   // Parametros do modelo da potencia
18   double indiceDeConsistencia;           // K
19   double indiceDeComportamento;          // n
20
21   // Reynolds critico arbitrario (2100 como base de separacao)
22   double reynoldsCriticoPoco;
23   double reynoldsCriticoAnular;
24
25 public:
26   // Construtores
27   CModeloPotencia() {}
28   ~CModeloPotencia() {}
29
30   // Construtor com inicializacao do poco e do indice de
31   // consistencia
32   CModeloPotencia(CObjetoPoco* poco, double indiceDeConsistencia,
33                   double indiceDeComportamento)
34     : CModeloReologico(poco),
35       indiceDeConsistencia(indiceDeConsistencia),
36       indiceDeComportamento(indiceDeComportamento)
37
38   {
39     DeterminarFluxoPoco();
40     DeterminarFluxoAnular();
41     IndiceDeComportamento(indiceDeComportamento);
42   }
43
44   // Getters
45   double ReynoldsCriticoPoco() const { return reynoldsCriticoPoco
46   ; }
47   double ReynoldsCriticoAnular() const { return
48     reynoldsCriticoAnular; }
49   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 7.17 a implementação da classe CModeloPotencia.

**Listing 7.17:** Arquivo de implementação da classe CModeloPotencia

```

1 #include "CModeloPotencia.h"
2 #include <cmath>
3 #include <stdexcept>
4
5 // Retorna o fator de atrito para fluido da lei da potencia com
6 // base em NRe e n
7
8 double CModeloPotencia::DeterminarFatorFriccao(double NRe, double n
    ) {
    // Coeficientes obtidos via regressao log-log com base nos
    dados reais do grafico

```

```

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,
11        -0.20192, -0.30519};
12    const double b_vals[] = {-1.14079, -1.12562, -1.25944,
13        -1.66278, -1.33815};
14
15    // Interpola o linear entre os dois pontos mais proximos
16    double a = 0.0, b = 0.0;
17    for (int i = 1; i < 5; ++i) {
18        if (n <= n_vals[i - 1] && n >= n_vals[i]) {
19            double t = (n - n_vals[i]) / (n_vals[i - 1] - n_vals[i]);
20            a = a_vals[i] + t * (a_vals[i - 1] - a_vals[i]);
21            b = b_vals[i] + t * (b_vals[i - 1] - b_vals[i]);
22            break;
23        }
24    }
25
26    double logF = a * std::log10(NRe) + b;
27
28 // Funcao generica para Reynolds critico (nao esta sendo usada
29 // diretamente aqui)
30 double CModeloPotencia::DeterminarReynoldsCritico(double Reynolds)
31 {
32     return 183.13 * std::pow(Reynolds, 0.3185);
33 }
34
35 // Calcula o numero de Reynolds para o escoamento no poco
36 double CModeloPotencia::DeterminarReynoldsPoco() {
37     reynoldsPoco =
38         ((89100 * poco->DensidadeEfetivaTotal() * std::pow(
39             vMediaPoco, 2 - indiceDeComportamento)) /
40             indiceDeConsistencia) *
41             (std::pow((0.0416 * poco->DiametroRevestimentoID()) / (3 +
42                 (1 / indiceDeComportamento)), indiceDeComportamento));
43
44     reynoldsCriticoPoco = DeterminarReynoldsCritico(reynoldsPoco);
45
46     return reynoldsPoco;

```

```

43 }
44
45 // Calcula o numero de Reynolds no espaco anular
46 double CModeloPotencia::DeterminarReynoldsAnular() {
47     reynoldsAnular =
48         (((109000 * poco->DensidadeEfetivaTotal() * std::pow(
49             vMediaAnular, 2 - indiceDeComportamento)) /
50             indiceDeConsistencia) *
51             (std::pow((0.0208 * (poco->DiametroPoco() - poco->
52                 DiametroRevestimentoOD())) / (2 + (1 /
53                     indiceDeComportamento)), indiceDeComportamento));
54
55     reynoldsCriticoAnular = DeterminarReynoldsCritico(
56         reynoldsAnular);
57
58     return reynoldsAnular;
59 }
60
61
62 // Determina o tipo de fluxo no poco com base no valor de Reynolds
63 // calculado
64 std::string CModeloPotencia::DeterminarFluxoPoco() {
65     vMediaPoco = DeterminarVelocidadeMediaPoco();
66     reynoldsPoco = DeterminarReynoldsPoco();
67     fluxoPoco = (reynoldsPoco <= reynoldsCriticoPoco) ? "Laminar" :
68         "Turbulento";
69     return fluxoPoco;
70 }
71
72 // Determina o tipo de fluxo no espaco anular
73 std::string CModeloPotencia::DeterminarFluxoAnular() {
74     vMediaAnular = DeterminarVelocidadeMediaAnular();
75     reynoldsAnular = DeterminarReynoldsAnular();
76     fluxoAnular = (reynoldsAnular <= reynoldsCriticoAnular) ? "
77         Laminar" : "Turbulento";
78     return fluxoAnular;
79 }
80
81
82 // Calcula a perda de carga por friccao no poco, separando para
83 // fluxo laminar e turbulento
84 double CModeloPotencia::CalcularPerdaPorFriccaoPoco() {
85     fatorFriccaoPoco = DeterminarFatorFriccao(reynoldsPoco,
86         indiceDeComportamento);

```

```

75
76     if (fluxoPoco == "Laminar") {
77         return ((indiceDeConsistencia * std::pow(vMediaPoco, -
78             indiceDeComportamento)) *
79                 std::pow(( (3 + (1 / indiceDeComportamento)) /
80                     0.0416), indiceDeComportamento)) /
81                 (144000 * std::pow(poco->DiametroRevestimentoID(), 1
82                     + indiceDeComportamento));
83     } else {
84         return (fatorFriccaoPoco * poco->DensidadeEfetivaTotal() *
85                 std::pow(vMediaPoco, 2)) /
86                 (25.8 * poco->DiametroRevestimentoID());
87     }
88 }
89
90 // Calcula a perda de carga por friccao no espaco anular
91 double CModeloPotencia::CalcularPerdaPorFriccaoAnular() {
92     double diametroAnular = poco->DiametroPoco() - poco->
93         DiametroRevestimentoOD();
94     fatorFriccaoAnular = DeterminarFatorFriccao(reynoldsAnular,
95         indiceDeComportamento);
96
97     if (fluxoAnular == "Laminar") {
98         return ((indiceDeConsistencia * std::pow(vMediaAnular, -
99             indiceDeComportamento)) *
100                 std::pow(((2 + (1 / indiceDeComportamento)) /
101                     0.0208), indiceDeComportamento)) /
102                 (144000 * std::pow(diametroAnular, 1 +
103                     indiceDeComportamento));
104     } else {
105         return (fatorFriccaoAnular * poco->DensidadeEfetivaTotal()
106                 * std::pow(vMediaAnular, 2)) /
107                 (21.1 * diametroAnular);
108     }
109 }

```

---

Fonte: produzido pelo autor.

### 7.1.1 Código Qt (interface)

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

Listing 7.18: Arquivo de implementação da classe CJanelaAdicionarFluido

---

```
1 #ifndef CJANELAADICIONARFLUIDO_H
2 #define CJANELAADICIONARFLUIDO_H
3
4 #include <QDialog>
5
6 /*
7 Classe da interface grafica que permite ao usuario adicionar ou
8 editar um fluido
9 Armazena os dados digitados: nome, densidade, viscosidade e faixa
10 de profundidade
11 */
12
13
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) {
28         profundidadeInicial = valor; }
29     void ProfundidadeFinal(const QString& valor) {
30         profundidadeFinal = valor; }
31     void ModoEdicao(bool opcao) { modoEdicao = opcao; }
32
33     // metodos para acessar os dados
34     QString NomeFluido() const { return nomeFluido; }
35     QString Densidade() const { return densidade; }
36     QString Viscosidade() const { return viscosidade; }
37     QString ProfundidadeInicial() const { return
38         profundidadeInicial; }
39     QString ProfundidadeFinal() const { return profundidadeFinal; }
40     bool ModoEdicao() const { return modoEdicao; }
```

```

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 7.19 a implementação da classe CJanelaAdicionarFluido.

**Listing 7.19:** Arquivo de implementação da classe CJanelaAdicionarFluido

```

1 #include "CJanelaAdicionarFluido.h"
2 #include "ui_CJanelaAdicionarFluido.h"
3 #include <QMessageBox>
4
5 // Construtor da janela - inicializa a interface grafica
6 CJanelaAdicionarFluido::CJanelaAdicionarFluido(QWidget *parent)
7     : QDialog(parent)
8     , ui(new Ui::CJanelaAdicionarFluido)
9 {
10     ui->setupUi(this);
11 }
12
13 // Destruitor - limpa a interface da memoria
14 CJanelaAdicionarFluido::~CJanelaAdicionarFluido()
15 {
16     delete ui;
17 }
18
19 // Acao quando o usuario clica em "OK"
20 void CJanelaAdicionarFluido::on_btnReturn_accepted()
21 {

```

```

22     // Se for modo de edicao, habilita todos os campos
23     if (ModoEdicao()) {
24         NomeFluido(ui->LnValorNome->text());
25         Densidade(ui->LnValorDensidade->text());
26         Viscosidade(ui->LnValorViscosidade->text());
27         ProfundidadeInicial(ui->LnValorProfundidadeInicial->text())
28             ;
29         ProfundidadeFinal(ui->LnValorProfundidadeFinal->text());
30
31         // Verifica se algum campo ficou vazio
32         if (ui->LnValorNome->text().isEmpty() ||
33             ui->LnValorDensidade->text().isEmpty() ||
34             ui->LnValorViscosidade->text().isEmpty() ||
35             ui->LnValorProfundidadeInicial->text().isEmpty() ||
36             ui->LnValorProfundidadeFinal->text().isEmpty()) {
37             QMessageBox::warning(this, "Erro", "Por\u00e9m favor, \u00e7o preencha
38                 \u2022 todos \u2022 os \u2022 campos !");
39         }
40
41     } else {
42         // Modo sem edicao da faixa de profundidade
43         NomeFluido(ui->LnValorNome->text());
44         Densidade(ui->LnValorDensidade->text());
45         Viscosidade(ui->LnValorViscosidade->text());
46
47         ui->LnValorProfundidadeInicial->setDisabled(true);
48         ui->LnValorProfundidadeFinal->setDisabled(true);
49
50         if (ui->LnValorNome->text().isEmpty() ||
51             ui->LnValorDensidade->text().isEmpty() ||
52             ui->LnValorViscosidade->text().isEmpty()) {
53             QMessageBox::warning(this, "Erro", "Por\u00e9m favor, \u00e7o preencha
54                 \u2022 todos \u2022 os \u2022 campos !");
55         }
56     }
57 }
58
59 // Acao quando o usuario clica em "Cancelar"
60 void CJanelaAdicionarFluido::on_btnReturn_rejected()
61 {
62     close();
63 }
```

---

Fonte: produzido pelo autor.

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

Listing 7.20: Arquivo de implementação da classe CJanelaAdicionarTrechoTubulacao

---

```
1 #ifndef CJANELAADICIONARTRECHOTUBULACAO_H
2 #define CJANELAADICIONARTRECHOTUBULACAO_H
3
4 #include <QDialog>
5
6 /*
7 Classe da interface grafica que permite adicionar um trecho de
8 tubulacao ao sistema
9 usuario insere informacoes da geometria da tubulacao,
10 propriidades fisicas e dados do fluido
11 Esses dados sao usados para simulacoes de comportamento termico e
12 mecanico da tubulacao
13 */
14
15
16 class CJanelaAdicionarTrechoTubulacao : public QDialog
17 {
18     Q_OBJECT
19
20 public:
21     explicit CJanelaAdicionarTrechoTubulacao(QWidget *parent =
22         nullptr);
23     ~CJanelaAdicionarTrechoTubulacao();
24
25     // metodos para acessar os dados inseridos
26     QString Trecho() const { return trecho; }
27     QString ProfundidadeInicial() const { return
28         profundidadeInicial; }
29     QString ProfundidadeFinal() const { return profundidadeFinal; }
30     QString DiametroExterno() const { return diametroExterno; }
31     QString DiametroInterno() const { return diametroInterno; }
32     QString CoeficientePoisson() const { return coeficientePoisson;
33     }
34     QString CoeficienteExpansaoTermica() const { return
35         coeficienteExpansaoTermica; }
```

```

32     QString ModuloElasticidade() const { return moduloElasticidade; }
33     }
34     QString PesoUnidade() const { return pesoUnidade; }
35     QString NomeFluido() const { return nomeFluido; }
36     QString Densidade() const { return densidade; }
37     QString Viscosidade() const { return viscosidade; }
38
39     // metodos para definir os dados
40     void Trecho(const QString& valor) { trecho = valor; }
41     void ProfundidadeInicial(const QString& valor) {
42         profundidadeInicial = valor; }
43     void ProfundidadeFinal(const QString& valor) {
44         profundidadeFinal = valor; }
45     void DiametroExterno(const QString& valor) { diametroExterno =
46         valor; }
47     void DiametroInterno(const QString& valor) { diametroInterno =
48         valor; }
49     void CoeficientePoisson(const QString& valor) {
50         coeficientePoisson = valor; }
51     void CoeficienteExpansaoTermica(const QString& valor) {
52         coeficienteExpansaoTermica = valor; }
53     void ModuloElasticidade(const QString& valor) {
54         moduloElasticidade = valor; }
55     void PesoUnidade(const QString& valor) { pesoUnidade = valor; }
56     void NomeFluido(const QString& valor) { nomeFluido = valor; }
57     void Densidade(const QString& valor) { densidade = valor; }
58     void Viscosidade(const QString& valor) { viscosidade = valor; }
59     void ModoEdicao(bool opcao) { edit = opcao; }
60
61     private slots:
62     void on_btnReturn_accepted(); // acao ao confirmar
63     void on_btnReturn_rejected(); // acao ao cancelar
64
65     private:
66     bool edit = false; // indica se esta no modo de edicao
67     Ui::CJanelaAdicionarTrechoTubulacao *ui = nullptr;
68
69     // dados da tubulacao
70     QString trecho;
71     QString profundidadeInicial;
72     QString profundidadeFinal;
73     QString diametroExterno;

```

```

66     QString diametroInterno;
67     QString coeficientePoisson;
68     QString coeficienteExpansaoTermica;
69     QString moduloElasticidade;
70     QString pesoUnidade;
71
72     // dados do fluido no trecho
73     QString nomeFluido;
74     QString densidade;
75     QString viscosidade;
76 };
77
78 #endif // CJANELAADICIONARTRECHOTUBULACAO_H

```

---

Fonte: produzido pelo autor.

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

Listing 7.21: Arquivo de implementação da classe CJanelaAdicionarTrechoTubulacao

```

1 #include "CJanelaAdicionarTrechoTubulacao.h"
2 #include "ui_CJanelaAdicionarTrechoTubulacao.h"
3 #include <QMessageBox>
4
5 // Construtor: inicializa a interface
6 CJanelaAdicionarTrechoTubulacao::CJanelaAdicionarTrechoTubulacao(
7     QWidget *parent)
8     : QDialog(parent)
9     , ui(new Ui::CJanelaAdicionarTrechoTubulacao)
10 {
11     ui->setupUi(this);
12 }
13 // Destrutor: libera memoria da interface
14 CJanelaAdicionarTrechoTubulacao::~CJanelaAdicionarTrechoTubulacao()
15 {
16     delete ui;
17 }
18
19 // Acao ao clicar em OK
20 void CJanelaAdicionarTrechoTubulacao::on_btnReturn_accepted()
21 {
22     // verifica se campos obrigatorios estao vazios
23     bool camposVazios =

```

```

24     ui->editTrecho->text().isEmpty() ||
25     ui->editProfInicial->text().isEmpty() ||
26     ui->editProfFinal->text().isEmpty() ||
27     ui->editDiametroExterno->text().isEmpty() ||
28     ui->editDiametroInterno->text().isEmpty() ||
29     ui->editCoefPoisson->text().isEmpty() ||
30     ui->editCoefExpansao->text().isEmpty() ||
31     ui->editModuloElasticidade->text().isEmpty() ||
32     ui->editPesoUnid->text().isEmpty() ||
33     ui->editNome->text().isEmpty() ||
34     ui->editDensidade->text().isEmpty() ||
35     ui->editViscosidade->text().isEmpty();
36
37     if (edit && camposVazios) {
38         QMessageBox::warning(this, "Erro", "Por favor, preencha
39             todos os campos!");
40         return; // nao continua se estiver faltando campo
41     }
42
43     // define todos os valores a partir dos campos
44     Trecho(ui->editTrecho->text());
45     ProfundidadeInicial(ui->editProfInicial->text());
46     ProfundidadeFinal(ui->editProfFinal->text());
47     DiametroExterno(ui->editDiametroExterno->text());
48     DiametroInterno(ui->editDiametroInterno->text());
49     CoeficientePoisson(ui->editCoefPoisson->text());
50     CoeficienteExpansaoTermica(ui->editCoefExpansao->text());
51     ModuloElasticidade(ui->editModuloElasticidade->text());
52     PesoUnidade(ui->editPesoUnid->text());
53     NomeFluido(ui->editNome->text());
54     Densidade(ui->editDensidade->text());
55     Viscosidade(ui->editViscosidade->text());
56
57 // Acao ao clicar em Cancelar
58 void CJanelaAdicionarTrechoTubulacao::on_btnReturn_rejected()
59 {
60     close();
61 }

```

---

Fonte: produzido pelo autor.

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

Listing 7.22: Arquivo de implementação da classe CJanelaGraficoPressaoHidroestatica

---

```
1 #ifndef CJANELAGRAFICOPRESSAOHIDROESTATICA_H
2 #define CJANELAGRAFICOPRESSAOHIDROESTATICA_H
3
4 #include <QDialog>
5 #include <vector>
6
7 /*
8 Classe da interface grafica que exibe o grafico de pressao
9 hidrostatica x profundidade
10 Recebe os dados (profundidade e pressao) e plota o grafico usando
11 QCustomPlot
12 Permite exportar o grafico em imagem
13 */
14
15 namespace Ui {
16
17 class CJanelaGraficoPressaoHidroestatica;
18 }
19
20
21 public:
22     explicit CJanelaGraficoPressaoHidroestatica(
23         const std::pair<std::vector<double>, std::vector<double>>&
24             dados,
25         QWidget *parent = nullptr);
26     ~CJanelaGraficoPressaoHidroestatica();
27
28     // metodo para atualizar os dados do grafico
29     void PerfilHidrostatico(const std::pair<std::vector<double>,
30                             std::vector<double>>& novoPerfil);
31
32 private slots:
33     // acao ao clicar no botao de exportar imagem
34     void on_BtnExportarGrafico_clicked();
35
36     // vetores com dados de profundidade e pressao
37
```

```
38     std::vector<double> profundidades;
39     std::vector<double> pressoes;
40
41     // funcao que monta o grafico
42     void PlotarGraficoPressaoxProfundidade();
43 };
44
45 #endif // CJANELAGRAFICOPRESSAOHIDROESTATICA_H
```

Fonte: produzido pelo autor.

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

Listing 7.23: Arquivo de implementação da classe CJanelaGraficoPressaoHidroestatica

```
1 #include "CJanelaGraficoPressaoHidroestatica.h"
2 #include "ui_CJanelaGraficoPressaoHidroestatica.h"
3 #include <QFileDialog>
4 #include <QMessageBox>
5 #include <algorithm>
6
7 // Construtor: recebe os dados de profundidade e pressao e plota o
8 // grafico
9 CJanelaGraficoPressaoHidroestatica::
10    CJanelaGraficoPressaoHidroestatica(
11        const std::pair<std::vector<double>, std::vector<double>>&
12            dados,
13        QWidget *parent)
14        : QDialog(parent)
15        , ui(new Ui::CJanelaGraficoPressaoHidroestatica)
16        , profundidades(dados.first)
17        , pressoes(dados.second)
18    {
19        ui->setupUi(this);
20        PlotarGraficoPressaoxProfundidade();
21    }
22    ~CJanelaGraficoPressaoHidroestatica()
23    {
24        delete ui;
25    }
```

```

26 // Permite atualizar os dados apos a criacao da janela
27 void CJanelaGraficoPressaoHidroestatica::PerfilHidrostatico(
28     const std::pair<std::vector<double>, std::vector<double>>&
29     novoPerfil)
30 {
31     profundidades = novoPerfil.first;
32     pressoes = novoPerfil.second;
33     PlotarGraficoPressaoxProfundidade();
34 }
35 // Funcao que plota o grafico com base na relacao profundidade x
36 // pressao
37 // Parte da logica para segmentacao por inclinacao foi adaptada com
38 // auxilio de IA (ChatGPT)
39 void CJanelaGraficoPressaoHidroestatica::
40     PlotarGraficoPressaoxProfundidade()
41 {
42     QVector<double> x(profundidades.begin(), profundidades.end());
43     QVector<double> y(pressoes.begin(), pressoes.end());
44
45     auto *plot = ui->customPlotPressaoMediaProfundidade;
46     plot->clearGraphs();
47
48     // Funcao lambda para comparar inclinacao entre segmentos
49     auto isMesmaInclinacao = [] (double dy1, double dy2, double
50         tolerancia) {
51         return std::abs(dy1 - dy2) < tolerancia;
52     };
53
54     double tolerancia = 1e-2;
55     double inclinacaoAnterior = (y[1] - y[0]) / (x[1] - x[0]);
56
57     trechoX.push_back(x[0]);
58     trechoY.push_back(y[0]);
59
60     for (int i = 1; i < x.size(); ++i) {
61         double inclinacaoAtual = (y[i] - y[i - 1]) / (x[i] - x[i -
62             1]);
63
64         // Se mudou muito a inclinacao, cria um novo trecho no

```

```

        grafico
62    if (!isMesmaInclinacao(inclinacaoAtual, inclinacaoAnterior,
63        tolerancia)) {
64        int index = plot->graphCount();
65        plot->addGraph();
66        plot->graph(index)->setData(trechoX, trechoY);
67
68        // Uso de cor e estilo dinamico inspirado em sugestao
69        // de IA para melhorar visualizacao
70        QPen pen;
71        pen.setWidth(1);
72        pen.setColor(QColor::fromHsv((index * 70) % 360, 255,
73            200));
74        plot->graph(index)->setPen(pen);
75
76        QCPScatterStyle estilo(QCPScatterStyle::ssCircle, 4);
77        plot->graph(index)->setScatterStyle(estilo);
78        plot->graph(index)->setLineStyle(QCPGraph::lsLine);
79
80        plot->graph(index)->setName(QString("Fluido %1").arg(
81            index + 1));
82
83        trechoX.clear();
84        trechoY.clear();
85        inclinacaoAnterior = inclinacaoAtual;
86    }
87
88    // adiciona ultimo trecho
89    int index = plot->graphCount();
90    plot->addGraph();
91    plot->graph(index)->setData(trechoX, trechoY);
92
93    QPen pen;
94    pen.setWidth(1);
95    pen.setColor(QColor::fromHsv((index * 70) % 360, 255, 200));
96    plot->graph(index)->setPen(pen);
97
98    QCPScatterStyle estilo(QCPScatterStyle::ssCircle, 4);

```



```

135     plot->replot();
136 }
137
138 // Botao de exportar imagem do grafico (logica de exportacao por
139 // QPixmap)
139 void CJanelaGraficoPressaoHidroestatica::
140     on_BtnExportarGrafico_clicked()
140 {
141     QString fileName = QFileDialog::getSaveFileName(this, "Salvar
142         grafico", "", "PNG (*.png);;JPEG (*.jpg)");
143
143     if (!fileName.isEmpty()) {
144         QPixmap imagem = ui->customPlotPressaoMediaProfundidade ->
145             toPixmap(800, 600);
146         imagem.save(fileName);
147     }

```

---

Fonte: produzido pelo autor.

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

Listing 7.24: Arquivo de implementação da classe CJanelaMenu

```

1 #ifndef CJANELAMENU_H
2 #define CJANELAMENU_H
3
4 #include < QMainWindow >
5
6 /*
7 Janela principal do programa, que funciona como menu inicial
8 Aqui o usuario pode escolher entre os dois modulos principais do
9 software:
10 - Modulo 1: simulacoes de pressao e escoamento
11 - Modulo 2: analise de deslocamentos axiais da tubulacao
12 */
13 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
23     ~JanelaMenu(); // destrutor
24
25 private slots:
26     void on_btnModulo01_clicked(); // acao ao clicar no Modulo 1
27     void on_btnModulo02_clicked(); // acao ao clicar no Modulo 2
28
29 private:
30     Ui::JanelaMenu *ui;
31 };
32
33 #endif // CJANELAMENU_H

```

---

Fonte: produzido pelo autor.

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

Listing 7.25: Arquivo de implementação da classe CJanelaMenu

```

1 #include "CJanelaMenu.h"
2 #include "ui_CJanelaMenu.h"
3
4 #include "CSimuladorReologico.h"
5 #include "CSimuladorPerdaTubulacao.h"
6
7 // Construtor da janela principal do menu
8 JanelaMenu::JanelaMenu(QWidget *parent)
9     : QMainWindow(parent)
10    , ui(new Ui::JanelaMenu)
11 {
12     ui->setupUi(this);
13
14     // Caixa de texto apenas para visualizacao, sem interacao
15     ui->textEdit->setReadOnly(true);
16     ui->textEdit->setMouseTracking(false);
17     ui->textEdit->setFocusPolicy(Qt::NoFocus);
18
19     // Configuracao da descricao do Modulo 1
20     ui->lbnModulo01->.setAlignment(Qt::AlignCenter);
21
22     // Texto com HTML para deixar a explicacao mais visual
23     QString buttonText_01 =
24         "<b>Modulo 01 - Hidr ulica de Perfura o </b><br>"
```

```

25         "<ul>"  

26         "<li>Transporte de Cascalho.</li>"  

27         "<li>Fluxo nao Newtoniano na Coluna e Anular.</li>"  

28     "</ul>";  

29  

30     ui ->lbnModulo01 ->setText(buttonText_01);  

31     ui ->btnModulo01 ->setText(""); // Botao fica invisivel para dar  

            lugar ao label formatado  

32     ui ->lbnModulo01 ->setAttribute(Qt::WA_TransparentForMouseEvents)  

            ; // Label nao 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 - Analise de Tensões na Coluna</b><br>"  

39         "<ul>"  

40         "<li>Carga Axial.</li>"  

41         "<li>Colapso e Explosao 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 Po o");  

60     w ->show();  

61 }

```

```
62
63 // Quando o usuario clica no Modulo 2, abre a janela de simulacao
   de perda na tubulacao
64 void JanelaMenu::on_btnModulo02_clicked()
65 {
66     CSimuladorPerdaTubulacao *w = new CSimuladorPerdaTubulacao(this
67         );
68     w->setWindowTitle("SEAPEP - Software Educacional de Engenharia
69         de Po o");
69 }
```

---

Fonte: produzido pelo autor.

# Capítulo 8

## Resultados

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

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

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

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

### 8.1 Metodologia de Validação

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

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

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

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

## 8.2 Casos de Teste

### 8.2.1 Validação da pressão Hidrostática

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

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

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

**Resposta esperada: 136 psig**

$$P_H = 0.052\rho L - P_{formação}$$

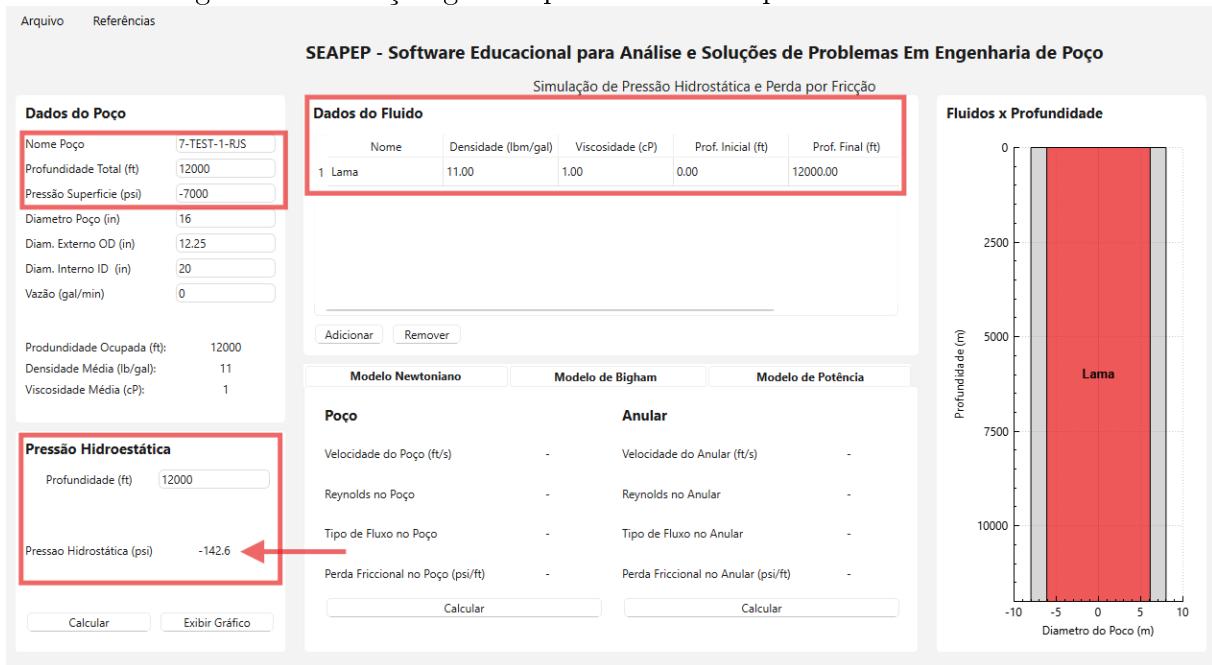
Onde:  $\rho=11$  lbm/gal (densidade do fluido);  $L=12.000$  ft (profundidade total);  $P_{formação} = 7000$  ft (pressão da formação);

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

$$P_H = -136psig$$

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

Figura 8.1: Solução gerada para cálculo da pressão hidrostática



Fonte: Produzido pelo autor.

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

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

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

- Determine o regime de escoamento no tubo de perfuração.

**Resposta esperada: turbulento**

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

**Resposta esperada: 51,3 psi/1.000 ft**

- Determine o regime de escoamento no anular.

**Resposta esperada: turbulento**

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

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

Onde:  $dp_f$  (Perda Friccional);

## Cálculos no anular

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

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

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

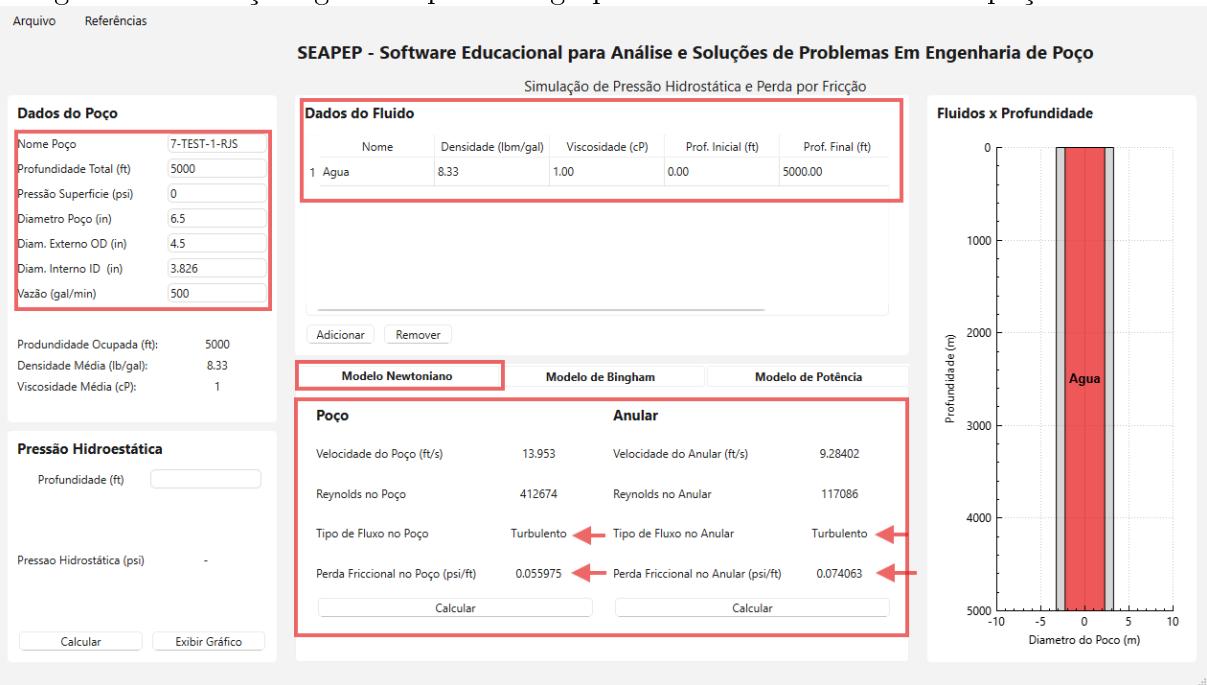
Onde:  $\rho$  = densidade;  $\mu$  = viscosidade;  $N_{re}$  (Número de Reynolds);

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

Onde:  $dp_f$  (Perda Friccional);

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

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



Fonte: Produzido pelo autor.

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

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

Resolva o Exercício 4.40 considerando um fluido plástico de Bingham com densidade de 10 lbm/gal, viscosidade plástica de 25 cP e ponto de escoamento (yield point) de 5 lbf/100 ft<sup>2</sup>. (traduzido, Jr. et al. (1991))

**Resposta esperada:**

**Regime de escoamento: turbulento**

**Perda de pressão no tubo: 132 psi/1.000 ft**

**Perda de pressão no anular: 185 psi/1.000 ft**

#### Cálculos no tubo

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

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

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

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

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

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

### Cálculos no anular

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

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

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

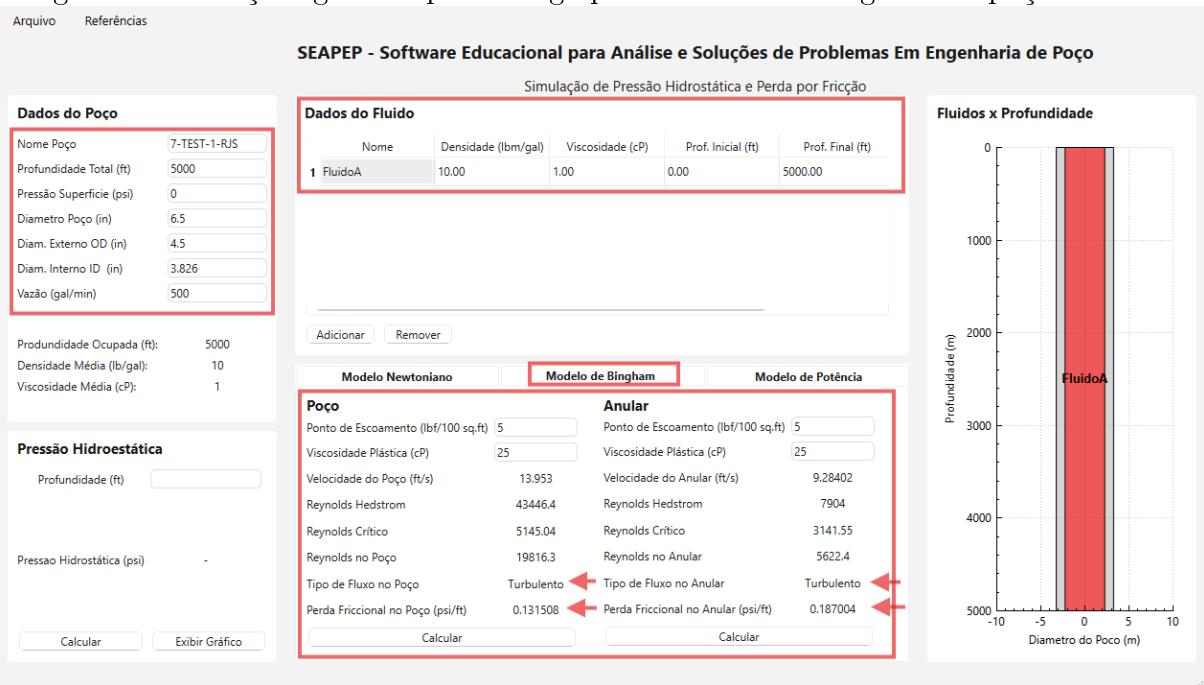
$$N_{re} = \frac{757\rho\bar{v}(d_2 - d_1)}{\mu_p} = \frac{757(10)(9.284)(6.5 - 4.5)}{25} = 5622.39$$

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

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

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

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



Fonte: Produzido pelo autor.

#### 8.2.4 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: turbulent**

**Perda de pressão no tubo: 144 psi/1.000 ft**

**Regime no anular: turbulent**

**Perda de pressão no anular: 205 psi/1.000 ft**

#### Cálculos no tubo

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

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

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

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

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

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

### Cálculos no anular

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

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

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

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

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

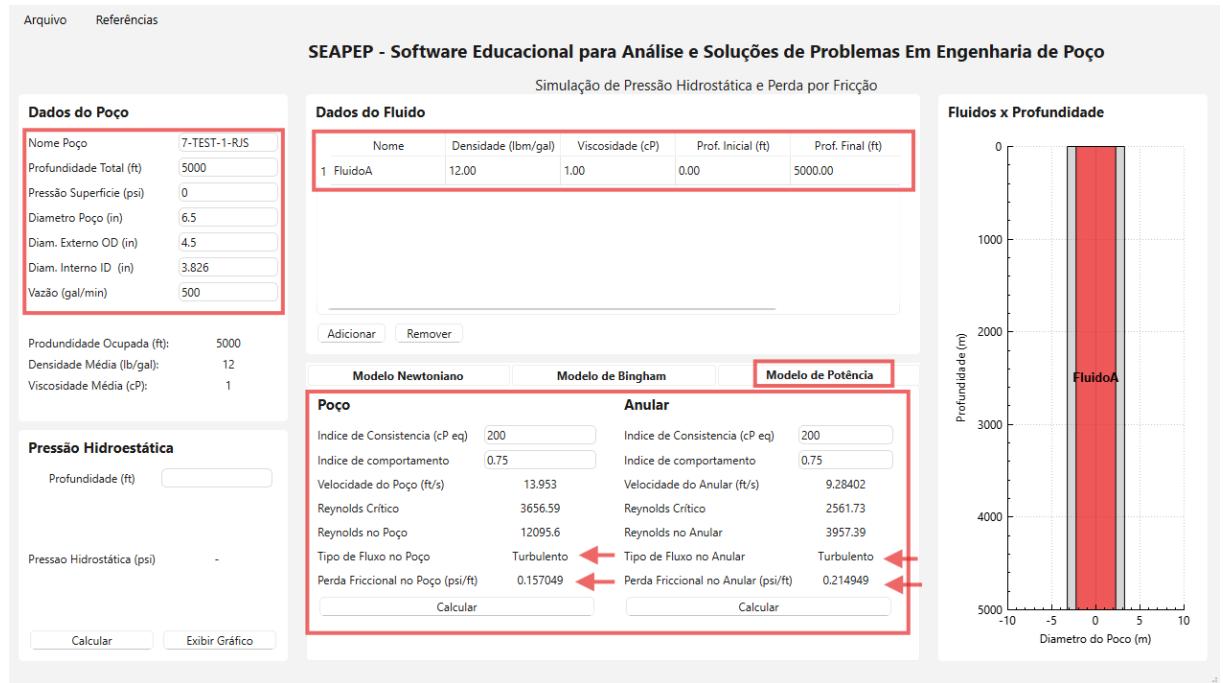
$$\frac{dp_f}{dL} = \frac{f \rho \bar{v}^2}{21.1(d_2 - d_1)} = \frac{0.008(12)(9.284)^2}{21.1(6.5 - 4.5)} = 0.196 \text{psi/ft}$$

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

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

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

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



Fonte: Produzido pelo autor.

### 8.2.5 Validação da variação do comprimento do tubo devido a força pistão

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

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

**Resposta esperada: 3ft**

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

Onde :

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

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

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

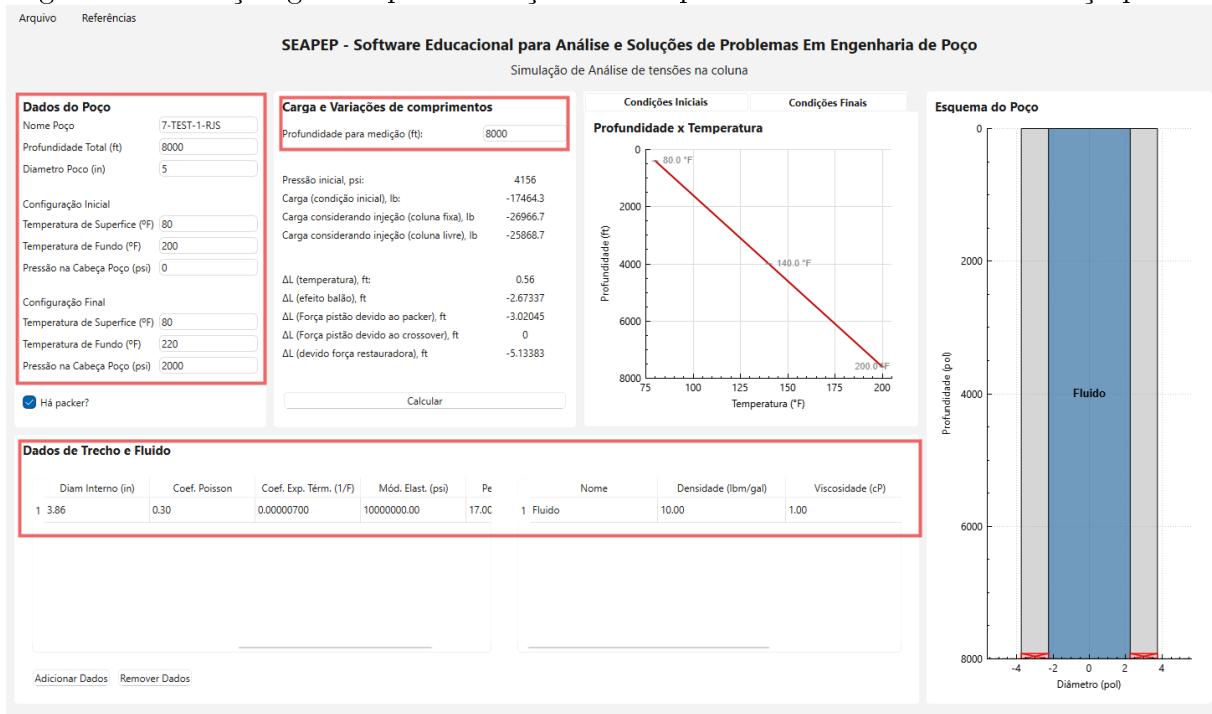
$$A_{in} = \frac{\pi}{4}(d_2^2 - d_1^2) = \frac{\pi}{4}(5^2 - 3.862^2) = 7.92 \text{ in}^2$$

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

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

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

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



### 8.2.6 Validação da variação do comprimento do tubo devido ao efeito balão

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

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

**Resposta esperada: -2.6ft**

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

Utilizandos os dados já calculados na validação acima:

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

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

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

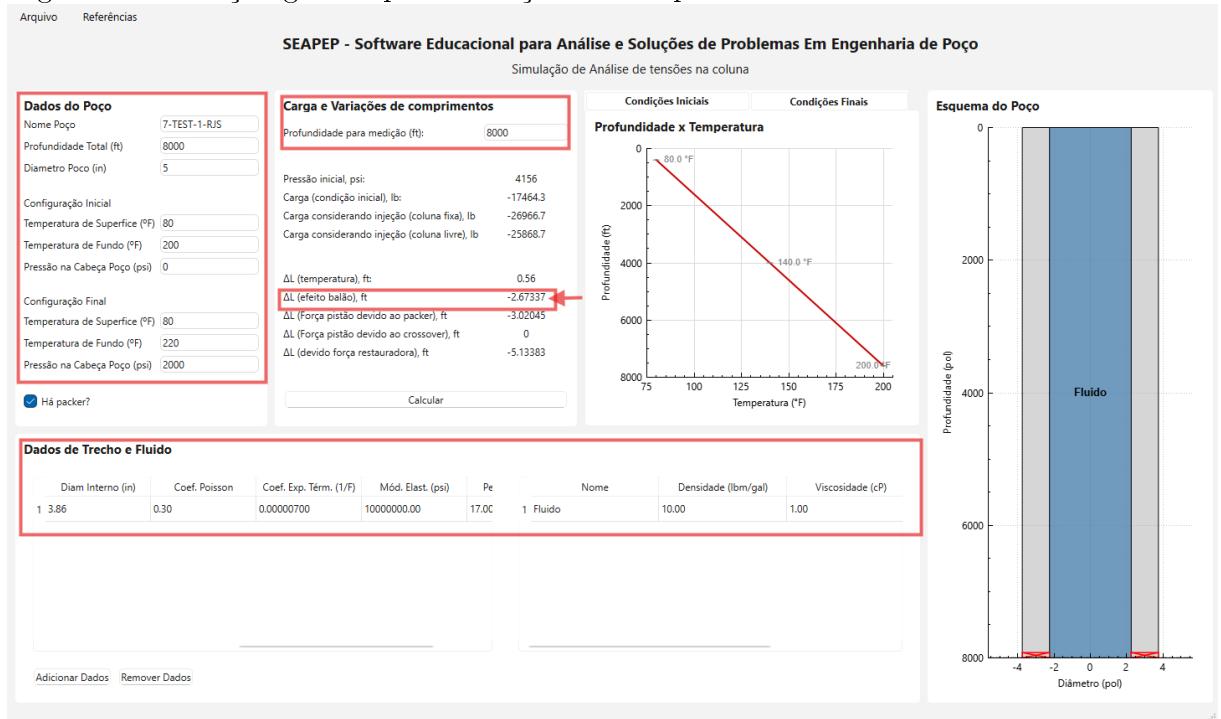
$$A_{in} = \frac{\pi}{4}(d_2^2 - d_1^2) = \frac{\pi}{4}(3.862^2) = 11.71\text{in}^2$$

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

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

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

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



Fonte: Produzido pelo autor.

### 8.2.7 Validação da variação do comprimento do tubo devido ao efeito da temperatura

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

Utilizando o mesmo cenário descrito no item anterior.

**Resposta esperada: 0.5ft**

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

onde:

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

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

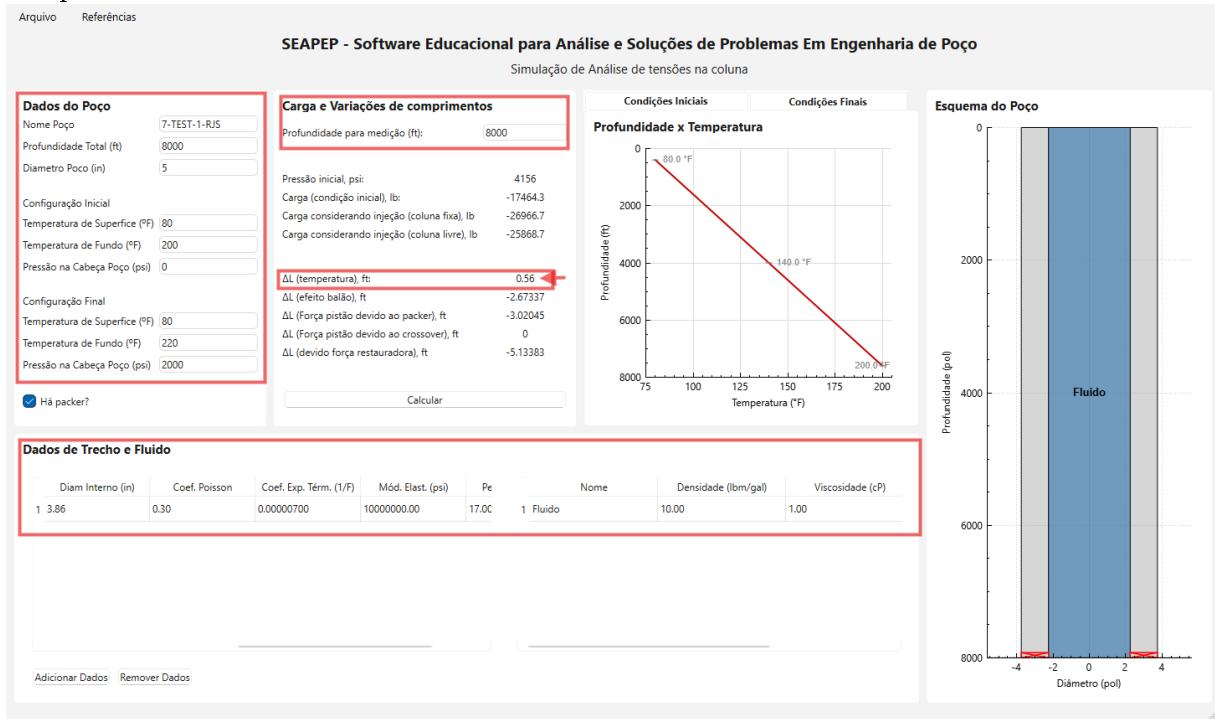
$$\Delta T = 150 - 140 = 10^{\circ}F$$

Logo:

$$\Delta L_t = (7.10^{-6})(8000)(10) = 0.56 \text{ ft}$$

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

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



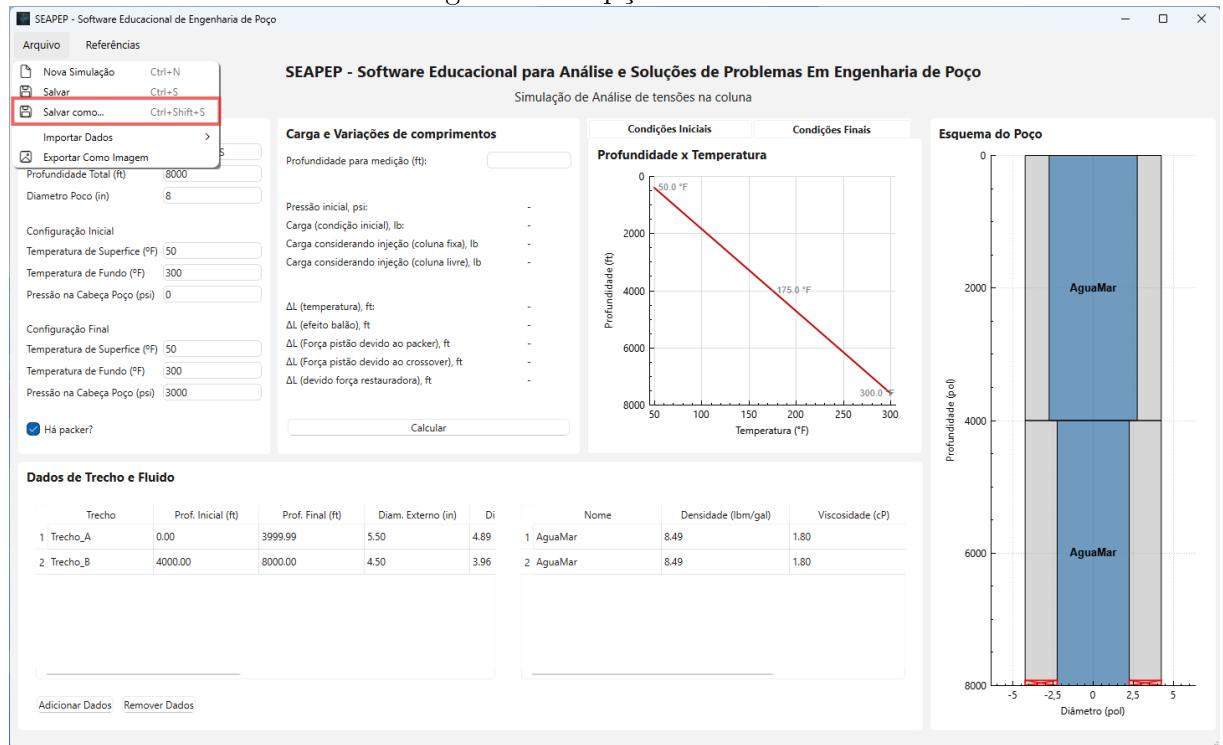
Fonte: Produzido pelo autor.

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

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

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

Figura 8.8: Opção de Salvar



Fonte: Produzido pelo autor.

Figura 8.9: Modelo do Arquivo .dat

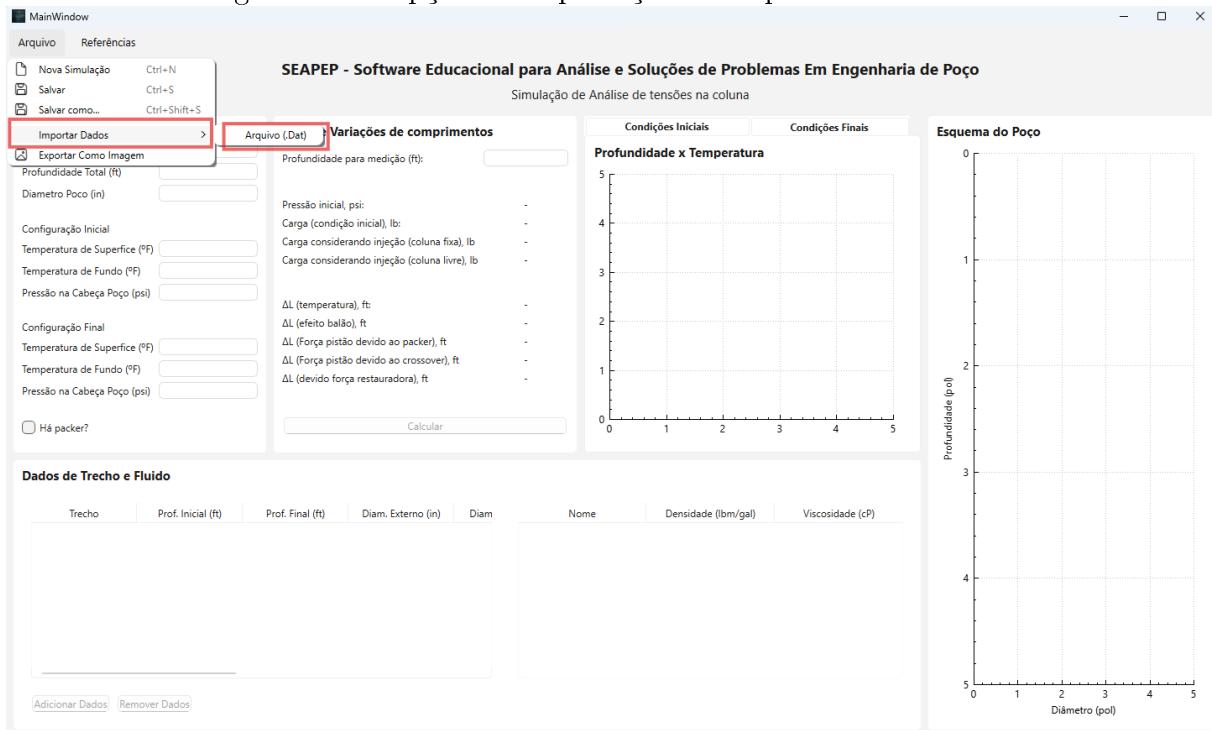
```
# L:\Desktop\Quest1.M002_Nasut
# Arquivo Editar Exibir
# Configuração do Poço
# Nome: Prof_Packer (ft)
# Prof_Packer (ft): 7-TES-01-R35
true

# Configuração dos Fluidos
# Nome_Trecho Prof_Inicial (ft) Prof_Final (ft) Diam_externo (in) Diam_interno (in) Coef_Poisson Coef_Exp_Tens_1(F) Mod_Elast_1 (psi) Peso/unid (lb/ft) Nome_fluido Densidade (lbm/gal) Viscosidade (cP)
Trecho_A 0.00 3999.99 5.50 4.89 0.30 0.00003000 30000000.00 22.00 Aguamar 8.49 1.80
Trecho_B 4000.00 8000.00 4.50 3.96 0.30 0.00003000 35000000.00 15.00 Aguamar 8.49 1.80
```

Fonte: Produzido pelo autor.

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

Figura 8.10: Opção de importação de arquivos no software



Fonte: Produzido pelo autor.

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

## 8.3 Conclusão

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

As comparações envolveram diversos modelos reológicos como: o modelo Newtoniano, o Plástico de Bingham e o modelo da Lei da Potência, além de cálculos de pressão hidrostática, perda de carga por fricção, variações térmicas e efeitos mecânicos complexos, como o efeito balão e deslocamentos axiais ( $\Delta L$ ). 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 10 - Manual do Usuário.

### 9.2 Documentação do desenvolvedor

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

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

Na Figura 9.1 mostra uma imagem da documentação gerada.

Figura 9.1: Logo e documentação do *software*  
**SoftwareEducacionalParaAnaliseESolucaoDeProblemasEmEngenhariaDePoco**

Main Page Classes Files Search

### File List

Here is a list of all files with brief descriptions:

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

Fonte: Produzido pelo autor.

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

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

Main Page Classes Files Search

### CSimuladorPoco.h

Go to the documentation of this file.

```

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

```

Fonte: Produzido pelo autor.

# Capítulo 10

## Manual do usuário

### 10.1 Instalação

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

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

#### 10.1.1 Dependências

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

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

### 10.2 Interface gráfica

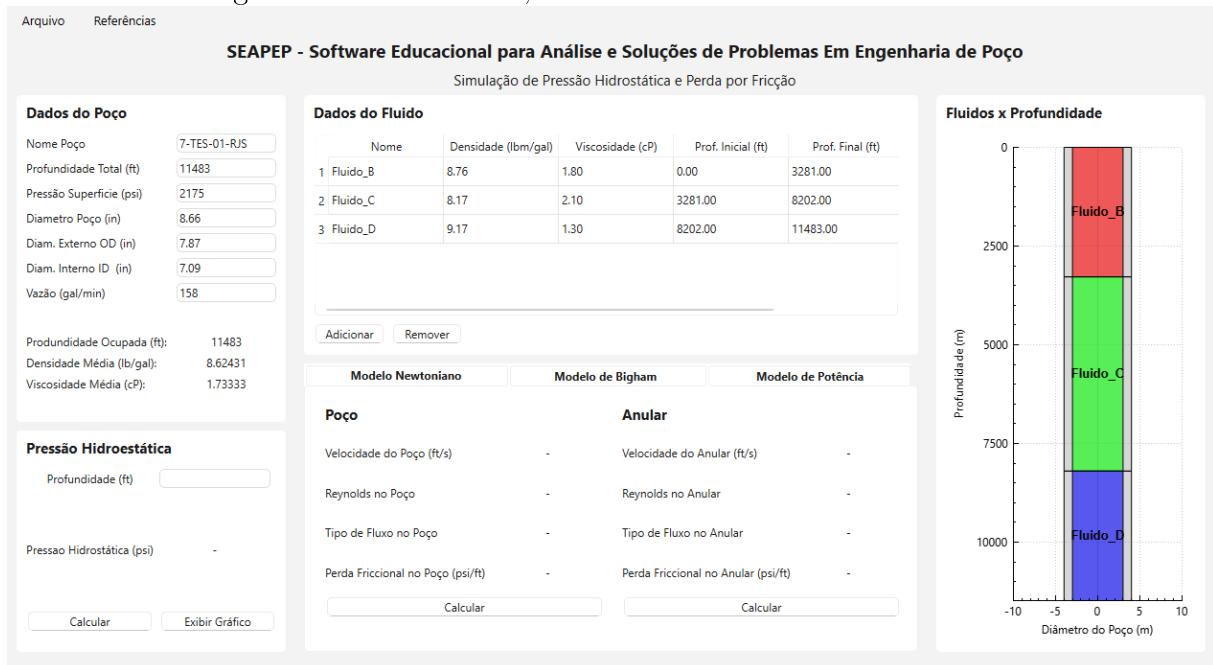
A interface do programa é apresentada nas Figuras: 10.1, 10.2 e 10.3.

Figura 10.1: Versão 1.1, Menu de interface do software



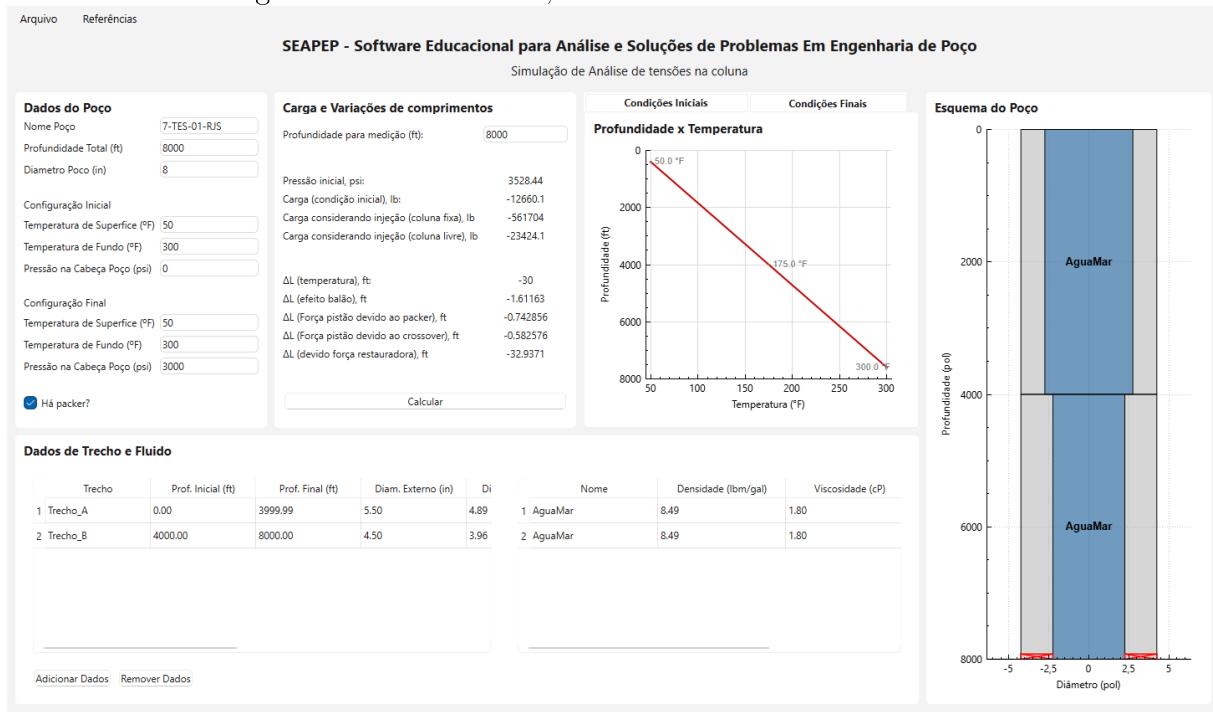
Fonte: Produzido pelo autor.

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



Fonte: Produzido pelo autor.

Figura 10.3: Versão 1.1, interface do modulo 02 software



Fonte: Produzido pelo autor.

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

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

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

## 10.3 Funcionalidades

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

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

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

### 10.3.1 Módulo 1 – Hidráulica de Perfuração

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

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

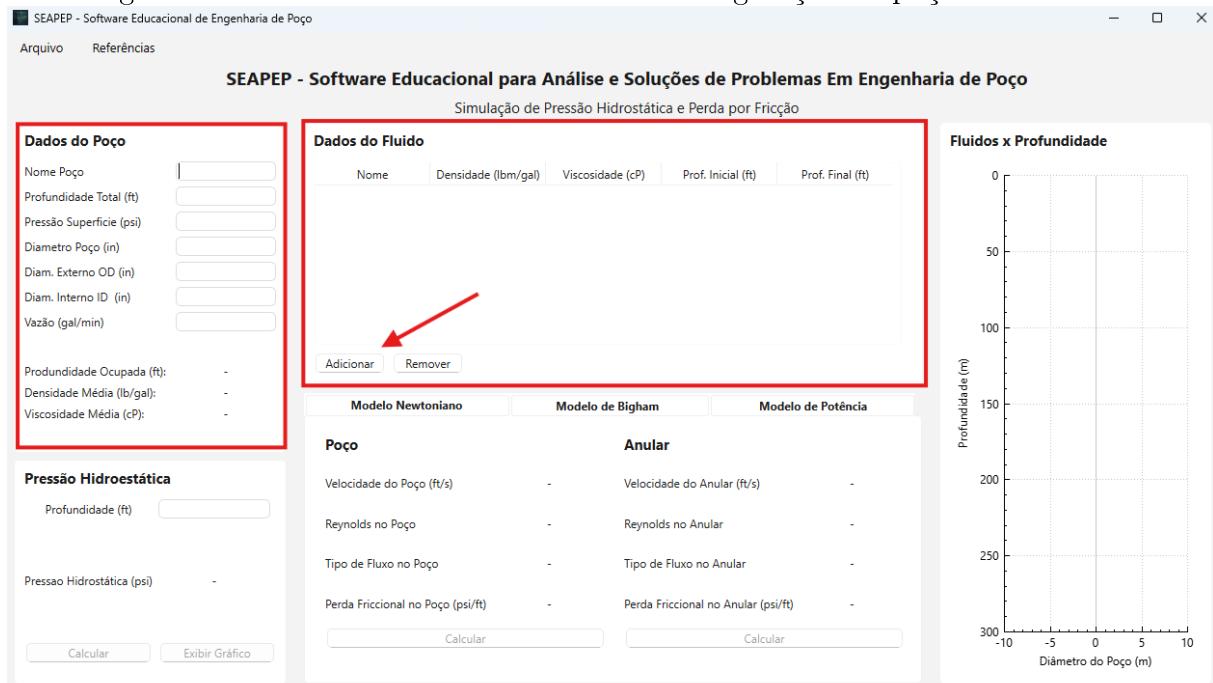
## Configuração Manual via Interface Gráfica

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

- Acesse o menu lateral e selecione a opção Configurar Poço.
- Escolha entre as seguintes funcionalidades:
  - Criar Poço: permite definir propriedades como profundidade total, diâmetro do poço, presença de revestimentos e pressão na superfície.
  - Adicionar Fluido: possibilita configurar fluidos específicos, definindo valores de densidade (lbm/gal) e viscosidade (cP), além de outras propriedades associadas à faixa de atuação por profundidade.

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

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



Fonte: Produzido pelo autor.

## Carregamento de Arquivos de Dados (.dat)

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

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

- A primeira linha representa os dados do poço.

- As linhas subsequentes listam os fluidos, um por linha.

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

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

```

ArquivoPoco.dat
Arquivo Editar Exibir

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

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


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

```

Fonte: Produzido pelo autor.

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

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

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

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

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

```
# 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           0.0000035           0.0000035           15              AguMar        8.51

Ln 11, Col 1 | 1.333 caracteres
```

Fonte: Produzido pelo autor.

# Referências Bibliográficas

- BUENO, André Duarte. 2003. *Programação orientada a objeto com c++*. Novatec. 43
- HALLIDAY, David, & RESNICK, Jearld Walker. 2009. *Fundamentos de física, volume 2: gravitação, ondas e termodinâmica*. 33
- Jr., Adam T. Bourgoyné, Millheim, Keith E., Chenevert, Martim E., & Jr., F. S. Young. 1991. *Applied drilling engineering*. Society of Petroleum Engineers. 17, 29, 30, 31, 32, 35, 38, 148, 149, 150, 152, 154
- Mitchell, Robert F., & Miska, Stefam Z. 2011. *Fundamentals of drilling engineering*. Society of Petroleum Engineers. 17, 32, 33, 34, 38, 39, 40