

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

PROJETO DE ENGENHARIA  
DESENVOLVIMENTO DO SOFTWARE  
SOFTWARE EDUCACIONAL PARA ANÁLISE E SOLUÇÃO DE  
PROBLEMAS EM ENGENHARIA DE POÇO  
DISCIPLINA :Projeto de Software Aplicado a Engenharia  
Setor de Modelagem Matemática Computacional

Versão 1  
NATHAN RANGEL MAGALHÃES  
THAUAN FERREIRA BARBOSA

Prof. André Duarte Bueno

MACAÉ - RJ  
Novembro - 2024

# Listas de Figuras

1.1	Diagrama de caso de uso – Caso de uso geral . . . . .	12
1.2	Diagrama de caso de uso específico: cálculo de pressão hidrostática e densidade . . . . .	13
1.3	Diagrama de caso de uso específico - cálculo da perda de pressão devido a perda por fricção no poço e no anular . . . . .	14
2.1	Diagrama de corpo livre, atuação de forças em um elemento de fluido. . . . .	19
2.2	Coluna composta por fluidos com diferentes características. . . . .	21
2.3	Fluxo laminar de fluido Newtoniano. . . . .	23
2.4	Diagrama de Pacotes . . . . .	29
3.1	Diagrama de classes . . . . .	31
3.2	Diagrama de sequência . . . . .	33
3.3	Diagrama de comunicação . . . . .	34
3.4	Diagrama de máquina de estado . . . . .	34
3.5	Diagrama de atividades . . . . .	35
4.1	Diagrama de componentes . . . . .	37
5.1	Versão 0.1, interface do software . . . . .	39
7.1	Soluções geradas pelo código para Modelo Newtoniano no Poço considerando Regime Laminar . . . . .	68
7.2	Soluções geradas pelo código para Modelo Newtoniano no Poço considerando Regime Turbulento . . . . .	69
7.3	Soluções geradas pelo código para Modelo Newtoniano no Anular considerando Regime Laminar . . . . .	70
7.4	Soluções geradas pelo código para Modelo Newtoniano no Anular considerando Regime Turbulento . . . . .	71
7.5	Soluções geradas pelo código para Modelo Plástico de Bingham no Poço considerando Regime Laminar . . . . .	72
7.6	Soluções geradas pelo código para Modelo Plástico de Bingham no Poço considerando Regime Turbulento . . . . .	73

7.7	Soluções geradas pelo código para Modelo Plástico de Bingham no Anular considerando Regime Laminar . . . . .	74
7.8	Soluções geradas pelo código para Modelo Plástico de Bingham no Anular considerando Regime Turbulento . . . . .	75
7.9	Soluções geradas pelo código para Modelo Lei de Potência no Poço considerando Regime Laminar . . . . .	76
7.10	Soluções geradas pelo código para Modelo Lei de Potência no Poço considerando Regime Turbulento . . . . .	76
7.11	Soluções geradas pelo código para Modelo Lei de Potência no Anular considerando Regime Laminar . . . . .	77
7.12	Soluções geradas pelo código para Modelo Lei de Potência no Anular considerando Regime Turbulento . . . . .	78
7.13	Soluções geradas pelo código para pressão hidrostática no fundo do poço .	79
8.1	Logo e documentação do <i>software</i> . . . . .	81
8.2	Código fonte da classe CSimuladorTemperatura, no Doxygen . . . . .	81

# **Lista de Tabelas**

1.1	Características básicas do programa . . . . .	9
1.2	Caso de uso 1 . . . . .	11
7.1	Configuração do poço turbulento . . . . .	66
7.2	Configuração do poço turbulento . . . . .	67
7.3	Configuração do poço laminar . . . . .	67
7.4	Configuração do poço turbulento . . . . .	67

# Sumário

<b>1</b>	<b>Concepção</b>	<b>8</b>
1.1	Nome do Sistema/Produto . . . . .	8
1.2	Especificação . . . . .	9
1.3	Requisitos . . . . .	9
1.3.1	Requisitos funcionais . . . . .	9
1.3.2	Requisitos não funcionais . . . . .	10
1.4	Casos de Uso . . . . .	10
1.4.1	Diagrama de caso de uso geral . . . . .	10
1.4.2	Diagrama de caso de uso específico . . . . .	11
<b>2</b>	<b>Elaboração</b>	<b>16</b>
2.1	Análise de domínio . . . . .	16
2.2	Formulação teórica . . . . .	18
2.2.1	Hidráulica de perfuração . . . . .	18
2.2.2	Pressão Hidrostática . . . . .	18
2.2.3	Pressão hidrostática em colunas com mais de um tipo de fluido . . .	20
2.2.4	Densidade Equivalente . . . . .	22
2.2.5	Modelos reológicos de fluidos de perfuração . . . . .	22
2.2.6	Perda de pressão friccional em um tubo de perfuração . . . . .	24
2.2.7	Perda de pressão friccional em um anular . . . . .	26
2.3	Identificação de pacotes – assuntos . . . . .	28
2.4	Diagrama de pacotes – assuntos . . . . .	28
<b>3</b>	<b>AOO – Análise Orientada a Objeto</b>	<b>31</b>
3.1	Diagramas de classes . . . . .	31
3.1.1	Dicionário de classes . . . . .	32
3.2	Diagrama de seqüência – eventos e mensagens . . . . .	32
3.2.1	Diagrama de sequência geral . . . . .	32
3.3	Diagrama de comunicação – colaboração . . . . .	33
3.4	Diagrama de máquina de estado . . . . .	34
3.5	Diagrama de atividades . . . . .	35

<b>4 Projeto</b>	<b>36</b>
4.1 Projeto do sistema . . . . .	36
4.2 Diagrama de componentes . . . . .	36
<b>5 Ciclos de Planejamento/Detalhamento</b>	<b>38</b>
5.1 Versão 0.1 - Uso modo terminal e Gnuplot para saída de gráfico . . . . .	38
<b>6 Ciclos Construção - Implementação</b>	<b>40</b>
6.1 Versão 0.1 - Código fonte - . . . . .	40
<b>7 Resultados</b>	<b>66</b>
7.1 Propriedades da simulação . . . . .	66
7.1.1 Configurações para o regime turbulento . . . . .	66
7.1.2 Configurações para o regime laminar . . . . .	67
7.2 Testes . . . . .	67
7.2.1 Validação da Perda de Fricção Pelo Modelo Newtoniano no Poço . .	67
7.2.2 Validação da Perda de Fricção Pelo Modelo Newtoniano no Anular	69
7.2.3 Validação da Perda de Fricção Pelo Modelo Bingham no Poco . .	71
7.2.4 Validação da Perda de Fricção Pelo Modelo Bingham no Anular .	73
7.2.5 Validação da Perda de Fricção Pelo Modelo de Potência no Poco .	75
7.2.6 Validação da Perda de Fricção Pelo Modelo de Potência no Anular.	77
7.2.7 Validação da Pressão hidrostática no Fundo do Poço . . . . .	78
<b>8 Documentação</b>	<b>80</b>
8.1 Documentação do usuário . . . . .	80
8.2 Documentação do desenvolvedor . . . . .	80
<b>Referências Bibliográficas</b>	<b>82</b>

0 - Concepção

# Capítulo 1

## Concepção

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

### 1.1 Nome do Sistema/Produto

O software chamado de Software Educacional Para Análise e Solução de Problemas em Engenharia de Poço, foi desenvolvido utilizando uma linguagem de programação em C++ com paradigma de orientação a objeto, sendo capaz de calcular a pressão hidrostática em qualquer ponto do poço, calcular as propriedades médias dos fluidos que ocupam o poço como viscosidade e densidade, determinar o fluxo de escoamento podendo ser laminar ou turbulento e utilizar 3 modelos reológicos (modelo Newtoniano, modelo Plástico de Bingham e modelo Lei de Potencias) para calcular a queda de pressão provocada pela perda de carga friccional.

O software permite ao usuário a interação com gráficos e a armazenagem de informações na forma de arquivo PDF.

A tabela 1.1 apresenta as características do software desenvolvido.

Tabela 1.1: Características básicas do programa

<b>Nome</b>	Software Educacional Para Análise e Solução de Problemas em Engenharia de Poço
<b>Componentes principais</b>	Banco de dados com métodos e propriedades da engenharia de poço. Algoritmo de aproximação de resultados. Interface gráfica para o plot de resultados. Saída gráfica e em pdf.
<b>Missão</b>	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.

## 1.2 Especificação

Deseja-se desenvolver um software com interface amigável que possibilite o usuário escolher de forma livre a funcionalidade de deseja utilizar naquele momento. Os cálculos realizados pelos software se baseiam em equações analíticas desenvolvidas na disciplina de Engenharia de poço.

Inicialmente o usuário precisará configurar as propriedades do poço, ação que pode ser feita de duas formas, tanto a partir da leitura de um arquivo .dat ou a partir do input individual das propriedade no terminal. Após configurar o poço o usuário poderá exibir as propriedades do poço, calcular a pressão hidrostática no fundo do poço ou selecionar uma profundidade na qual essa pressão será calculada, plotar gráficos de perfis como o gráfico da profundidade pela densidade e por fim poderá calcular a perda de pressão devido a perda de carga por fricção.

Ao iniciar o cálculo da perda de pressão é possível determinar o tipo de escoamento tanto no poço quanto no anular, e para calcular a perda de pressão o usuário poderá selecionar entre três modelos reológicos, o modelo Newtoniano, o modelo de Plástico de Bingham e o modelo de Lei de Potências.

## 1.3 Requisitos

### 1.3.1 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.
--------------	---

<b>RF-02</b>	O usuário deverá ter liberdade para alterar as propriedades reológicas do poço/fluido.
<b>RF-03</b>	Deve permitir a exportação de simulações.
<b>RF-04</b>	Deve permitir cenários de simulação baseado em diferentes modelos teóricos.
<b>RF-05</b>	O usuário deve ter liberdade para adicionar ou retirar simplificações das premissas do modelo.
<b>RF-06</b>	O usuário poderá visualizar seus resultados em um gráfico. O gráfico poderá ser salvo como imagem.

### 1.3.2 Requisitos não funcionais

<b>RNF-01</b>	Suas primeiras versões deve suportar apenas o SO Windows
<b>RNF-02</b>	Plotagem de diferentes gráficos para representação do estudo analisado durante a simulação através do GnuPlot.
<b>RNF-03</b>	Possibilitar exportação dos estudos realizados em saída de texto.
<b>RNF-04</b>	Apresentar interface em comandos terminais.

## 1.4 Casos de Uso

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

### 1.4.1 Diagrama de caso de uso geral

O diagrama de caso de uso geral da Figura 1.1 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 apresentados na Tabela 1.2.

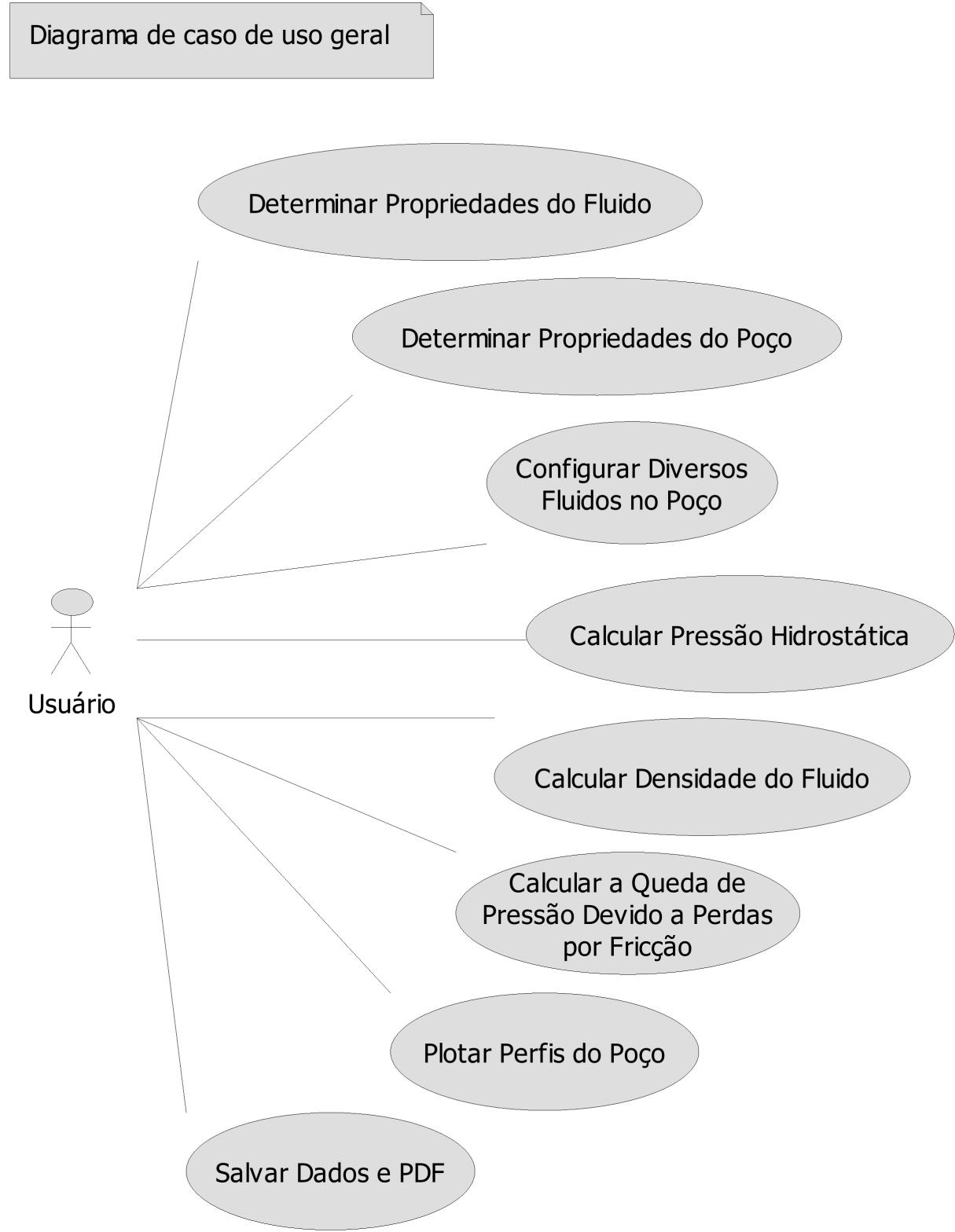
Tabela 1.2: Caso de uso 1

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. Determinar as propriedades do fluido</li><li>2. Determinar propriedades do poço</li><li>3. Configurar diversos 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 Devida a Perdas por Fricção</li><li>6. Plotar perfis de poço</li><li>7. Salvar dados em PDF</li></ol>

#### 1.4.2 Diagrama de caso de uso específico

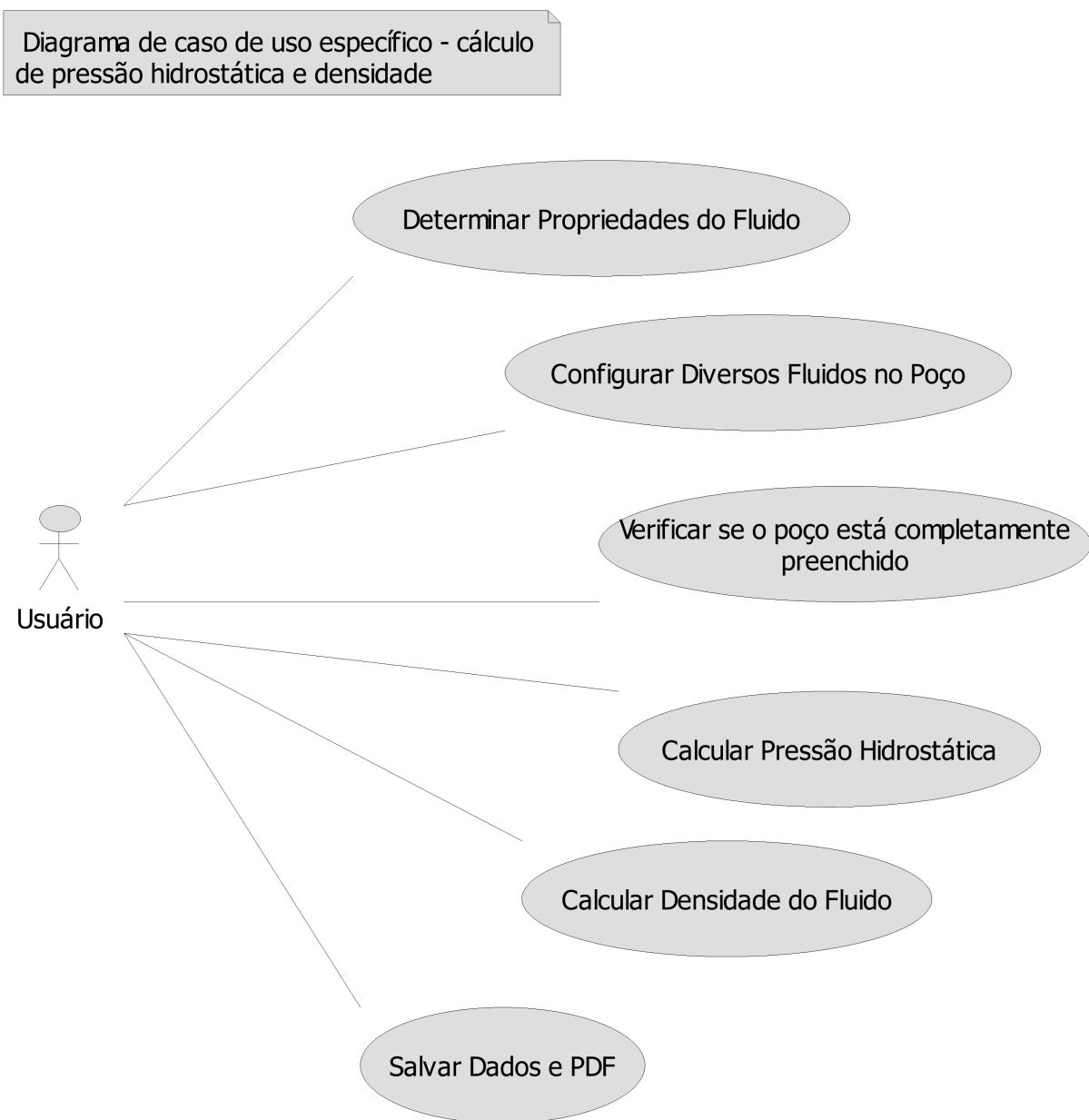
O caso de uso específico da Figura 1.2 mostra o cenário onde o usuário deseja calcular a pressão hidrostática e a densidade dos fluidos configurados no poço.

Figura 1.1: Diagrama de caso de uso – Caso de uso geral



Fonte: Produzido pelo autor.

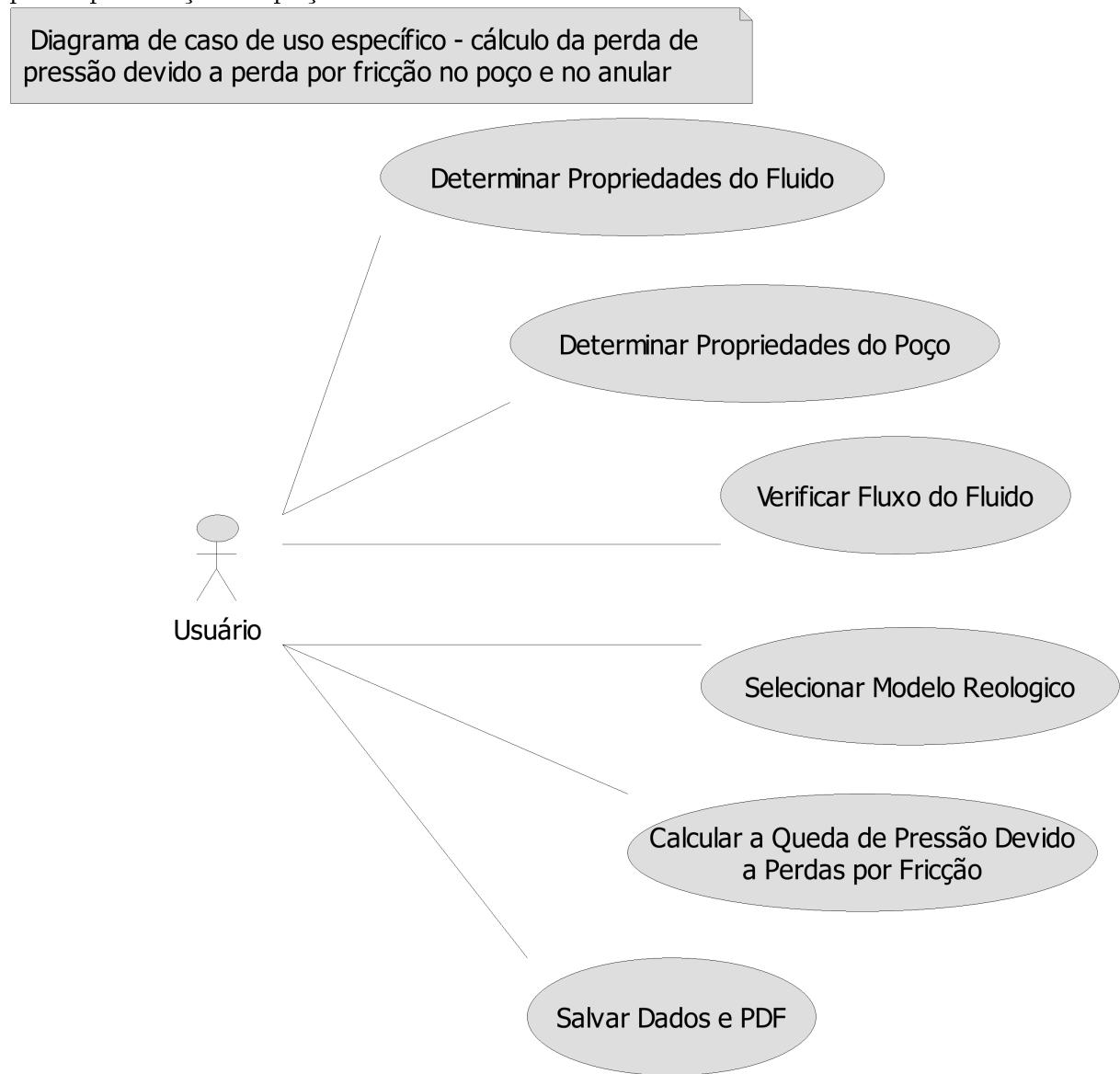
Figura 1.2: Diagrama de caso de uso específico: cálculo de pressão hidrostática e densidade



Fonte: Produzido pelo autor.

O caso de uso específico da Figura 1.3 mostra o cenário onde o usuário deseja calcular a perda de pressão devido a perda por fricção no poço e no anular.

Figura 1.3: 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.

1 - Elaboração

# Capítulo 2

## Elaboração

Neste capítulo será apresentada a elaboração do simulador, que envolve o desenvolvimento teórico, as equações analíticas, identificação dos pacotes e algoritmos adicionais relacionados ao software a ser desenvolvido.

### 2.1 Análise de domínio

O tópico análise de domínio é uma parte fundamental da elaboração de um projeto, na qual se faz necessário entender e delimitar os conceitos principais para a construção do simulador.

O presente projeto está relacionado a 5 conceitos fundamentais:

#### 1. Mecânica dos fluidos:

A mecânica dos fluidos, especialmente no contexto da engenharia de perfuração, envolve o estudo e o controle de como fluidos se comportam sob diferentes condições de pressão, velocidade e temperatura, além da interação com materiais sólidos (como cascalho e cimento). O sucesso na perfuração de poços de petróleo depende diretamente da capacidade de lidar com a dinâmica dos fluidos de perfuração, que são essenciais para estabilizar o poço, controlar pressões, remover cascalhos e cimentar o revestimento de forma eficiente. Nesse projeto iremos determinar as pressões de fluidos em duas condições de poço: quando a coluna e o fluido estão em repouso e quando fluidos são bombeados em uma coluna fixa.

#### 2. Mecânica das rochas:

A mecânica das rochas é crucial na engenharia de perfuração porque lida com o comportamento e as propriedades das formações rochosas atravessadas pelo poço. Durante a perfuração, a interação entre as tensões in situ e as propriedades das rochas impacta diretamente a estabilidade do poço, a capacidade de perfurar com eficiência e a integridade do revestimento. Entender a mecânica das rochas ajuda a prever e mitigar problemas como o colapso do poço, fraturas indesejadas e falhas

estruturais nas rochas. O comportamento das rochas sob compressão, cisalhamento, e outras tensões precisa ser cuidadosamente estudado para garantir que o poço permaneça estável, especialmente em formações frágeis ou propensas à fraturação.

### 3. Equações analíticas:

As equações analíticas desempenham um papel essencial na engenharia de perfuração, pois fornecem modelos matemáticos que permitem prever e controlar diversos parâmetros operacionais, como pressões, tensões, fluxo de fluidos e comportamento das rochas. O uso de equações analíticas permite aos engenheiros desenvolver soluções precisas e rápidas para problemas complexos, sem a necessidade de depender exclusivamente de simulações numéricas. Na perfuração, equações como a Lei de Darcy, para fluxo de fluidos em meios porosos, ou a equação de Bernoulli, para energia de fluxo em sistemas de fluidos, ajudam a modelar a circulação de lama de perfuração, prever perdas de fluido e estimar a pressão hidrostática necessária para manter a estabilidade do poço. Outro exemplo são as equações de Navier-Stokes, que são usadas para descrever o comportamento de fluidos complexos como os utilizados na lama de perfuração. Além disso, equações analíticas podem ser usadas para estimar as tensões nas paredes do poço, o que é essencial para prever fraturas ou falhas no revestimento, e calcular a pressão de poro, que deve ser cuidadosamente controlada para evitar blowouts.

### 4. Programação:

O paradigma de programação orientada a objetos (POO) é amplamente utilizado no desenvolvimento de grandes softwares, especialmente pela sua capacidade de organizar o código em estruturas modulares e reutilizáveis. POO se baseia em conceitos como classes, objetos, herança, encapsulamento, polimorfismo e abstração, permitindo que os problemas sejam divididos em partes menores e mais gerenciáveis. Isso facilita a manutenção, evolução e escalabilidade de sistemas complexos, como os usados em engenharia de perfuração, simulações e gestão de dados de poços. C++ é uma das linguagens mais populares nesse paradigma, especialmente em áreas que exigem alto desempenho e controle eficiente de recursos, como a perfuração de poços e a simulação geológica. Além de suportar POO, o C++ é conhecido por sua rapidez, principalmente por permitir o gerenciamento manual de memória e por sua rica biblioteca padrão e bibliotecas de terceiros, como Qt (para interfaces gráficas) e QCustomPlot (para gráficos científicos).

### 5. Modelagem Gráfica:

A modelagem gráfica é um componente vital em diversas áreas da engenharia, especialmente em projetos de perfuração e exploração de petróleo, pois permite a visualização e análise de dados complexos de forma intuitiva e acessível. Através

de representações gráficas, como gráficos 2D e 3D, mapas de subsuperfície, simulações visuais de poços e campos petrolíferos, os engenheiros podem tomar decisões mais informadas sobre a operação e planejamento. As bibliotecas gráficas como Qt (que integra facilmente com C++) são amplamente utilizadas para criar gráficos científicos e visualizações de dados complexos, como perfis de pressão, porosidade e velocidades de ondas P e S nas formações geológicas.

## 2.2 Formulação teórica

### 2.2.1 Hidráulica de perfuração

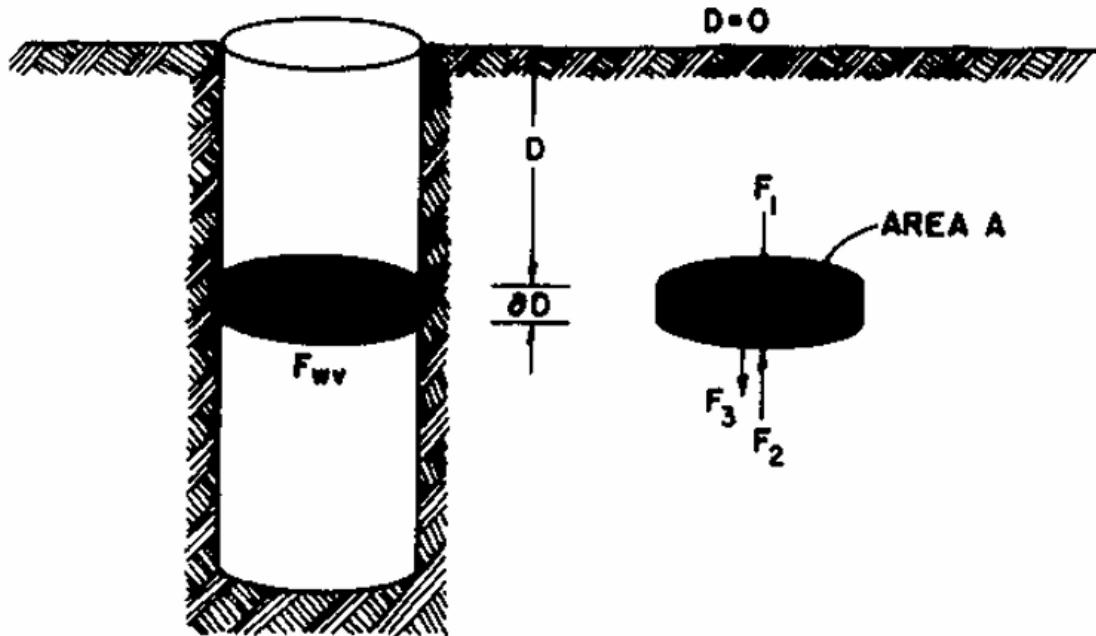
Na engenharia de perfuração um fluido de perfuração tem três funções principais: transportar cascalho, prevenir o influxo de fluidos e manter a estabilidade do poço. Para que possa cumprir tais funções o fluido depende do seu fluxo na tubulação e das pressões associadas a esse fluxo. Para que um engenheiro poça formular o melhor fluido de perfuração para cada situação específica ele deve ser capaz de prever as pressões e fluxos de fluidos no poço.

Os fluidos de perfuração podem ser bem variados em termos de composição e propriedades indo desde fluidos incompressíveis como a água até fluidos muito compressíveis como a espuma. O simulador se propõem a resolver dois tipos de problema, o primeiro deles sendo problemas estáticos que envolvem o cálculo de pressão hidrostática e o segundo deles com a movimentação de fluidos pelo tubo.

### 2.2.2 Pressão Hidrostática

A pressão hidrostática é a variação da pressão com a profundidade em uma coluna de fluido, que normalmente é mais facilmente calculada em condições de poço estático. Essa pressão pode ser deduzida considerando o diagrama de corpo livre mostrado na Figura 2.1.

Figura 2.1: Diagrama de corpo livre, atuação de forças em um elemento de fluido.



A partir dessa dedução chegamos a Equação (2.1) a seguir em unidades oil field, onde  $dp$  é a variação de pressão  $dZ$  é a variação de profundidade e  $\rho$  é a densidade do fluido.

$$\frac{dp}{dZ} = 0.05195\rho \quad (2.1)$$

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

### Fluidos incompressíveis

Sabemos que alguns fluidos usados como lama de perfuração tem um comportamento aproximadamente incompressível, como por exemplo o uso de água salgada, nesses casos a compressibilidade do fluido para baixas temperaturas pode ser desprezada e o peso específico pode ser considerado constante com a profundidade. De forma que a partir da integração da Equação (2.1) podemos chegar na equação hidrostática para fluidos incompressíveis:

$$p = 0.05195\rho Z + p_0 \quad (2.2)$$

Onde  $p_0$ , que é a constante de integração, é igual a pressão na superfície. Uma importante aplicação para essa equação é determinar a densidade correta de um fluido de perfuração, de forma que o mesmo seja capaz de evitar o influxo de fluidos da formação para o poço, evitando dessa forma kiks ou blowouts, além de não causar fraturas na formação que poderia provocar uma perda de circulação de fluido que também é indesejada.

### Fluidos compressíveis

Em muitas operações temos a presença de gás em algum momento da perfuração ou completação, podendo ser injetado ou fluir de alguma formação. Calcular a pressão hidrostática de um coluna de gás estática é um tanto quanto mais complicado devido ao fato da compressibilidade fazer com que a densidade do gás mude com a variação de pressão. O comportamento do gás é modelado utilizando a equação do gás real:

$$p = \rho z \frac{RT}{M} \quad (2.3)$$

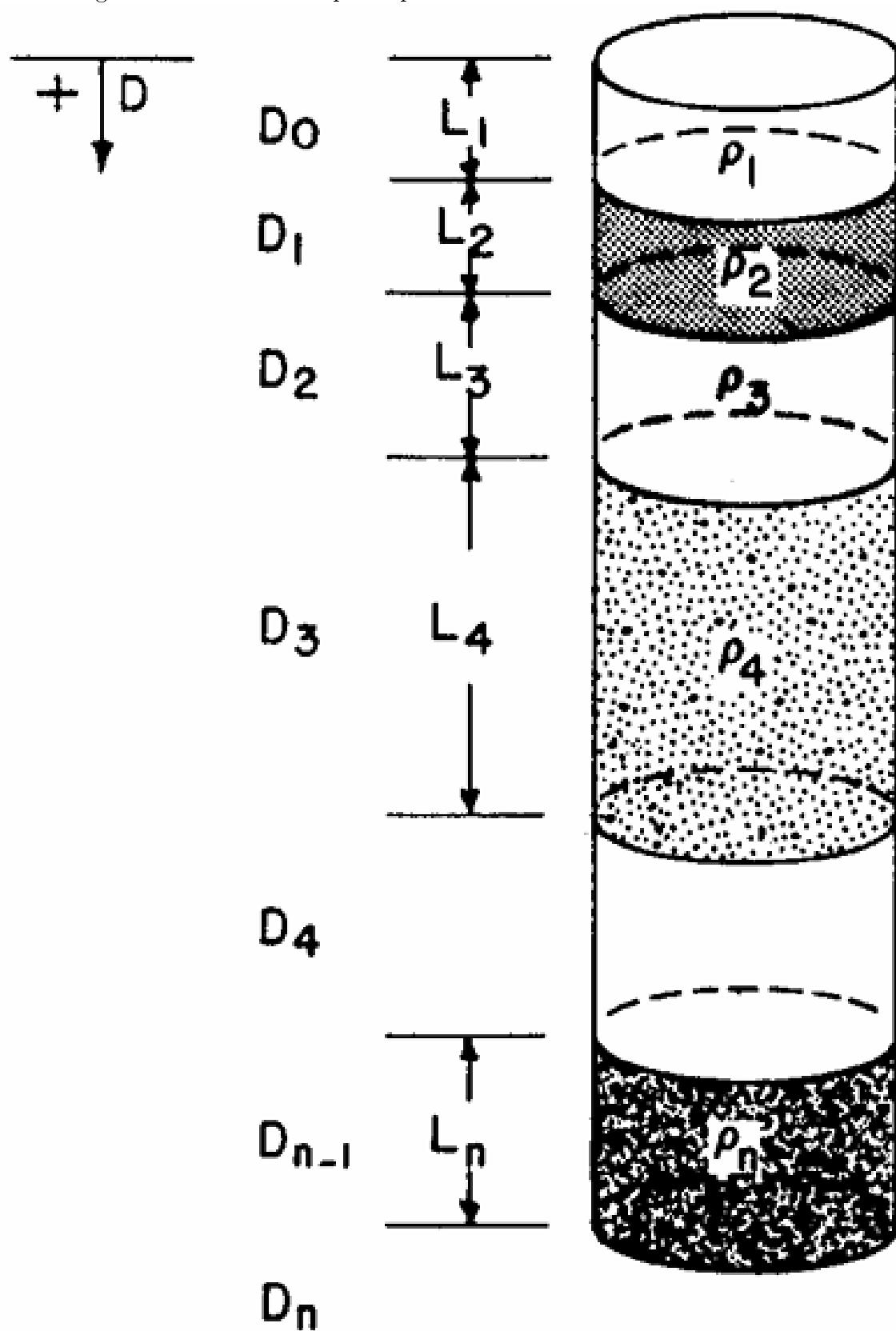
Realizando a combinação da equação da pressão hidrostática para fluidos incompressíveis e da equação do gás real chegamos a seguinte equação da pressão hidrostática para fluidos compressíveis:

$$p = p_0 \exp\left(\frac{M \Delta Z}{1544 \bar{z} T}\right) \quad (2.4)$$

#### 2.2.3 Pressão hidrostática em colunas com mais de um tipo de fluido

Outra situação muito comum durante a perfuração é a existência de seções com diferentes densidades de fluidos na coluna. Para se calcular a pressão hidrostática nesse tipo de situação precisamos determinar a variação de pressão separadamente para cada seção, como na Figura 2.2.

Figura 2.2: Coluna composta por fluidos com diferentes características.



Em geral a pressão em qualquer profundidade  $Z$  pode ser calculada por meio da equação:

$$p = p_0 + g \sum p_i (Z_i - Z_{i-1}) + g\rho_n (Z_i - Z_{i-1}) \quad (2.5)$$

### 2.2.4 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:

$$\rho_e = \frac{p}{0.05195Z} \quad (2.6)$$

A densidade da lama equivalente sempre deve ser calculada utilizando uma profundidade de referência específica.

### 2.2.5 Modelos reológicos de fluidos de perfuração

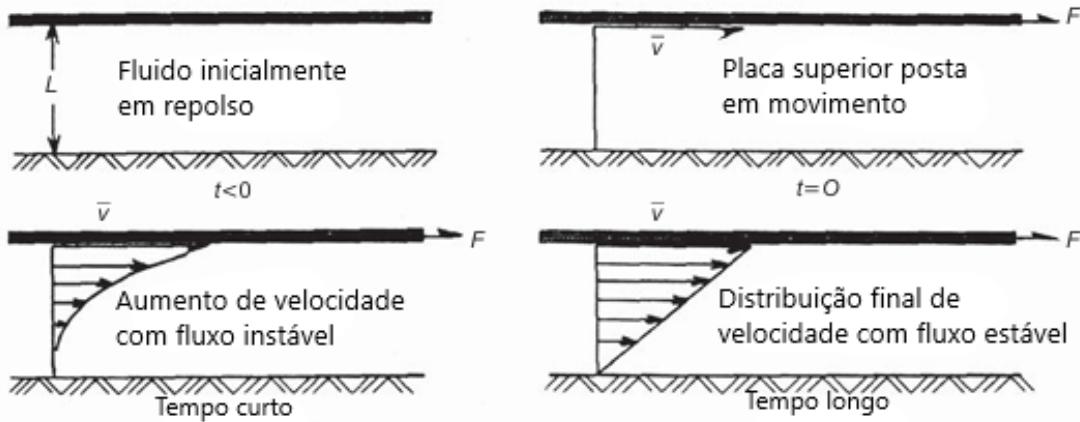
Durante o processo de perfuração de um poço muitas vezes forças viscoelásticas extremamente grandes precisam ser vencidas para que o fluido de perfuração se movea pelos conduítes longos e finos utilizados nesse processo, dessa forma se faz essencial a análise da perda de pressão por atrito. Na maioria dos casos as propriedades elásticas dos fluidos de perfuração e seus efeitos durante o fluxo de um poço são desprendíveis, sendo consideradas para o cálculo apenas as forças viscosas. Entretanto com o avanço tecnológico lamas cada vez mais complexas estão sendo formuladas de forma que os testes devem considerar as propriedades elásticas da deformação que ocorre durante o fluxo.

Para tal, se faz necessária uma descrição matemática e desenvolvimento de equações para a perda por atrito, de forma que modelos reológicos são geralmente utilizados por engenheiros de perfuração para aproximar o comportamento de um fluido, nesse projeto abordaremos três modelos sendo eles o modelo newtoniano, o modelo plástico de Bingham e o modelo de lei de potências. É importante ressaltar a existência de outros modelos que podem futuramente ser acrescentados no aprimoramento desse projeto.

#### 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 2.3.

Figura 2.3: Fluxo laminar de fluido Newtoniano.



Ao colocar a placa superior inicialmente em repouso em um movimento na direção  $x$  com uma velocidade constante  $v$  por um tempo suficiente, percebemos que uma força  $F$  constante é necessária para manter a placa superior em movimento, a magnitude dessa força pode ser determinada por:

$$\frac{F}{A} = \mu \frac{\nu}{L} \quad (2.7)$$

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

$$\tau = \frac{F}{A} \quad (2.8)$$

A taxa de cisalhamento é expressa como o gradiente da velocidade  $\frac{v}{L}$ :

$$\dot{\gamma} = \frac{d\nu}{dL} \approx \frac{\nu}{L} \quad (2.9)$$

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.

### Modelo de Fluido Newtoniano

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

$$\tau = \mu \dot{\gamma} \quad (2.10)$$

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 sera dobrada. Os principais fluidos newtonianos são água, gás e salmouras, fluidos muito comuns na engenharia de poço.

A relação linear descrita pela Equação (2.10) 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.

### Fluidos Plásticos de Bingham

O modelo plástico de Bingham (Bingham 1922) pode ser definido como:

$$\tau = \tau_y + \mu_p \dot{\gamma} \quad (2.11)$$

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. 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 é chamada de viscosidade plástica.

### Fluidos de Lei de Potência

O modelo de lei de potência (Ostwald 1925) pode ser definido como:

$$\tau = K \dot{\gamma}^n \quad (2.12)$$

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, e o parâmetro  $n$  é chamado de expoente da lei de potência ou índice de comportamento do fluxo.

### 2.2.6 Perda de pressão friccional em um tubo de perfuração

A perda de pressão friccional durante a circulação de fluidos em operações de perfuração pode ser calculada através de diferentes modelos de fluido. O primeiro paço 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:

$$N_{re} = \frac{928\rho\bar{v}d}{\mu} \quad (2.13)$$

Onde  $\rho$  é densidade efetiva total,  $d$  é o diâmetro interno do poço,  $\mu$  é a viscosidade efetiva total e  $\bar{v}$  é a velocidade média que pode ser obtida pela seguinte equação:

$$\bar{v} = \frac{q}{2.448d^2} \quad (2.14)$$

Onde  $q$  é a vazão do poço.

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.

Para fluxo laminar:

$$\frac{dp_f}{dL} = \frac{\mu\bar{v}}{1500d^2} \quad (2.15)$$

Para o fluxo turbulento:

$$\frac{dp_f}{dL} = \frac{f\rho\bar{v}^2}{25.8d} \quad (2.16)$$

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

$$N_{re} = \frac{928\rho\bar{v}d}{\mu_p} \quad (2.17)$$

Onde  $\mu_p$  é chamado de viscosidade plástica.

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 pela seguinte fórmula:

$$N_{rec} = \frac{1 - \frac{4}{3} \left( \frac{\tau_y}{\tau_w} \right) + \frac{1}{3} \left( \frac{\tau_y}{\tau_w} \right)^4}{8 \left( \frac{\tau_y}{\tau_w} \right)} N_{He} \quad (2.18)$$

Onde  $N_{He}$  é chamado de numero de Hedstrom e pode ser calculado pela seguinte fórmula:

$$N_{He} = \frac{37100 \rho \tau_y d^2}{\mu_p^2} \quad (2.19)$$

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

Para fluxo laminar:

$$\frac{dp_f}{dL} = \frac{\mu_p \bar{v}}{1500 d^2} + \frac{\tau_y}{225 d} \quad (2.20)$$

Para o fluxo turbulento podemos usar a Equação (2.16).

### 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:

$$N_{re} = \frac{89100 \rho \bar{v}^{2-n}}{K} \left( \frac{0.0416 d}{3 + \frac{1}{n}} \right)^n \quad (2.21)$$

A velocidade média pode ser obtida pela Equação (2.14). 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:

$$\frac{dp_f}{dL} = \frac{K \bar{v}^n \left( \frac{3+\frac{1}{n}}{0.0416} \right)^n}{144000 d^{1+n}} \quad (2.22)$$

Para o fluxo turbulento podemos usar a Equação (2.16).

### 2.2.7 Perda de pressão friccional em um anular

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

## Modelo de Fluido Newtoniano

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

$$N_{re} = \frac{757\rho\bar{v}(d_2 - d_1)}{\mu} \quad (2.23)$$

A velocidade média pode ser obtida pela equação:

$$\bar{v} = \frac{q}{2.448(d_2^2 - d_1^2)} \quad (2.24)$$

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:

$$\frac{dp_f}{dL} = \frac{\mu\bar{v}}{1000(d_2 - d_1)^2} \quad (2.25)$$

Para o fluxo turbulento:

$$\frac{dp_f}{dL} = \frac{f\rho\bar{v}^2}{21.1(d_2 - d_1)} \quad (2.26)$$

## Fluidos Plásticos de Bingham

Para um fluido que se comporta como plástico de Bingham a velocidade média podem ser obtida pela Equações (2.24). O número de Reynolds pode ser obtido a partir da seguinte equação:

$$N_{re} = \frac{757\rho\bar{v}(d_2 - d_1)}{\mu_p} \quad (2.27)$$

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

$$N_{He} = \frac{24700\rho\tau_y(d_2 - d_1)^2}{\mu_p^2} \quad (2.28)$$

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

Para fluxo laminar:

$$\frac{dp_f}{dL} = \frac{\mu\bar{v}}{1000(d_2 - d_1)^2} + \frac{\tau_y}{200(d_2 - d_1)} \quad (2.29)$$

Para o fluxo turbulento podemos usar a Equação (2.26).

## 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:

$$N_{re} = \frac{109000\rho\bar{v}^{2-n}}{K} \left( \frac{0.0208(d_2 - d_1)}{2 + \frac{1}{n}} \right)^n \quad (2.30)$$

A velocidade média pode ser obtida pela Equação (2.24). 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:

$$\frac{dp_f}{dL} = \frac{K\bar{v}^n \left( \frac{2+\frac{1}{n}}{0.0208} \right)^n}{144000(d_2 - d_1)^{1+n}} \quad (2.31)$$

Para o fluxo turbulento podemos usar a Equação (2.26).

## 2.3 Identificação de pacotes – assuntos

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

## 2.4 Diagrama de pacotes – assuntos

O diagrama de pacotes é apresentado na Figura 2.4.

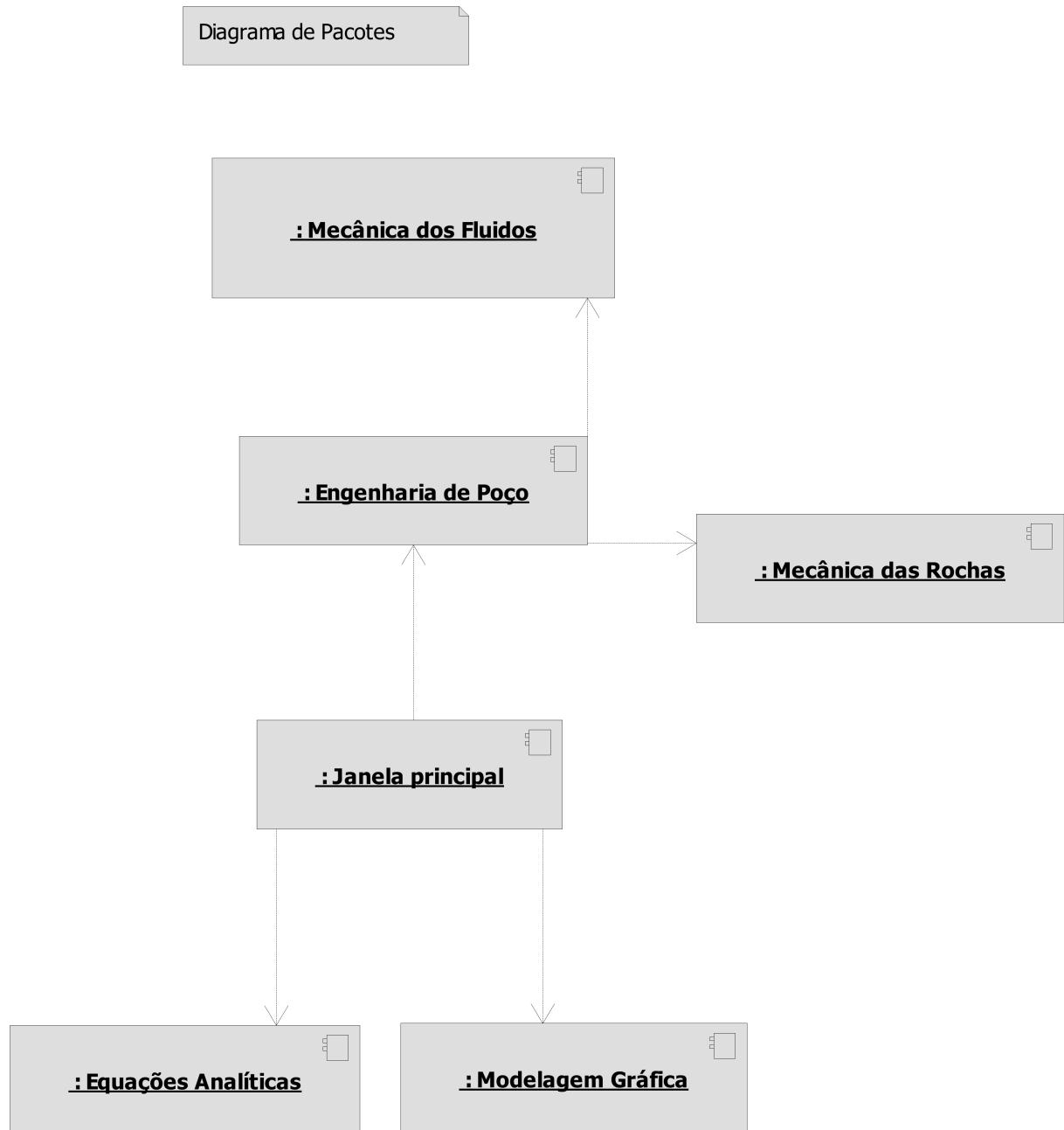


Figura 2.4: Diagrama de Pacotes

2 - Análise Orientada a Objeto

# Capítulo 3

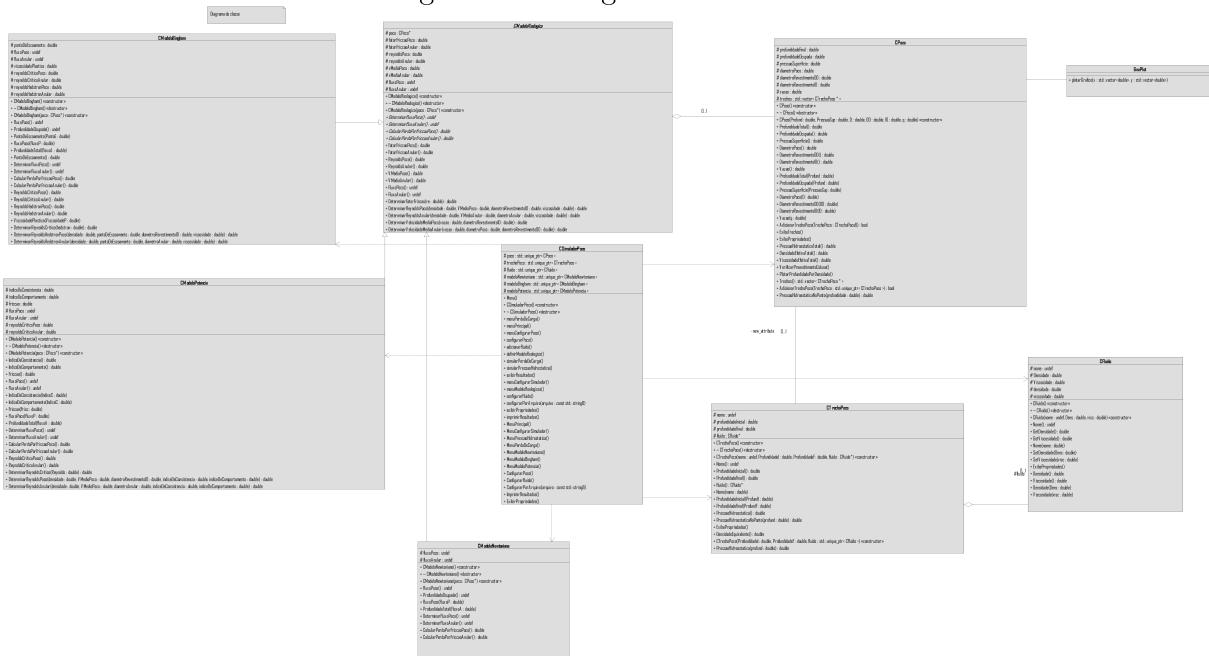
## AOO – Análise Orientada a Objeto

Neste capítulo são apresentados as classes desenvolvidas no projeto, suas relações, atributos e métodos. Ainda teremos um breve conceito de cada classe. Todos os diagramas respeitam a estrutura UML (Linguagem de Modelagem Unificada) para auxiliar na padronização e compreensão. Ainda veremos, além do diagrama de classes, os diagramas de sequência, de comunicação, de máquina de estado e de atividades.

### 3.1 Diagramas de classes

O diagrama de classes é apresentado na Figura 3.1. Ele tem como objetivo apresentar todas as classes, seus atributos, métodos, heranças e relações entre as classes.

Figura 3.1: Diagrama de classes



Fonte: Produzido pelo autor.

### 3.1.1 Dicionário de classes

O software é composto por um total de 9 classes:

- **CSimuladorPoco:** Classe responsável por integrar todas as funcionalidades do simulador.
- **CPoco:** Classe responsável por fornecer os valores e propriedades do poço.
- **CTrechoPoco:** Classe herdeira responsável por subdividir as diferentes partes do poço, permitindo uma análise detalhada das seções.
- **CFluido:** Classe responsável por prover os valores e propriedades do fluido.
- **CModeloReológico:** Classe responsável pelos modelos utilizados para calcular a perda de pressão friccional.
- **CModeloNewtoniano:** Classe responsável por calcular perda de pressão friccional para o modelo newtoniano.
- **CModeloBingham:** Classe responsável por calcular perda de pressão friccional para o modelo Plástico de Bingham.
- **CModeloPotencia:** Classe responsável por calcular perda de pressão friccional para o modelo Lei de Potência.
- **QCustomPlot:** Classe responsável por gerar gráficos utilizando a biblioteca QCus-  
tomPlot (obtida em <https://www.qcustomplot.com/>), para visualização dos dados.

O diagrama de classes é apresentado na Figura 3.1. Nele podemos observar todas as classes, seus atributos, métodos, heranças e seus relacionamentos.

## 3.2 Diagrama de seqüência – eventos e mensagens

O diagrama de sequência enfatiza a troca de eventos e mensagens e sua ordem temporal. Contém informações sobre o fluxo de controle do software. Costuma ser montado a partir de um diagrama de caso de uso e estabelece o relacionamento dos atores (usuários e sistemas externos) com alguns objetos do sistema.

### 3.2.1 Diagrama de sequência geral

A seguir, é apresentado o diagrama de sequência geral na Figura 3.2.

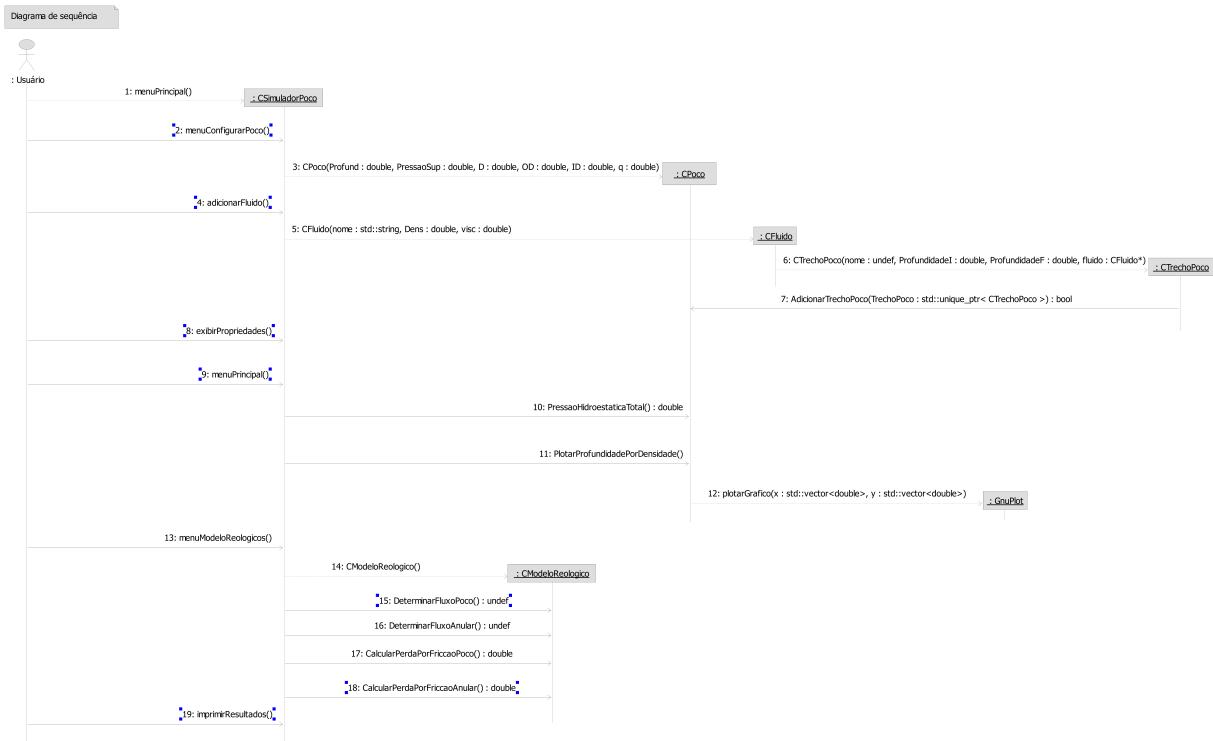


Figura 3.2: Diagrama de sequênciа

### 3.3 Diagrama de comunicação – colaboração

O diagrama de comunicação tem como objetivo apresentar as interações dos objetos, juntamente com sua sequência de processos.

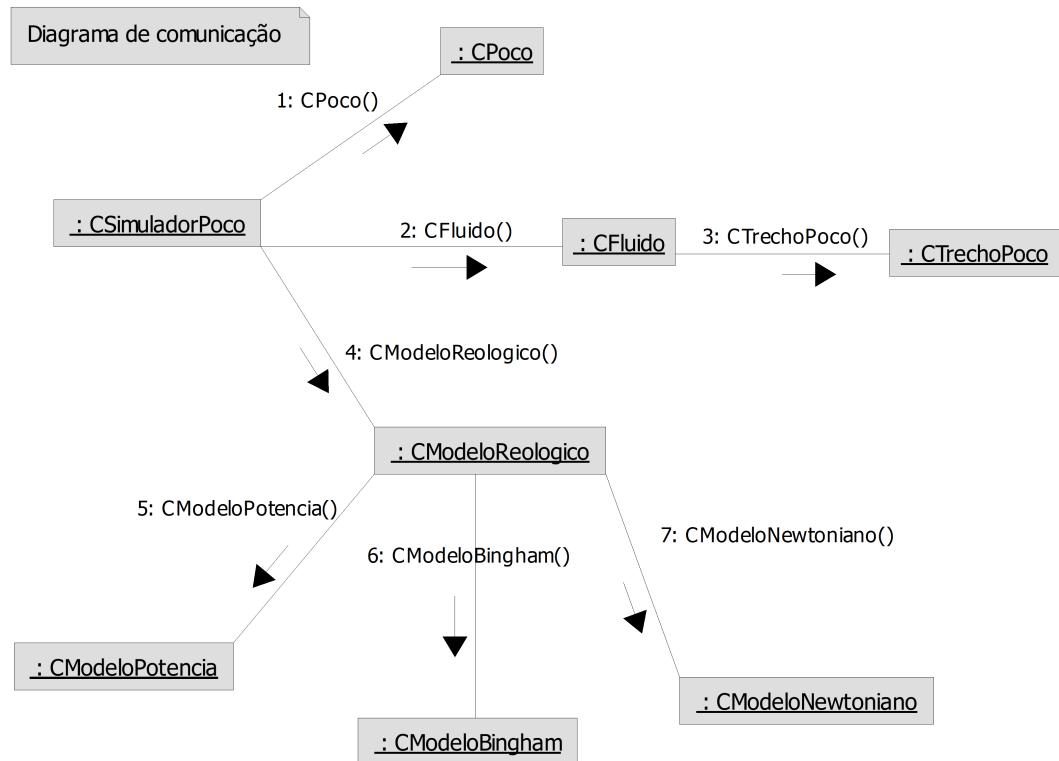


Figura 3.3: Diagrama de comunicação

### 3.4 Diagrama de máquina de estado

Um diagrama de máquina de estado representa os diversos estados que o objeto assume e os eventos que ocorrem ao longo de sua vida ou mesmo ao longo de um processo (histórico do objeto). A Figura 3.4 mostra um diagrama de máquina de estado para o objeto modelo newtoniano.

Inicialmente, os dados são recebidos pela classe responsável pela simulação do poço. Em seguida, seus atributos são criados, e o processo de definição do poço é iniciado. Conforme a configuração do número de seções, a simulação pode seguir para uma seção ou diversas seções.

Após essa definição, o fluido do poço é configurado, podendo ser gás ou óleo. A partir daí, é realizada a simulação do poço, onde os cálculos são processados para determinar os parâmetros necessários. Por fim, os resultados da simulação são plotados e apresentados para análise. Caso o processo seja concluído com sucesso, ele finaliza suas ações.

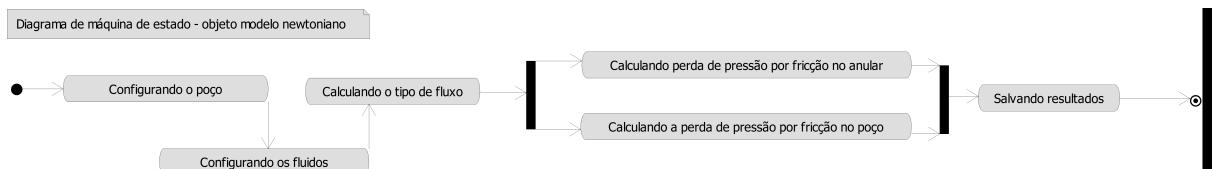


Figura 3.4: Diagrama de máquina de estado

### 3.5 Diagrama de atividades

No diagrama de atividades apresentado, é mostrado em detalhes uma atividade específica. Para o presente caso, será apresentado o diagrama do método CalcularPerdaPorFriccaoPoco da classe CModeloNewtoniano.

Inicialmente, os dados são recebidos pela classe responsável pela simulação de fluidos. Seus atributos são atualizados conforme os dados de entrada. O primeiro passo realizado pelo método é calcular a velocidade média do poço, após esse cálculo o método realiza uma verificação para saber se o número de Reynolds foi calculado, caso não tenha sido o programa apresentara uma mensagem de erro, caso tenha sido o método segue para a determinação do fluxo. O fluxo pode ser laminar ou turbulento existindo uma ação específica para cada um dos caminhos.

Os cálculos são realizados com base nas características específicas de cada fluido, sendo necessárias as respectivas propriedades físicas para completar o processo. Após a conclusão dos cálculos, o sistema finaliza suas operações, retornando os resultados da simulação.

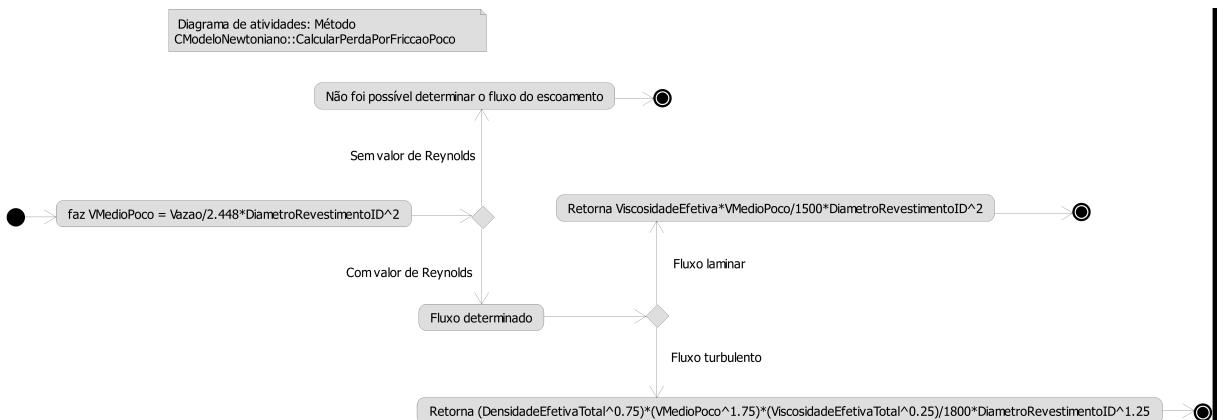


Figura 3.5: Diagrama de atividades

#### Nota:

Não perca de vista a visão do todo; do projeto de engenharia como um todo. Cada capítulo, cada seção, cada parágrafo deve se encaixar. Este é um diferencial fundamental do engenheiro em relação ao técnico, a capacidade de desenvolver projetos, de ver o todo e suas diferentes partes, de modelar processos/sistemas/produtos de engenharia.

# Capítulo 4

## Projeto

Neste capítulo são apresentadas questões relacionadas ao desenvolvimento do projeto, como ambiente de desenvolvimento e bibliotecas gráficas, comentados juntamente com a evolução de versões. Também são apresentados os diagramas de componentes e de implantação.

### 4.1 Projeto do sistema

O paradigma de programação selecionado foi o orientado o objeto.

A linguagem de programação selecionada foi C++ pelos seguintes motivos:

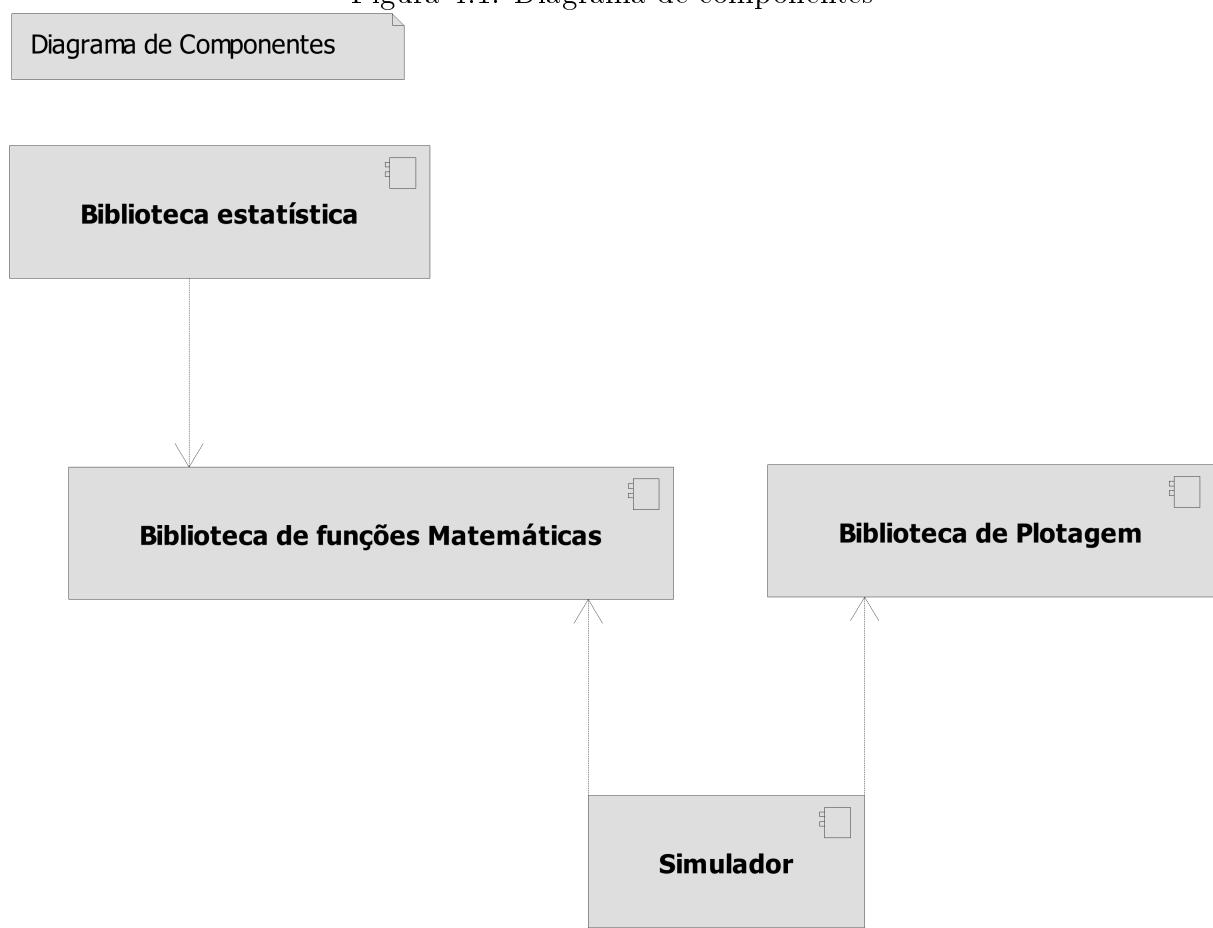
- Linguagem de programação de alto desempenho, adequada para cálculos numéricos intensivos.
- Amplo suporte a orientação a objeto e diversos vínculos com UML.
- Dezenas de bibliotecas de suporte prontas, gráficos (biblioteca Gnuplot) e bibliotecas para gerar saídas em pdf.
- Programação de alto nível com vários graus de abstração.
- Diversos ambientes de desenvolvimento - IDEs, compiladores, debugers, profilers.
- Compiladores e ferramentas gratuitas disponíveis, facilitando o uso por alunos.

Iremos apresentar a seguir os diagramas de componentes e de implementação.

### 4.2 Diagrama de componentes

O diagrama de componentes mostra as relações entre todos os componentes do *software* e é apresentado na Figura 4.1

Figura 4.1: Diagrama de componentes



Fonte: Produzido pelo autor.

Na Figura 4.1, 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 5

## Ciclos de Planejamento/Detalhamento

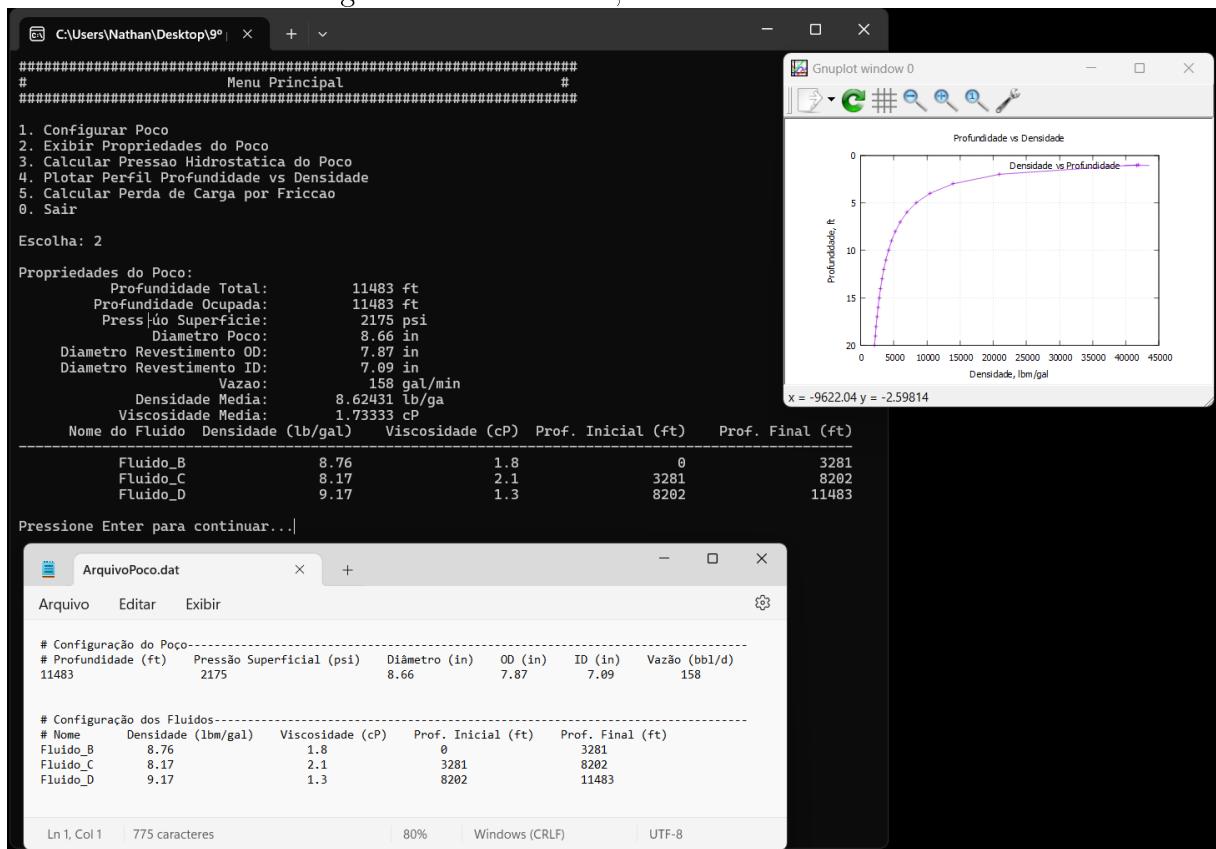
Apresenta-se neste capítulo as versões do software desenvolvido.

### 5.1 Versão 0.1 - Uso modo terminal e Gnuplot para saída de gráfico

Na primeira versão, foi utilizado um sistema de entrada e saída de dados inteiramente pelo terminal, sem o uso de bibliotecas de interface gráfica. O Gnuplot foi empregado para a geração de gráficos e visualização de dados, permitindo uma forma simples e direta de exibir os resultados ao usuário. Essa versão foi desenvolvida em um ambiente de desenvolvimento baseado em terminal, tudo isso no sistema operacional Windows 11.

Note que é um software simples, com uma interação baseada em texto. O usuário pode gerar gráficos e simular diferentes senários.5.1.

Figura 5.1: Versão 0.1, interface do software



Fonte: Produzido pelo autor.

# Capítulo 6

## Ciclos Construção - Implementação

Neste capítulo, são apresentados os códigos fonte implementados.

### 6.1 Versão 0.1 - Código fonte -

Como visto na seção anterior, a versão 0.1 foi a última desenvolvida utilizando execução no terminal. Abaixo serão exibidos 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 6.1 o arquivo com código da função `main`.

---

Listing 6.1: Arquivo de implementação da função main

---

```
1 #include "CAuxiliar.h"
2 #include "CSimuladorPoco.h"
3
4 int main() {
5
6     CAuxiliar cabecalho;
7     CSimuladorPoco simulador;
8
9     cabecalho.cabecalho();
10    simulador.MenuPrincipal();
11
12
13
14
15 }
```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.2 o arquivo de cabeçalho da classe `CSimuladorPoco`.

Listing 6.2: Arquivo de implementação da classe CSimuladorPoco

```
1 #ifndef CSIMULADORPOCO_H
2 #define CSIMULADORPOCO_H
3
4 #include "CPoco.h"
5 #include "CTrechoPoco.h"
6 #include "CModeloNewtoniano.h"
7 #include "CModeloBingham.h"
8 #include "CModeloPotencia.h"
9 #include <memory>
10 #include <vector>
11
12 class CSimuladorPoco {
13 protected:
14     std::unique_ptr<CPoco> poco;
15     std::unique_ptr<CTrechoPoco> trechoPoco;
16     std::unique_ptr<CFluido> fluido;
17     std::unique_ptr<CModeloNewtoniano> modeloNewtoniano;
18     std::unique_ptr<CModeloBingham> modeloBingham;
19     std::unique_ptr<CModeloPotencia> modeloPotencia;
20
21
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();
32     void MenuModeloNewtoniano();
33     void MenuModeloBingham();
34     void MenuModeloPotencia();
35
36
37     // Métodos auxiliares para configurar o poço e fluidos
38     void ConfigurarPoco();
39     void ConfigurarFluido();
40     void ConfigurarPorArquivo(const std::string& arquivo);
41     void ImprimirResultados();
```

```

42
43     // MÃ©todos de simulaÃ§Ã£o
44     void ExibirPropriedades();
45
46 };
47
48 #endif // CSIMULADORPOCO_H

```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.3 a implementação da classe CSimuladorPoco.

Listing 6.3: Arquivo de implementação da classe CSimuladorPoco

```

1 #ifndef CSIMULADORPOCO_H
2 #define CSIMULADORPOCO_H
3
4 #include "CPoco.h"
5 #include "CTrechoPoco.h"
6 #include "CModeloNewtoniano.h"
7 #include "CModeloBingham.h"
8 #include "CModeloPotencia.h"
9 #include <memory>
10 #include <vector>
11
12 class CSimuladorPoco {
13 protected:
14     std::unique_ptr<CPoco> poco;
15     std::unique_ptr<CTrechoPoco> trechoPoco;
16     std::unique_ptr<CFluido> fluido;
17     std::unique_ptr<CModeloNewtoniano> modeloNewtoniano;
18     std::unique_ptr<CModeloBingham> modeloBingham;
19     std::unique_ptr<CModeloPotencia> modeloPotencia;
20
21
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();

```

```

32     void MenuModeloNewtoniano();
33     void MenuModeloBingham();
34     void MenuModeloPotencia();
35
36
37     // Métodos auxiliares para configurar o poço e fluidos
38     void ConfigurarPoco();
39     void ConfigurarFluido();
40     void ConfigurarPorArquivo(const std::string& arquivo);
41     void ImprimirResultados();
42
43     // Métodos de simulação
44     void ExibirPropriedades();
45
46 };
47
48 #endif // CSIMULADORPOCO_H

```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.4 o arquivo de cabeçalho da classe CPoco.

Listing 6.4: Arquivo de implementação da classe CSimuladorPoco

```

1 #ifndef CPOCO_H
2 #define CPOCO_H
3
4 #include <vector>
5 #include <memory>
6 #include "CFluido.h"
7 #include "CTrechoPoco.h"
8
9 class CPoco {
10 protected:
11     double profundidadeFinal = 0.0;
12     double profundidadeOcupada = 0;
13     double pressaoSuperficie = 0.0;
14     double diametroPoco = 0.0;
15     double diametroRevestimentoOD = 0.0;
16     double diametroRevestimentoID = 0.0;
17     double vazao = 0.0;
18     std::vector<std::unique_ptr<CTrechoPoco>> trechos;
19
20 public:
21     //construtor

```

```

22     CPoco() {}
23     ~CPoco() {}
24     CPoco(double Profund, double PressaoSup, double D, double OD,
25            double ID, double q)
26     : profundidadeFinal(Profund), pressaoSuperficie(PressaoSup),
27       diametroPoco(D), diametroRevestimentoOD(OD),
28       diametroRevestimentoID(ID), vazao(q) {}

29
30     // Getters
31     double ProfundidadeTotal() const { return profundidadeFinal; }
32     double ProfundidadeOcupada() const { return profundidadeOcupada;
33       ; }
34     double PressaoSuperficie() const { return pressaoSuperficie; }
35     double DiametroPoco() const { return diametroPoco; }
36     double DiametroRevestimentoOD() const { return
37       diametroRevestimentoOD; }
38     double DiametroRevestimentoID() const { return
39       diametroRevestimentoID; }
40     double Vazao() const { return vazao; }
41     std::vector<CTrechoPoco*> Trechos() const;

42
43     // Setters
44     void ProfundidadeTotal( double Profund ) { profundidadeFinal =
45       Profund; }
46     void ProfundidadeOcupada( double Profund ) {
47       profundidadeOcupada = Profund; }
48     void PressaoSuperficie( double PressaoSup ) { pressaoSuperficie
49       = PressaoSup; }
50     void DiametroPoco( double D ) { diametroPoco = D; }
51     void DiametroRevestimentoOD( double OD ) {
52       diametroRevestimentoOD = OD; }
53     void DiametroRevestimentoID( double ID ) {
54       diametroRevestimentoID = ID; }
55     void Vazao( double q ) { vazao = q; }

56
57     // Metodos
58     bool AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco>
59       TrechoPoco);
60     double PressaoHidroestaticaTotal() const;
61     double PressaoHidroestaticaNoPonto(double profundidade) const;
62     double DensidadeEfetivaTotal() const;
63     double ViscosidadeEfetivaTotal() const;

```

```

52     void VerificarPreenchimentoColuna();
53     void PlotarProfundidadePorDensidade();
54
55 };
56
57 #endif

```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.5 a implementação da classe CPoco.

Listing 6.5: Arquivo de implementação da classe CSimuladorPoco

---

```

1 #include "CPoco.h"
2 #include <iostream>
3 #include <vector>
4 #include <fstream>
5 #include <cstdlib>
6
7 // MÃ©todos
8
9 std::vector<CTrechoPoco*> CPoco::Trechos() const {
10    std::vector<CTrechoPoco*> trechosPonteiros;
11    for (const auto& trecho : trechos) {
12        trechosPonteiros.push_back(trecho.get()); // Adiciona o
13        ponteiro do trecho ao vetor
14    }
15    return trechosPonteiros; // Retorna o vetor de ponteiros
16 }
17 bool CPoco::AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco>
18                                 TrechoPoco) {
19    double ProfundidadeFluido = TrechoPoco->ProfundidadeFinal() -
20                                TrechoPoco->ProfundidadeInicial();
21
22    // Verifica se a profundidade total ocupada + profundidade do
23    // novo fluido excede a profundidade total do poÃ§o
24    if (profundidadeOcupada + ProfundidadeFluido <=
25        profundidadeFinal) {
26        trechos.push_back(std::move(TrechoPoco));
27        profundidadeOcupada += ProfundidadeFluido;
28        return true;
29    } else {
30        std::cout << "Erro: o fluido excede a profundidade total do
31        poÃ§o!\n";
32    }
33}

```

```
27         return false;
28     }
29 }
30
31 double CPoco::PressaoHidroestaticaTotal() const {
32     double PressaoTotal = 0.0;
33     for (const auto& Trecho : trechos) {
34         PressaoTotal += Trecho->PressaoHidroestatica();
35     }
36     return PressaoTotal + pressaoSuperficie;
37 }
38
39 double CPoco::PressaoHidroestaticaNoPonto(double profundidade)
40     const {
41     double PressaoTotal = pressaoSuperficie;
42     double profundidadeAcumulada = 0.0;
43
44     for (const auto& Trecho : trechos) {
45         double profundidadeTrecho = Trecho->ProfundidadeFinal() -
46             Trecho->ProfundidadeInicial();
47
48         // Verifica se a profundidade estÃ¡ dentro do trecho atual
49         if (profundidade <= profundidadeAcumulada +
50             profundidadeTrecho) {
51             // Calcula a contribuiÃ§Ã£o do trecho atÃ© a
52             // profundidade desejada
53             PressaoTotal += Trecho->PressaoHidroestatica(
54                 profundidade - profundidadeAcumulada);
55             break;
56         } else {
57             // Adiciona a pressÃ£o hidrostÃ;stica do trecho completo
58             PressaoTotal += Trecho->PressaoHidroestatica();
59             profundidadeAcumulada += profundidadeTrecho;
60         }
61     }
62
63     return PressaoTotal;
64 }
65
66 void CPoco::VerificarPreenchimentoColuna() {
67     double ProfundidadeNaoOcupada = profundidadeFinal -
68         profundidadeOcupada;
```

```
63
64     if (ProfundidadeNaoOcupada > 0) {
65         std::cout << "Uma coluna de fluido precisa ser adicionada!\n";
66         std::cout << std::endl;
67     } else {
68         std::cout << "A coluna de fluidos equivale à profundidade da coluna do poço!\n";
69         std::cout << std::endl;
70     }
71 }
72
73 double CPoco::DensidadeEfetivaTotal() const {
74     double DensidadeTotal = 0.0;
75     double ComprimentoTotal = 0.0;
76
77     for (const auto& Trecho : trechos) {
78         double ComprimentoTrecho = Trecho->ProfundidadeFinal() -
79             Trecho->ProfundidadeInicial();
80         DensidadeTotal += Trecho->DensidadeEquivalente() *
81             ComprimentoTrecho;
82         ComprimentoTotal += ComprimentoTrecho;
83     }
84
85     return DensidadeTotal / ComprimentoTotal;
86 }
87
88 double CPoco::ViscosidadeEfetivaTotal() const {
89     double ViscosidadeTotal = 0.0;
90     for (const auto& Trecho : trechos) {
91         ViscosidadeTotal += Trecho->Fluido()->Viscosidade();
92     }
93     return ViscosidadeTotal / trechos.size();
94 }
95
96 void CPoco::PlotarProfundidadePorDensidade() {
97     std::vector<double> Profundidade;
98     std::vector<double> Densidade;
99
100    double ProfunTotal = 0;
101
102    // Coletar dados para a profundidade e densidade
```

```

101     for (const auto& trecho : trechos) {
102         double Intervalo = trecho->ProfundidadeFinal() - trecho->
103             ProfundidadeInicial();
104
105         for (double i = 0; i <= Intervalo; i += 1) { // Usando um
106             incremento menor
107             double ProfundidadeAtual = ProfunTotal + i; // Atualiza
108                 a profundidade em cada iteração
109             double Dens = PressaoHidroestaticaNoPonto(
110                 ProfundidadeAtual) / (ProfundidadeAtual * 0.05195);
111
112             Densidade.push_back(Dens);
113             Profundidade.push_back(ProfundidadeAtual); // Armazena
114                 a profundidade atual
115         }
116         ProfunTotal += Intervalo; // Avança a profundidade total
117     }
118
119 // Escrever dados em arquivo
120 std::ofstream outputFile("dados.txt");
121
122 // Comando Gnuplot para plotar os dados
123 std::ofstream gnuplotFile("plot_script.gp");
124 if (!gnuplotFile.is_open()) {
125     std::cerr << "Erro ao abrir o arquivo plot_script.gp para "
126         "escrita." << std::endl;
127     return;
128 }
129 gnuplotFile << "set title 'Profundidade vs Densidade'\n";
130 gnuplotFile << "set xlabel 'Densidade, lbm/gal'\n"; //
131             Corrigido o label
132 gnuplotFile << "set ylabel 'Profundidade, ft'\n"; // Corrigido
133             o label
134 gnuplotFile << "set xrange [20:0]\n"; // Inverter o eixo Y
135 gnuplotFile << "set grid\n"; // Adicionar grade ao gráfico

```

```

134     gnuplotFile << "set style data linespoints\n"; // Estilo de
           linha com pontos
135
136     // Plota apenas uma curva
137     gnuplotFile << "plot 'dados.txt' using 2:1 with linespoints"
           title 'Densidade vs Profundidade'\n";
138     gnuplotFile << "set terminal pngcairo size 1920,1080\n";
139     gnuplotFile << "set output 'Profundidade_vs_densidade.png'\n";
140     gnuplotFile << "pause -1\n"; // Pausa para que você possa ver
           o gráfico
141     gnuplotFile.close();
142
143     // Executa o Gnuplot com o script gerado
144     std::system("gnuplot -persist plot_script.gp");
145 }
```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.6 o arquivo de cabeçalho da classe CFluido.

Listing 6.6: Arquivo de implementação da classe CFluido

```

1 #ifndef CFLUIDO_H
2 #define CFLUIDO_H
3
4 #include <string>
5
6 class CFluido {
7 protected:
8     std::string nome;
9     double densidade = 0.0;
10    double viscosidade = 0.0;
11
12 public:
13     // Construtor
14     CFluido() {}
15     ~CFluido() {}
16     CFluido(std::string nome, double Dens, double visc)
17         : nome(nome), densidade(Dens), viscosidade(visc) {}
18
19     // Getters
20     std::string Nome() const { return nome; }
21     double Densidade() const { return densidade; }
22     double Viscosidade() const { return viscosidade; }
23
```

```

24     // Setters
25     void Nome(double nome) { nome = nome; }
26     void Densidade(double Dens) { densidade = Dens; }
27     void Viscosidade(double visc) { viscosidade = visc; }
28
29 };
30
31 #endif

```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.7 a implementação da classe CFluido.

Listing 6.7: Arquivo de implementação da classe CFluido

---

```
1 #include "CFluido.h"
```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.8 o arquivo de cabeçalho da classe CModeloReologico.

Listing 6.8: Arquivo de implementação da classe CModeloReologico

---

```

1 ifndef CMODELOREOLOGICO_H
2 define CMODELOREOLOGICO_H
3
4 include <string>
5 include "CPoco.h"
6
7 class CModeloReologico {
8
9 protected:
10    double fatorFriccaoPoco = 0.0;
11    double fatorFriccaoAnular = 0.0;
12    double reynoldsPoco = 0.0;
13    double reynoldsAnular = 0.0;
14    double vMediaPoco = 0.0;
15    double vMediaAnular = 0.0;
16    std::string fluxoPoco;
17    std::string fluxoAnular;
18    CPoco* poco;
19
20 public:
21     //Construtores
22     CModeloReologico() {}
23     virtual ~CModeloReologico() {}
24     CModeloReologico(CPoco* poco) : poco(poco) {}
25

```

```

26     //Getters
27     double FatorFriccaoPoco() const { return fatorFriccaoPoco; }
28     double FatorFriccaoAnular() const { return fatorFriccaoAnular; }
29     }
30     double ReynoldsPoco() const { return reynoldsPoco; }
31     double ReynoldsAnular() const { return reynoldsAnular; }
32     double VMediaPoco() const { return vMediaPoco; }
33     double VMediaAnular() const { return vMediaAnular; }
34     std::string FluxoPoco() const { return fluxoPoco; }
35     std::string FluxoAnular() const { return fluxoAnular; }
36
37     //MÃ©todos
38     double DeterminarFriccao(double re);
39     double DeterminarReynoldsPoco(double densidade, double
40         VMedioPoco, double diametroRevestimentoID, double
41         viscosidade);
42     double DeterminarReynoldsAnular(double densidade, double
43         VMedioAnular, double diametroAnular, double viscosidade);
44     double DeterminarVelocidadeMediaPoco(double vazao, double
45         diametroRevestimentoID);
46     double DeterminarVelocidadeMediaAnular(double vazao, double
47         diametroPoco, double diametroRevestimentoOD);
48     virtual std::string DeterminarFluxoPoco() = 0;
49     virtual std::string DeterminarFluxoAnular() = 0;
50     virtual double CalcularPerdaPorFriccaoPoco() = 0;
51     virtual double CalcularPerdaPorFriccaoAnular() = 0;
52 };
53
54 #endif

```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.9 a implementação da classe CModeloReologico.

Listing 6.9: Arquivo de implementação da classe CModeloReologico

```

1 #include <iostream>
2 #include <cmath>
3 #include <iomanip>
4
5 #include "CModeloReologico.h"
6
7 double CModeloReologico::DeterminarReynoldsPoco(double densidade,
8         VMedioPoco, double diametroRevestimentoID, double
9         viscosidade) {

```

```
8     reynoldsPoco = (928 * densidade * VMedioPoco *
9         diametroRevestimentoID) / viscosidade;
10    return reynoldsPoco;
11}
12double CModeloReologico::DeterminarReynoldsAnular(double densidade,
13    double VMedioAnular, double diametroAnular, double viscosidade)
14{
15    reynoldsAnular = (757 * densidade * VMedioAnular *
16        diametroAnular) / viscosidade;
17    return reynoldsAnular;
18}
19
20double CModeloReologico::DeterminarVelocidadeMediaPoco(double vazao
21    , double diametroRevestimentoID){
22    vMediaPoco = vazao / (2.448 * std::pow(diametroRevestimentoID,
23        2));
24    return vMediaPoco;
25}
26
27
28double CModeloReologico::DeterminarVelocidadeMediaAnular(double
29    vazao, double diametroPoco, double diametroRevestimentoOD){
30    vMediaAnular = vazao / (2.448 * (std::pow(diametroPoco, 2) -
31        std::pow(diametroRevestimentoOD, 2)));
32    return vMediaAnular;
33}
34
35    // Definir a equação f(x) que precisa ser resolvida
36    auto f = [re](double x) {
37        return 1 / std::sqrt(x) - 4 * std::log10(re * std::sqrt(x))
38            + 0.395;
39    };
40
41    // Definir a derivada de f(x) para o método de Newton-Raphson
```

```

40     auto df = [] (double x) {
41         return -0.5 / (x * std::sqrt(x)) - (2 / (x * std::log(10)));
42     ;
43 };
44 // MÃ©todo de Newton-Raphson para resolver f(x) = 0
45 auto newtonRaphson = [&] (double x0, double tol = 1e-6, int
46     max_iter = 1000000) {
47     double x = x0;
48     for (int i = 0; i < max_iter; i++) {
49         double fx = f(x);
50         double dfx = df(x);
51         if (std::abs(fx) < tol) {
52             return x;
53         }
54         x -= fx / dfx;
55     }
56     return x;
57 };
58 // Estimar o chute inicial usando o fator laminar
59 double x0 = laminar_fator(re);
60
61 // Resolver a equaÃ§Ã£o de Fanning usando Newton-Raphson
62 return newtonRaphson(x0);
63 }
```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.10 o arquivo de cabeçalho da classe CModeloNewtoniano.

Listing 6.10: Arquivo de implementação da classe CModeloNewtoniano

---

```

1 #ifndef CMODELONEWTONIANO_H
2 #define CMODELONEWTONIANO_H
3
4 #include "CModeloReológico.h"
5
6
7 class CModeloNewtoniano : public CModeloReológico {
8
9
10 public:
11     // Construtor
12     CModeloNewtoniano() {}
```

```

13     ~CModeloNewtoniano() {}
14     CModeloNewtoniano(CPoco* poco) : CModeloReológico(poco){}
15
16     // MÃ©todos
17     std::string DeterminarFluxoPoco() override;
18     std::string DeterminarFluxoAnular() override;
19     double CalcularPerdaPorFricçãoPoco() override;
20     double CalcularPerdaPorFricçãoAnular() override;
21
22 };
23
24 #endif

```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.11 a implementação da classe CModeloNewtoniano.

Listing 6.11: Arquivo de implementação da classe CModeloNewtoniano

```

1 #include "CModeloNewtoniano.h"
2 #include <iostream>
3 #include <cmath>
4
5 // Função para determinar o tipo de fluxo no poço
6 std::string CModeloNewtoniano::DeterminarFluxoPoco() {
7
8     DeterminarVelocidadeMediaPoco(poco->Vazão(), poco->
9         DiametroRevestimentoID());
10    DeterminarReynoldsPoco(poco->DensidadeEfetivaTotal(),
11        vMediaPoco, poco->DiametroRevestimentoID(), poco->
12        ViscosidadeEfetivaTotal());
13    fluxoPoco = (reynoldsPoco <= 2100) ? "Laminar" : "Turbulento";
14    // Determinação do fluxo
15
16    return fluxoPoco;
17 }
18
19 // Função para determinar o tipo de fluxo no espaço anular
20 std::string CModeloNewtoniano::DeterminarFluxoAnular() {
21
22     double diametroAnular = poco->DiametroPoco() - poco->
23         DiametroRevestimentoOD();
24
25     DeterminarVelocidadeMediaAnular(poco->Vazão(), poco->
26         DiametroPoco(), poco->DiametroRevestimentoOD());

```

```
21     DeterminarReynoldsAnular(poco->DensidadeEfetivaTotal(),
22         vMediaAnular, diametroAnular, poco->ViscosidadeEfetivaTotal
23         ());
24     this->fluxoAnular = (reynoldsAnular <= 2100) ? "Laminar" : "
25         Turbulento"; // Determinação do fluxo
26
27 // Função para calcular a perda de carga por fricção no poço
28 double CModeloNewtoniano::CalcularPerdaPorFriccaoPoco() {
29     if (fluxoPoco.empty()) {
30         DeterminarFluxoPoco();
31     }
32
33     this->fatorFriccaoPoco = DeterminarFatorFriccao(reynoldsPoco);
34
35     if (fluxoPoco == "Laminar") {
36         return (poco->ViscosidadeEfetivaTotal() * vMediaPoco) /
37             (1500 * std::pow(poco->DiametroRevestimentoID(), 2));
38     } else { // Fluxo turbulento
39         return (fatorFriccaoPoco * poco->DensidadeEfetivaTotal() *
40             std::pow(vMediaPoco, 2)) / (25.8 * poco->
41             DiametroRevestimentoID());
42     }
43
44 // Função para calcular a perda de carga por fricção no espaço
45 // anular
46
47 double CModeloNewtoniano::CalcularPerdaPorFriccaoAnular() {
48     if (fluxoAnular.empty()) {
49         DeterminarFluxoAnular();
50     }
51
52     double diametroAnular = poco->DiametroPoco() - poco->
53         DiametroRevestimentoOD();
54     this->fatorFriccaoAnular = DeterminarFatorFriccao(
55         reynoldsAnular);
56
57     if (fluxoAnular == "Laminar") {
58         return (poco->ViscosidadeEfetivaTotal() * vMediaAnular) /
59             (1000 * std::pow(diametroAnular, 2));
60 }
```

```

53     } else { // Fluxo turbulento
54         return (fatorFriccaoAnular * poco->DensidadeEfetivaTotal()
55             * std::pow(vMediaAnular, 2) ) / (21.1 * diametroAnular);
56     }

```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.12 o arquivo de cabeçalho da classe CModeloBingham.

Listing 6.12: Arquivo de implementação da classe CModeloBingham

```

1 #ifndef CMODELOBINGHAM_H
2 #define CMODELOBINGHAM_H
3
4 #include "CModeloReologico.h"
5
6
7 class CModeloBingham : public CModeloReologico {
8
9 protected:
10    double viscosidadePlastica = 0.0;
11    double pontoDeEscoamento = 0.0;
12    double reynoldsCriticoPoco = 0.0;
13    double reynoldsCriticoAnular = 0.0;
14    double reynoldsHedstronPoco = 0.0;
15    double reynoldsHedstronAnular = 0.0;
16
17 public:
18     //Construtor
19     CModeloBingham() {}
20     ~CModeloBingham() {}
21     CModeloBingham(CPoco* poco) : CModeloReologico(poco){}
22
23     // Getters
24     double PontoDeEscoamento() const { return pontoDeEscoamento; }
25     double ReynoldsCriticoPoco() const { return reynoldsCriticoPoco
26         ; }
27     double ReynoldsCriticoAnular() const { return
28         reynoldsCriticoAnular; }
29     double ReynoldsHedstronPoco() const { return
30         reynoldsHedstronPoco; }
31     double ReynoldsHedstronAnular() const { return
32         reynoldsHedstronAnular; }

```

```

30     // Setters
31     void PontoDeEscoamento( double PontoE ) { pontoDeEscoamento =
32         PontoE; }
33
34     // Métodos
35
36     double DeterminarReynoldsCritico(double hedstron);
37     double DeterminarReynoldsHedstronPoco(double densidade, double
38         pontoDeEscoamento, double diametroRevestimentoID, double
39         viscosidade);
40     double DeterminarReynoldsHedstronAnular(double densidade,
41         double pontoDeEscoamento, double diametroAnular, double
42         viscosidade);
43
44     std::string DeterminarFluxoPoco() override;
45     std::string DeterminarFluxoAnular() override;
46     double CalcularPerdaPorFriccaoPoco() override;
47     double CalcularPerdaPorFriccaoAnular() override;
48
49 };
50
51 #endif

```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.13 a implementação da classe CModeloBingham.

Listing 6.13: Arquivo de implementação da classe CModeloBingham

```

1 #include "CModeloBingham.h"
2 #include <iostream>
3 #include <cmath>
4
5 #include <cmath>
6 #include <iostream>
7
8
9 double CModeloBingham::DeterminarReynoldsCritico(double hedstron) {
10     // Define a função f(x) que representa a equação para
11     // Reynolds crítico
12     auto func = [hedstron](double x) {
13         return ((x / std::pow(1 - x, 3)) * 16800) - hedstron;
14     };

```

```

15     // Define a derivada de f(x)
16     auto derivadaFunc = [] (double x) {
17         return (16800 * (1 + x) / std::pow(1 - x, 4));
18     };
19
20     // Método de Newton-Raphson para encontrar a raiz
21     auto newtonRaphson = [&] (double x) {
22         for (int i = 0; i < 10000; ++i) {
23             x = x - func(x) / derivadaFunc(x);
24             if (fabs(func(x)) < 1e-2) break; // Verifica a
25                 convergência
26         }
27         return x;
28     };
29
30     // Aplica Newton-Raphson a partir de um chute inicial
31     double y = newtonRaphson(0.99);
32
33     // Calcula e retorna o Reynolds crítico usando o valor
34     // encontrado
35     return ((1 - ((4.0 / 3.0) * y) + ((1.0 / 3.0) * std::pow(y, 4)))
36     ) * hedstron) / (8 * y);
37 }
38
39 double CModeloBingham::DeterminarReynoldsHedstronPoco(double
40     densidade, double pontoDeEscoamento, double
41     diametroRevestimentoID, double viscosidade) {
42     reynoldsCriticoPoco = DeterminarReynoldsCritico(
43         reynoldsHedstronPoco);
44     return (37100 * densidade * pontoDeEscoamento * std::pow(
45         diametroRevestimentoID, 2))/ (std::pow(viscosidade, 2));
46 }
47
48 double CModeloBingham::DeterminarReynoldsHedstronAnular(double
49     densidade, double pontoDeEscoamento, double diametroAnular,
50     double viscosidade) {
51     reynoldsCriticoAnular = DeterminarReynoldsCritico(
52         reynoldsHedstronAnular);
53     return (24700 * densidade * pontoDeEscoamento * std::pow(
54         diametroAnular, 2))/ (std::pow(viscosidade, 2));
55 }
56
57 // Função para determinar o tipo de fluxo no poço

```

```

46 std::string CModeloBingham::DeterminarFluxoPoco() {
47
48     vMediaPoco = DeterminarVelocidadeMediaPoco(poco->Vazao(), poco
49         ->DiametroRevestimentoID());
50
51     reynoldsPoco = DeterminarReynoldsPoco(poco->
52         DensidadeEfetivaTotal(), vMediaPoco, poco->
53         DiametroRevestimentoID(), viscosidadePlastica);
54
55     reynoldsHedstronPoco = DeterminarReynoldsHedstronPoco(poco->
56         DensidadeEfetivaTotal(), pontoDeEscoamento, poco->
57         DiametroRevestimentoID(), viscosidadePlastica);
58
59     fluxoPoco = (reynoldsPoco <= reynoldsCriticoPoco) ? "Laminar" :
60         "Turbulento"; // Determinação do fluxo
61
62     return fluxoPoco;
63 }
64
65 // Função para determinar o tipo de fluxo no espaço anular
66 std::string CModeloBingham::DeterminarFluxoAnular() {
67
68     double diametroAnular = poco->DiametroPoco() - poco->
69         DiametroRevestimentoOD();
70
71     vMediaAnular = DeterminarVelocidadeMediaAnular(poco->Vazao(),
72         poco->DiametroPoco(), poco->DiametroRevestimentoOD());
73     reynoldsAnular = DeterminarReynoldsAnular(poco->
74         DensidadeEfetivaTotal(), vMediaAnular, diametroAnular,
75         viscosidadePlastica);
76     reynoldsHedstronAnular = DeterminarReynoldsHedstronAnular(poco
77         ->DensidadeEfetivaTotal(), pontoDeEscoamento, diametroAnular
78         , viscosidadePlastica);
79
80     fluxoAnular = (reynoldsAnular <= reynoldsCriticoAnular) ? "
81         Laminar" : "Turbulento"; // Determinação do fluxo
82
83     return fluxoAnular;
84 }
85
86 // Função para calcular a perda de carga por fricção no poço
87 double CModeloBingham::CalcularPerdaPorFriccaoPoco() {
88     if (fluxoPoco.empty()) {
89         DeterminarFluxoPoco();
90     }

```

```

75
76     fatorFriccaoPoco = DeterminarFatorFriccao(reynoldsPoco);
77     std::cout << viscosidadePlastica << std::endl;
78     std::cout << vMediaPoco << std::endl;
79     std::cout << poco->DiametroRevestimentoID() << std::endl;
80     std::cout << pontoDeEscoamento << std::endl;
81
82     if (fluxoPoco == "Laminar") {
83         return ((viscosidadePlastica * vMediaPoco) / (1500 * (std
84             ::pow(poco->DiametroRevestimentoID(), 2))) + (
85                 pontoDeEscoamento/(225*poco->DiametroRevestimentoID())));
86     } else { // Fluxo turbulento
87         return (fatorFriccaoPoco * poco->DensidadeEfetivaTotal() *
88             std::pow(vMediaPoco, 2) ) / (25.8 * poco->
89             DiametroRevestimentoID());
90     }
91 }
92
93 // Função para calcular a perda de carga por fricção no espaço
94 // anular
95 double CModeloBingham::CalcularPerdaPorFriccaoAnular() {
96     if (fluxoAnular.empty()) {
97         DeterminarFluxoAnular();
98     }
99
100    double diametroAnular = poco->DiametroPoco() - poco->
101        DiametroRevestimentoOD();
102    fatorFriccaoAnular = DeterminarFatorFriccao(reynoldsAnular);
103
104    if (fluxoAnular == "Laminar") {
105        return ((viscosidadePlastica * vMediaAnular) / (1000 * std
106            ::pow(diametroAnular, 2))) + (pontoDeEscoamento/(200*
107                diametroAnular));
108    } else { // Fluxo turbulento
109        return (fatorFriccaoAnular * poco->DensidadeEfetivaTotal()
110            * std::pow(vMediaAnular, 2) ) / (21.1 * diametroAnular);
111    }
112 }

```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.14 o arquivo de cabeçalho da classe CModeloPotencia.

---

Listing 6.14: Arquivo de implementação da classe CModeloPotencia

```
1 #ifndef CMODELOPOTENCIA_H
2 #define CMODELOPOTENCIA_H
3
4 #include "CModeloReologico.h"
5
6
7 class CModeloPotencia : public CModeloReologico {
8
9 protected:
10    double indiceDeConsistencia = 0.0;
11    double indiceDeComportamento = 1.0;
12    double reynoldsCriticoPoco = 2100.0;
13    double reynoldsCriticoAnular = 2100.0;
14
15 public:
16    //Construtor
17    CModeloPotencia() {}
18    ~CModeloPotencia() {}
19    CModeloPotencia(CPoco* poco) : CModeloReologico(poco){}
20
21    // Getters
22    double ReynoldsCriticoPoco() const { return reynoldsCriticoPoco
23        ; }
24    double ReynoldsCriticoAnular() const { return
25        reynoldsCriticoAnular; }
26    double IndiceDeConsistencia() const { return
27        indiceDeConsistencia; }
28    double IndiceDeComportamento() const { return
29        indiceDeComportamento; }
30    std::string FluxoPoco() const { return fluxoPoco; }
31    std::string FluxoAnular() const { return fluxoAnular; }
32
33    // Setters
34    void IndiceDeConsistencia( double IndiceC ) {
35        indiceDeConsistencia = IndiceC; }
36    void IndiceDeComportamento( double IndiceC ) {
37        indiceDeComportamento = IndiceC; }
38    void FluxoPoco( double FluxoP ) { fluxoPoco = FluxoP; }
39    void ProfundidadeTotal( double FloxoA ) { fluxoAnular = FloxoA;
40        }
41
42
```

```

36     // MÃ©todos
37     double DeterminarReynoldsCritico(double Reynolds);
38     double DeterminarReynoldsPoco(double densidade, double
39         VMedioPoco, double diametroRevestimentoID, double
40         indiceDeConsistencia, double indiceDeComportamento);
41     double DeterminarReynoldsAnular(double densidade, double
42         VMedioPoco, double diametroAnular, double
43         indiceDeConsistencia, double indiceDeComportamento);
44     std::string DeterminarFluxoPoco() override;
45     std::string DeterminarFluxoAnular() override;
46     double CalcularPerdaPorFriccaoPoco() override;
47     double CalcularPerdaPorFriccaoAnular() override;
48
49 };
50
51 #endif

```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.15 a implementação da classe CModeloPotencia.

Listing 6.15: Arquivo de implementação da classe CModeloPotencia

---

```

1 #include "CModeloPotencia.h"
2 #include <iostream>
3 #include <cmath>
4
5 double CModeloPotencia::DeterminarReynoldsCritico(double Reynolds)
6 {
7     return 0.0;
8 }
9
10 double CModeloPotencia::DeterminarReynoldsPoco(double densidade,
11     double VMedioPoco, double diametroRevestimentoID, double
12     indiceDeConsistencia, double indiceDeComportamento) {
13     reynoldsPoco = ((89100 * densidade * std::pow(VMedioPoco, 2 -
14         indiceDeComportamento)) / indiceDeConsistencia) * (std::pow
15         ((0.0416 * diametroRevestimentoID)/(3+(1/
16         indiceDeComportamento)),indiceDeComportamento));
17
18     if (indiceDeComportamento < 0.4){
19         reynoldsCriticoPoco = DeterminarReynoldsCritico(
20             reynoldsPoco);
21     }

```

```
16     return reynoldsPoco;
17 }
18
19 double CModeloPotencia::DeterminarReynoldsAnular(double densidade,
20         double vMedioPoco, double diametroAnular, double
21         indiceDeConsistencia, double indiceDeComportamento) {
22     reynoldsAnular = ((109000 * densidade * std::pow(vMedioPoco, 2-
23             indiceDeComportamento)) / indiceDeConsistencia) * (std::pow
24             ((0.0208 * diametroAnular)/(2+(1/indiceDeComportamento)),
25             indiceDeComportamento));
26
27     if (indiceDeComportamento < 0.4){
28         reynoldsCriticoAnular = DeterminarReynoldsCritico(
29             reynoldsAnular);
30     }
31
32     return reynoldsAnular;
33 }
34
35 // Função para determinar o tipo de fluxo no poço
36 std::string CModeloPotencia::DeterminarFluxoPoco() {
37
38     vMediaPoco = DeterminarVelocidadeMediaPoco(poco->Vazao(), poco
39             ->DiametroRevestimentoID());
40     reynoldsPoco = DeterminarReynoldsPoco(poco->
41             DensidadeEfetivaTotal(), vMediaPoco, poco->
42             DiametroRevestimentoID(), indiceDeConsistencia,
43             indiceDeComportamento);
44
45     fluxoPoco = (reynoldsPoco <= reynoldsCriticoPoco) ? "Laminar" :
46             "Turbulento"; // Determinação do fluxo
47
48     return fluxoPoco;
49 }
50
51
52 // Função para determinar o tipo de fluxo no espaço anular
53 std::string CModeloPotencia::DeterminarFluxoAnular() {
54
55     double diametroAnular = poco->DiametroPoco() - poco->
56             DiametroRevestimentoOD();
57     vMediaAnular = DeterminarVelocidadeMediaAnular(poco->Vazao(),
58             poco->DiametroPoco(), poco->DiametroRevestimentoOD());
59     reynoldsAnular = DeterminarReynoldsAnular(poco->
```

```
        DensidadeEfetivaTotal(), vMediaAnular, diametroAnular,
        indiceDeConsistencia, indiceDeComportamento);

45
46     fluxoAnular = (reynoldsAnular <= reynoldsCriticoAnular) ? "
47         Laminar" : "Turbulento"; // Determinação do fluxo
48     return fluxoAnular;
49 }

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

71
72     double diametroAnular = poco->DiametroPoco() - poco->
73         DiametroRevestimentoOD();
74     fatorFriccaoAnular = DeterminarFatorFriccao(reynoldsAnular);
75
76     if (fluxoAnular == "Laminar") {
```

```
76     return ((indiceDeConsistencia * std::pow(vMediaAnular, -
77         indiceDeComportamento)) * std::pow(( (2+(1/
78             indiceDeComportamento) ) / 0.0208),
79             indiceDeComportamento))
80     / (144000 * std::pow(diametroAnular, (1+
81         indiceDeComportamento)));
82 } else { // Fluxo turbulento
83     return (fatorFriccaoAnular * poco->DensidadeEfetivaTotal()
84         * std::pow(vMediaAnular, 2) ) / (21.1 * diametroAnular);
85 }
```

---

Fonte: produzido pelo autor.

# Capítulo 7

## Resultados

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

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

### 7.1 Propriedades da simulação

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

#### 7.1.1 Configurações para o regime turbulento

Abaixo na Tabela 7.1 temos as propriedades do poço.

Tabela 7.1: Configuração do poço turbulento

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

Abaixo na Tabela 7.2 temos as propriedades dos fluidos.

Tabela 7.2: Configuração do poço turbulento

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

### 7.1.2 Configurações para o regime laminar

Abaixo na Tabela 7.3 temos as propriedades do poço.

Tabela 7.3: Configuração do poço laminar

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

Abaixo na Tabela 7.4 temos as propriedades dos fluidos.

Tabela 7.4: Configuração do poço turbulento

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

## 7.2 Testes

### 7.2.1 Validação da Perda de Fricção Pelo Modelo Newtoniano no Poço

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

Para o fluxo laminar no poço realizamos os seguinte cálculos:

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

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

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

Na Figura 7.1 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.1: Soluções geradas pelo código para Modelo Newtoniano no Poço considerando Regime Laminar

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

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

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

Para o fluxo turbulento no poço realizamos os seguinte cálculos:

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

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

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

Na Figura 7.2 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.2: Soluções geradas pelo código para Modelo Newtoniano no Poço considerando Regime Turbulento

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

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

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

Fator de Friccao no Poco: 0.00543119

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

### 7.2.2 Validação da Perda de Fricção Pelo Modelo Newtoniano no Anular

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

Para o fluxo laminar no anular realizamos os seguinte cálculos:

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

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

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

Na Figura 7.3 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.3: Soluções geradas pelo código para Modelo Newtoniano no Anular considerando Regime Laminar

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

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

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

Para o fluxo turbulento no anular realizamos os seguinte cálculos:

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

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

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

Na Figura 7.4 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.4: Soluções geradas pelo código para Modelo Newtoniano no Anular considerando Regime Turbulento

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

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

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

Fator de Friccao no Anular: 0.00698677

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

### 7.2.3 Validação da Perda de Fricção Pelo Modelo Bingham no Poco

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

Para o fluxo laminar no poço realizamos os seguinte cálculos:

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

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

$$N_{rec} = 33000$$

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

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

Na Figura 7.5 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.5: Soluções geradas pelo código para Modelo Plástico de Bingham no Poço considerando Regime Laminar

Fonte: Produzido pelo autor.

Para o fluxo turbulento no poço realizamos os seguinte cálculos:

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

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

$$N_{rec} = 27000$$

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

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

Na Figura 7.6 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.6: Soluções geradas pelo código para Modelo Plástico de Bingham no Poço considerando Regime Turbulento

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

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

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

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

Fator de Friccao: 0.005478

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

#### 7.2.4 Validação da Perda de Fricção Pelo Modelo Bingham no Anular

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

Para o fluxo laminar no anular realizamos os seguinte cálculos:

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

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

$$N_{rec} = 6000$$

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

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

Na Figura 7.7 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.7: Soluções geradas pelo código para Modelo Plástico de Bingham no Anular considerando Regime Laminar

Fonte: Produzido pelo autor.

Para o fluxo turbulento no anular realizamos os seguinte cálculos:

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

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

$$N_{rec} = 5000$$

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

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

Na Figura 7.8 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.8: Soluções geradas pelo código para Modelo Plástico de Bingham no Anular considerando Regime Turbulento

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

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

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

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

Fator de Friccao: 0.0070541

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

### 7.2.5 Validação da Perda de Fricção Pelo Modelo de Potência no Poco

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

Para o fluxo laminar no poço realizamos os seguinte cálculos:

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

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

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

Na Figura 7.9 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.9: Soluções geradas pelo código para Modelo Lei de Potência no Poço considerando Regime Laminar

Fonte: Produzido pelo autor.

Para o fluxo turbulento no poço realizamos os seguinte cálculos:

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

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

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

Na Figura 7.10 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.10: Soluções geradas pelo código para Modelo Lei de Potência no Poço considerando Regime Turbulento

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

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

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

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

Fator de Friccao: 0.010983

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

### 7.2.6 Validação da Perda de Fricção Pelo Modelo de Potência no Anular

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

Para o fluxo laminar no poço realizamos os seguinte cálculos:

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

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

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

Na Figura 7.9 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.11: Soluções geradas pelo código para Modelo Lei de Potência no Anular considerando Regime Laminar

Fonte: Produzido pelo autor.

Para o fluxo turbulento no poço realizamos os seguinte cálculos:

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

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

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

Na Figura 7.12 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.12: Soluções geradas pelo código para Modelo Lei de Potência no Anular considerando Regime Turbulento

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

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

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

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

Fator de Friccao: 0.0121322

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

### 7.2.7 Validação da Pressão hidrostática no Fundo do Poço

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

Foram realizados os seguintes cálculos:

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

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

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

Na Figura 7.13 temos os resultados obtidos pelo software que podem ser validados pelos cálculos acima.

Figura 7.13: Soluções geradas pelo código para pressão hidrostática no fundo do poço

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

Fonte: Produzido pelo autor.

# Capítulo 8

## 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.

### 8.1 Documentação do usuário

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

### 8.2 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 8.1 mostra uma imagem da documentação gerada.

Figura 8.1: 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 8.2

Figura 8.2: Código fonte da classe CSimuladorTemperatura, 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.



# Referências Bibliográficas



