

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

PROJETO DE ENGENHARIA  
DESENVOLVIMENTO DO 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  
Dezembro - 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 . . .	20
2.2	Coluna composta por fluidos com diferentes características . . . . .	22
2.3	Fluxo laminar de fluido Newtoniano . . . . .	24
2.4	Diagrama de pacotes . . . . .	30
3.1	Diagrama de classes . . . . .	33
3.2	Diagrama de sequência . . . . .	36
3.3	Diagrama de comunicação . . . . .	37
3.4	Diagrama de máquina de estado . . . . .	38
3.5	Diagrama de atividades . . . . .	38
4.1	Diagrama de componentes . . . . .	40
5.1	Versão 0.1, interface do software . . . . .	42
7.1	Soluções geradas pelo código para modelo Newtoniano no poço considerando regime laminar . . . . .	91
7.2	Soluções geradas pelo código para modelo Newtoniano no poço considerando regime turbulento . . . . .	92
7.3	Soluções geradas pelo código para modelo Newtoniano no anular considerando regime laminar . . . . .	93
7.4	Soluções geradas pelo código para modelo Newtoniano no anular considerando regime turbulento . . . . .	94
7.5	Soluções geradas pelo código para modelo plástico de Bingham no poço considerando regime laminar . . . . .	95
7.6	Soluções geradas pelo código para modelo plástico de Bingham no poço considerando regime turbulento . . . . .	96

7.7	Soluções geradas pelo código para modelo plástico de Bingham no anular considerando regime laminar . . . . .	97
7.8	Soluções geradas pelo código para modelo plástico de Bingham no anular considerando regime turbulento . . . . .	98
7.9	Soluções geradas pelo código para modelo lei de potência no poço considerando regime laminar . . . . .	99
7.10	Soluções geradas pelo código para modelo lei de potência no poço considerando regime turbulento . . . . .	100
7.11	Soluções geradas pelo código para modelo lei de potência no anular considerando regime laminar . . . . .	101
7.12	Soluções geradas pelo código para modelo lei de potência no anular considerando regime turbulento . . . . .	102
7.13	Soluções geradas pelo código para pressão hidrostática no fundo do poço .	103
7.14	Gráfico pressão hidrostática por profundidade . . . . .	104
7.15	Terminal mostrando a funcionalidade armazenar os resultados . . . . .	105
7.16	Terminal mostrando a funcionalidade armazenar os resultados . . . . .	105
8.1	Logo e documentação do <i>software</i> . . . . .	107
8.2	Código fonte da classe CSimuladorPoco, no <i>Doxygen</i> . . . . .	107
9.1	Imagen da interface gráfica . . . . .	109
9.2	Configurar o poço no simulador: funções de criar o poço e impostar o dados a partir de um arquivo .dat . . . . .	111
9.3	Arquivo .dat para adicionar propriedades do poço e dos fluidos no simulador	111

# **Lista de Tabelas**

1.1	Características básicas do programa . . . . .	9
1.2	Caso de uso 1 . . . . .	11
7.1	Configuração do poço turbulento . . . . .	89
7.2	Configuração do poço turbulento . . . . .	90
7.3	Configuração do poço laminar . . . . .	90
7.4	Configuração do poço turbulento . . . . .	90

# 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	Termos e unidades . . . . .	18
2.2.2	Hidráulica de perfuração . . . . .	19
2.2.3	Pressão hidrostática . . . . .	20
2.2.4	Pressão hidrostática em colunas com mais de um tipo de fluido . . .	21
2.2.5	Densidade equivalente . . . . .	23
2.2.6	Modelos reológicos de fluidos de perfuração . . . . .	23
2.2.7	Perda de pressão fricional em um tubo de perfuração . . . . .	26
2.2.8	Perda de pressão fricional em um anular . . . . .	27
2.3	Identificação de pacotes – assuntos . . . . .	29
2.4	Diagrama de pacotes – assuntos . . . . .	29
<b>3</b>	<b>AOO – análise orientada a objeto</b>	<b>32</b>
3.1	Diagramas de classes . . . . .	32
3.1.1	Dicionário de classes . . . . .	34
3.2	Diagrama de sequência – eventos e mensagens . . . . .	34
3.2.1	Diagrama de sequência geral . . . . .	35
3.3	Diagrama de comunicação – colaboração . . . . .	37
3.4	Diagrama de máquina de estado . . . . .	37
3.5	Diagrama de atividades . . . . .	38

<b>4 Projeto</b>	<b>39</b>
4.1 Projeto do sistema . . . . .	39
4.2 Diagrama de componentes . . . . .	39
<b>5 Ciclos de planejamento/detalhamento</b>	<b>41</b>
5.1 Versão 0.1 - Uso modo terminal e <i>Gnuplot</i> para saída de gráfico . . . . .	41
<b>6 Ciclos construção - implementação</b>	<b>43</b>
6.1 Versão 0.1 - código fonte - . . . . .	43
<b>7 Resultados</b>	<b>89</b>
7.1 Propriedades da simulação . . . . .	89
7.1.1 Configurações para o regime turbulento . . . . .	89
7.1.2 Configurações para o regime laminar . . . . .	90
7.2 Testes . . . . .	90
7.2.1 Validação da perda de fricção pelo modelo Newtoniano no poço . . . . .	90
7.2.2 Validação da perda de fricção pelo modelo Newtoniano no anular . . . . .	92
7.2.3 Validação da perda de fricção pelo modelo Bingham no poco . . . . .	94
7.2.4 Validação da perda de fricção pelo modelo Bingham no anular . . . . .	96
7.2.5 Validação da perda de fricção pelo modelo de potência no poco . . . . .	98
7.2.6 Validação da perda de fricção pelo modelo de potência no anular . . . . .	100
7.2.7 Validação da pressão hidrostática no fundo do poço . . . . .	102
7.2.8 Validação da funcionalidade impressão . . . . .	104
<b>8 Documentação</b>	<b>106</b>
8.1 Documentação do usuário . . . . .	106
8.2 Documentação do desenvolvedor . . . . .	106
<b>9 Manual do usuário</b>	<b>108</b>
9.1 Instalação . . . . .	108
9.1.1 Dependências . . . . .	108
9.2 Interface gráfica . . . . .	108
9.3 Funcionalidades . . . . .	109
9.4 Como adicionar informações do poço/fluido . . . . .	110
<b>Referências Bibliográficas</b>	<b>112</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 plotar resultados. Saída gráfica e em arquivo .dat.
<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 pelo software se baseiam em equações desenvolvidas na disciplina de Engenharia de poço.

Inicialmente o usuário precisara 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 propriedades 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 como Mitchell & Miska (2011) e Jr. <i>et al.</i> (1991).
--------------	---

<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 devem suportar os sistemas operacionais Linux e <i>Windows</i> .
<b>RNF-02</b>	Plotagem de diferentes gráficos para representação do estudo analisado durante a simulação através do <i>Gnuplot</i> .
<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 apresentadas na Tabela 1.2.

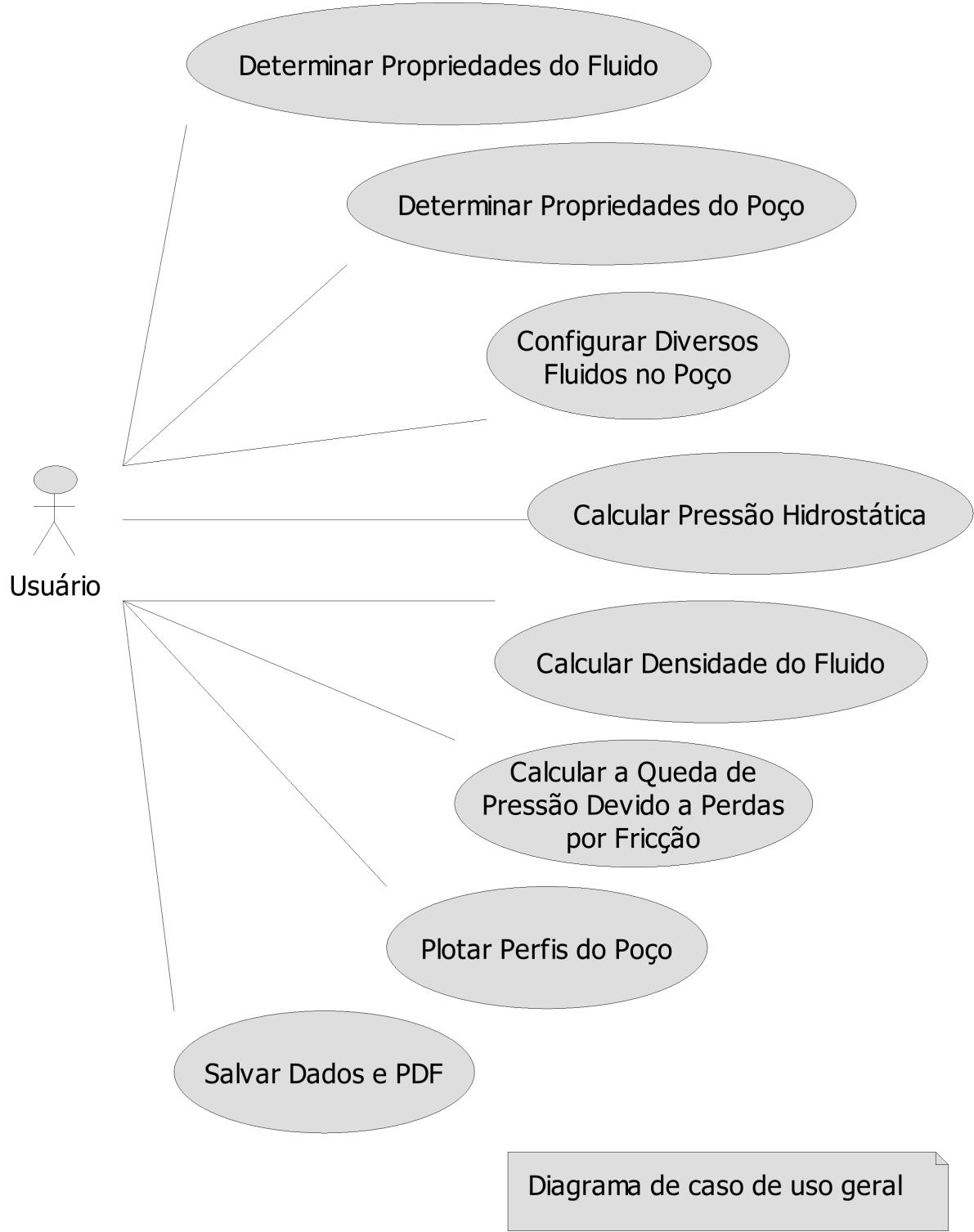
Tabela 1.2: Caso de uso 1

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 devido a perdas por fricção</li><li>7. Plotar perfis de poço</li><li>7. Salvar dados em arquivos .dat</li></ol>

#### 1.4.2 Diagrama de caso de uso específico

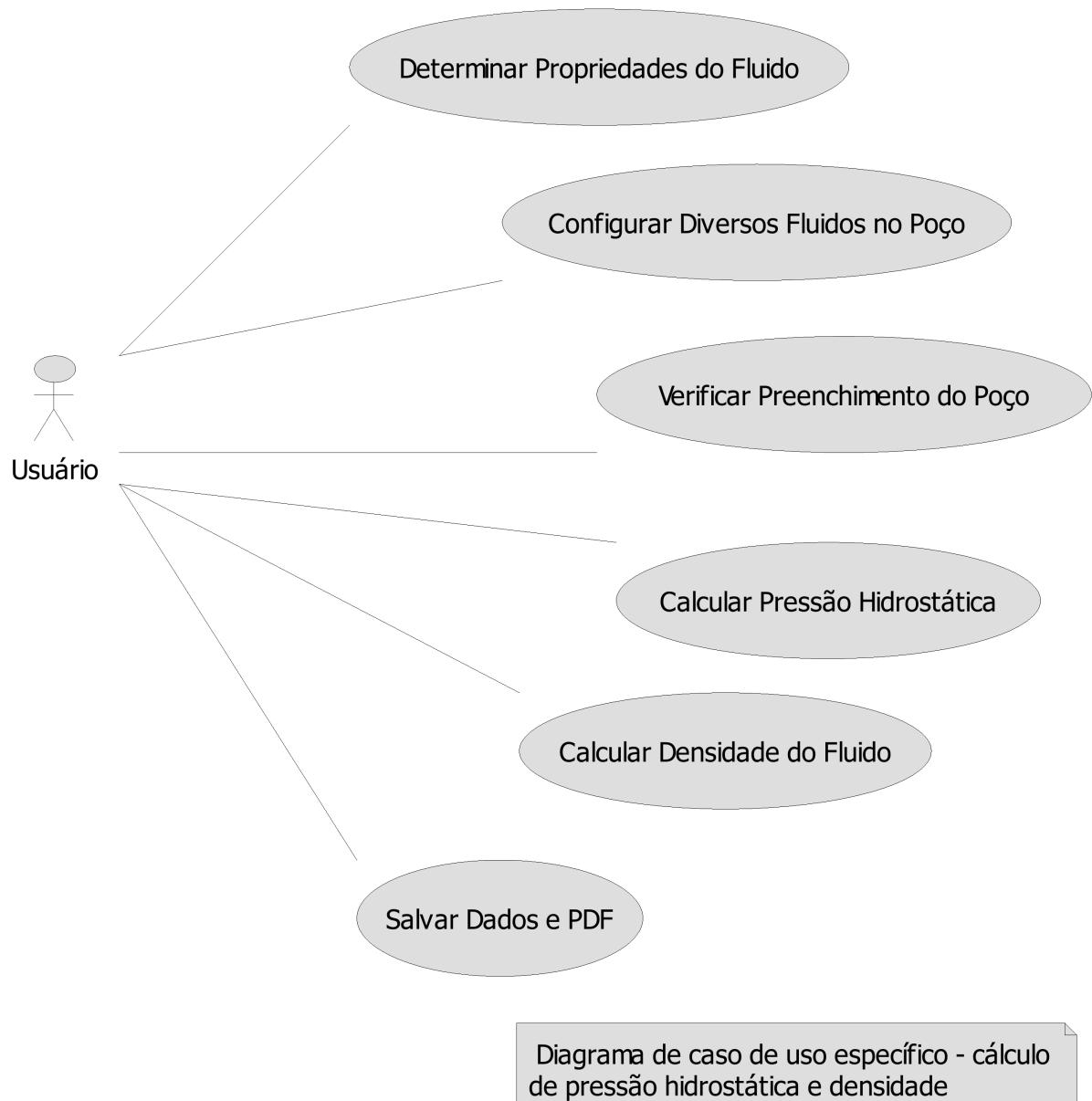
O caso de uso específico da Figura 1.2 mostra o cenário onde o usuário deseja calcular a pressão hidrostática e a densidade dos fluidos configurados no poço.

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

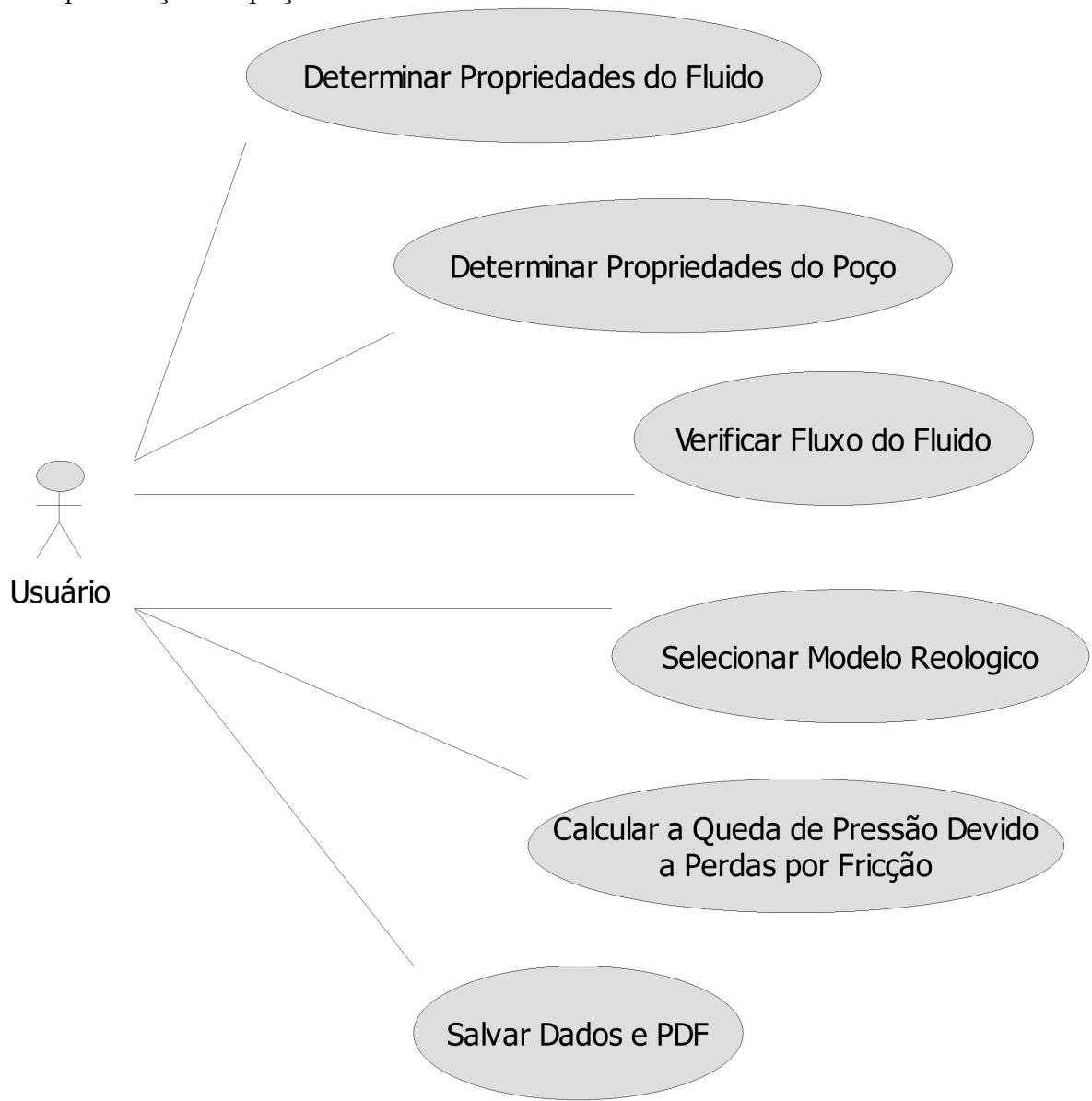


Diagrama de caso de uso específico - cálculo da perda de pressão devido a perda por fricção no poço e no anular

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 desenvolverem 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 Gnuplot (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 Termos e unidades

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

- $dp$  é a variação de pressão [ $\text{psi}$ ];
- $dZ$  é a variação de profundidade [ $\text{ft}$ ];
- $\rho$  é a densidade do fluido [ $\text{lbm/gal}$ ];
- $p_0$  é a constante de integração igual a pressão na superfície [ $\text{psi}$ ];
- $p$  é a pressão [ $\text{psi}$ ];
- $Z$  é a profundidade [ $\text{ft}$ ];
- $z$  é o fator de desvio de gás;
- $R$  é constante universal dos gases [ $\text{psi} \cdot \text{ft}^3/\text{lb} - \text{mol} \cdot ^\circ\text{R}$ ];
- $T$  é a temperatura absoluta [ $^\circ\text{R}$ ];
- $M$  é o peso molecular do gás [ $\text{lb/lb} - \text{mol}$ ];
- $\Delta Z$  é a variação de profundidade [ $\text{ft}$ ];
- $g$  é a gravidade [ $\text{ft}/\text{s}^2$ ];
- $v$  velocidade [ $\text{ft}/\text{s}$ ];
- $\tau$  é a tensão de cisalhamento exercida sobre o fluido [ $\text{psi}$ ];
- $\mu$  é a viscosidade aparente [ $\text{cP}$ ];
- $\dot{\gamma}$  é a taxa de cisalhamento [ $1/\text{s}$ ];
- $\tau_y$  é a tensão de escoamento ou o ponto de escoamento [ $\text{lbf}/100.\text{sq.ft}$ ];
- $\mu_p$  é a viscosidade plástica [ $\text{cP}$ ];

- $K$  é o índice de consistência do fluido [ $cP$ ];
- $n$  é o expoente da lei de potência ou o índice de comportamento do fluxo;
- $N_{re}$  é o número de Reynolds;
- $d$  é o diâmetro interno do revestimento ID [in];
- $\bar{v}$  é a velocidade média [ $ft/s$ ];
- $q$  é a vazão do poço [ $gal/min$ ];
- $\frac{dp_f}{dL}$  é a perda de pressão por fricção [ $psi/ft$ ];
- $f$  é o fator de fricção;
- $\tau_w$  é a tensão de cisalhamento na parede [ $lbf/ft^2$ ];
- $N_{rec}$  é o número de Reynolds crítico;
- $N_{He}$  é o número de Hedstrom;
- $d_1$  é o diâmetro externo do revestimento OD [in];
- $d_2$  é o diâmetro do poço [in];  $d_1$  é o diâmetro externo do revestimento OD [in];
- $d_2$  é o diâmetro do poço [in];

### 2.2.2 Hidráulica de perfuração

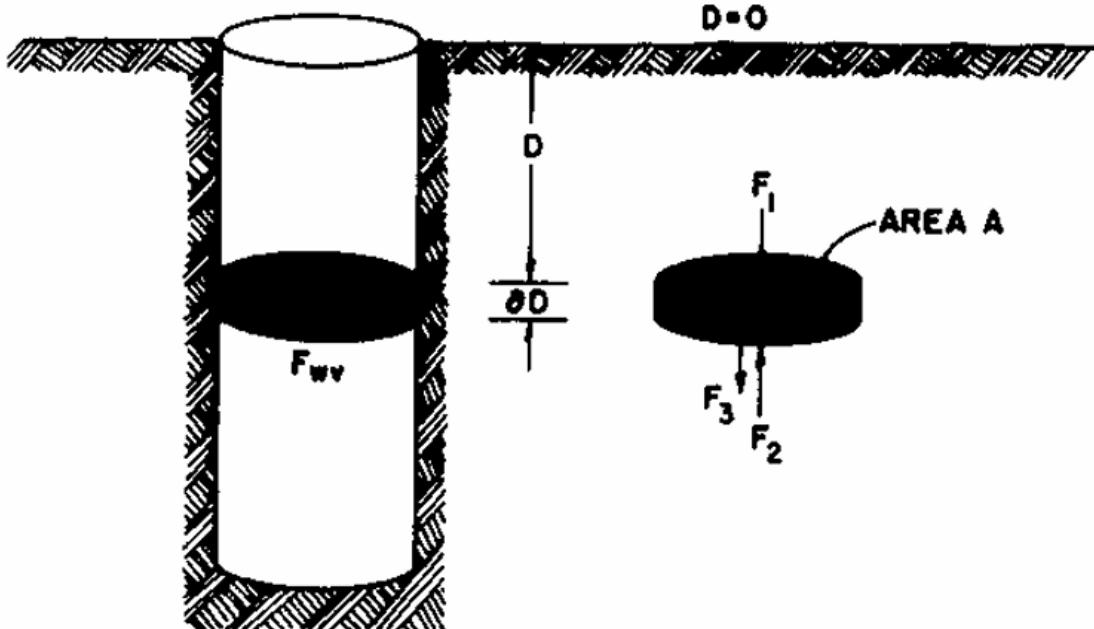
Na engenharia de perfuração um fluido de perfuração tem três funções principais: transportar cascalho, prevenir o influxo de fluidos e manter a estabilidade do poço. Para que possa cumprir tais funções o fluido depende do seu fluxo na tubulação e das pressões associadas a esse fluxo. Para que um engenheiro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

### 2.2.3 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



Fonte Jr. et al. (1991)

A partir dessa dedução chegamos à Equação (2.1) a seguir em unidades *oil field*, onde  $dp$  é a variação de pressão [ $psi$ ],  $dZ$  é a variação de profundidade [ $ft$ ] e  $\rho$  é a densidade do fluido [ $lb/gal$ ].

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

Podemos calcular a pressão hidrostática para dois tipos 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  $p_0$  é a constante de integração, é igual a pressão na superfície [psi],  $p$  é a pressão [psi] e  $Z$  é a profundidade [ft]. 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 *kicks* ou *blowouts*, além de não causar fraturas na formação que poderia provocar uma perda de circulação de fluido que também é indesejada Jr. *et al.* (1991).

### 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 uma 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:

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

Onde  $z$  é o fator de desvio de gás,  $R$  é constante universal dos gases [psi.ft<sup>3</sup>/lb - mol.°R],  $T$  é a temperatura absoluta [°R] e  $M$  é o peso molecular do gás [lb/lb - mol].

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:

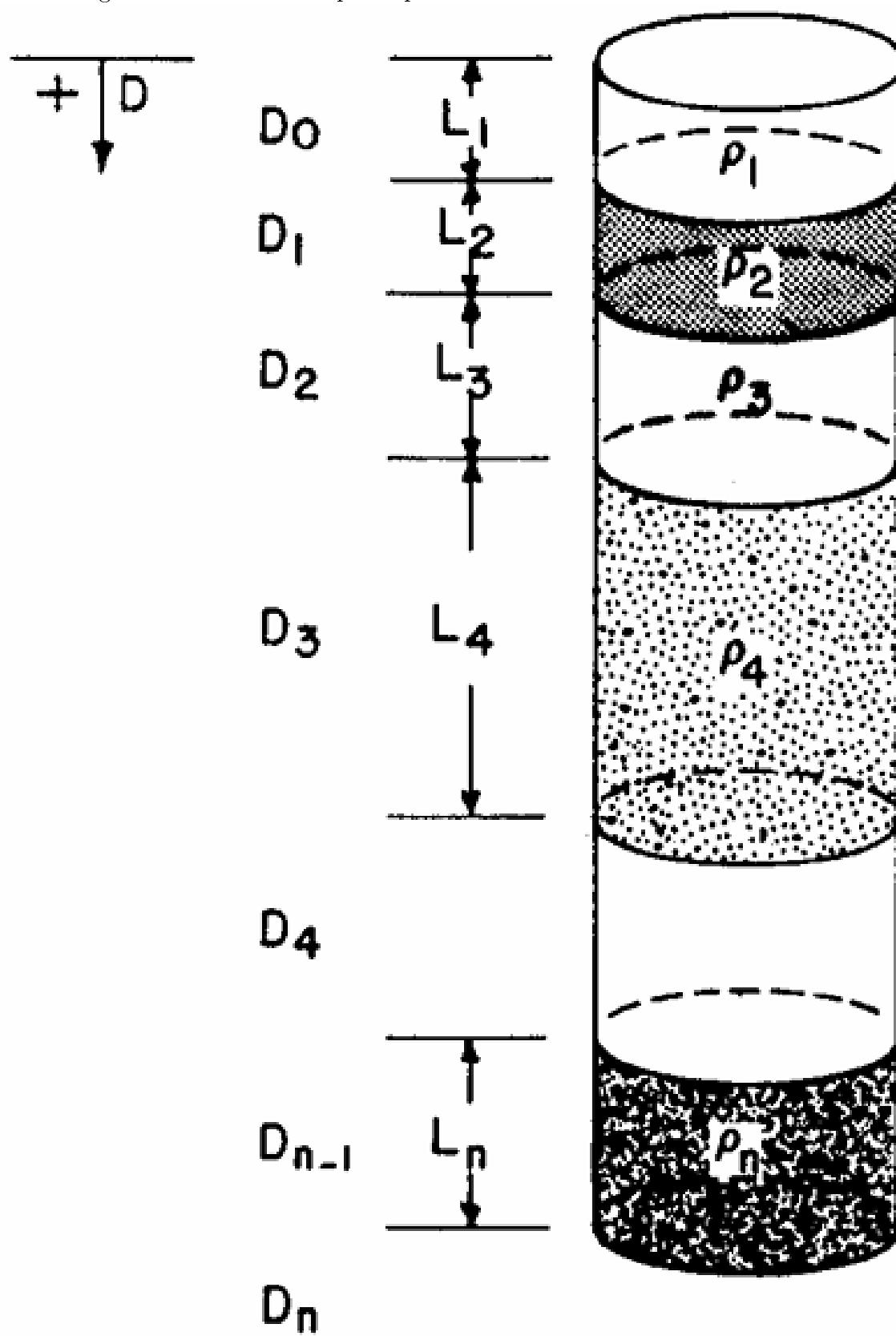
$$p = p_0 \exp\left(\frac{M\Delta Z}{1544zT}\right) \quad (2.4)$$

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

#### 2.2.4 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



Fonte Jr. et al. (1991)

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

ção:

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

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

### 2.2.5 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.

### 2.2.6 Modelos reológicos de fluidos de perfuração

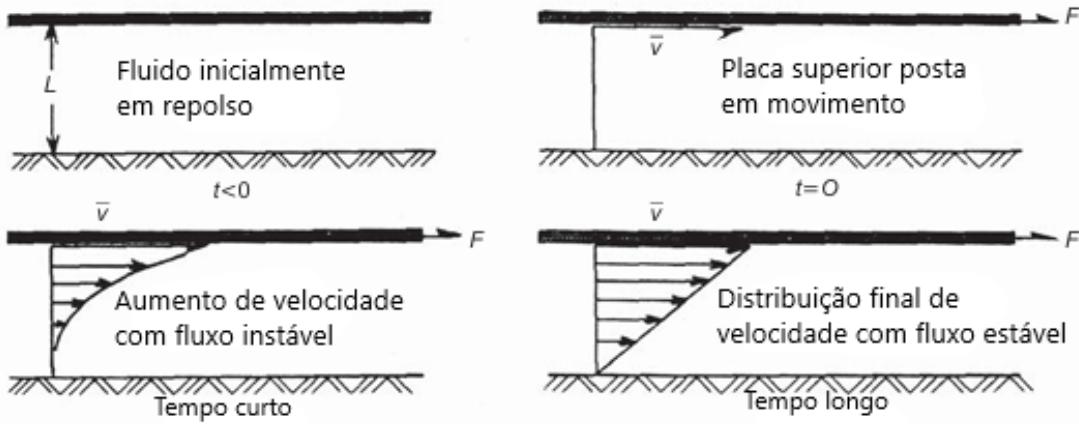
Durante o processo de perfuração de um poço muitas vezes forças viscoelásticas extremamente grandes precisam ser vencidas para que o fluido de perfuração se move pelas conduites longos e finos utilizados nesse processo, dessa forma se faz essencial a análise da perda de pressão por atrito. Na maioria dos casos as propriedades elásticas dos fluidos de perfuração e seus efeitos durante o fluxo de um poço são desprezíveis, sendo consideradas para o cálculo apenas as forças viscósas. 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 Mitchell & Miska (2011). É importante ressaltar a existência de outros modelos que podem futuramente ser acrescentados no aprimoramento desse projeto.

#### Visão geral dos modelos reológicos

As forças viscósas 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



Adaptado de Mitchell & Miska (2011)

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

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

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

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

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

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

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 Mitchell & Miska (2011).

## Modelo de fluido Newtoniano

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

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

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 será 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.

## Modelo de fluidos plásticos de Bingham

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

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

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 [ $lbf/100.sq.ft$ ]. Após a tensão de escoamento o fluido de Bingham se comporta como um fluido Newtoniano onde a mudança na tensão de cisalhamento é proporcional a mudança na taxa de cisalhamento. A constante de proporcionalidade  $\mu_p$  é chamada de viscosidade plástica [ $cP$ ].

## Modelo fluidos de lei de potência

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

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

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

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

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

### Modelo de fluido Newtoniano

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

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

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

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

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

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

Para fluxo laminar:

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

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:

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

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

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

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

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

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).

### 2.2.8 Perda de pressão friccional em um anular

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

## Modelo de fluido Newtoniano

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

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

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

A velocidade média pode ser obtida pela equação:

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

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 pode 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

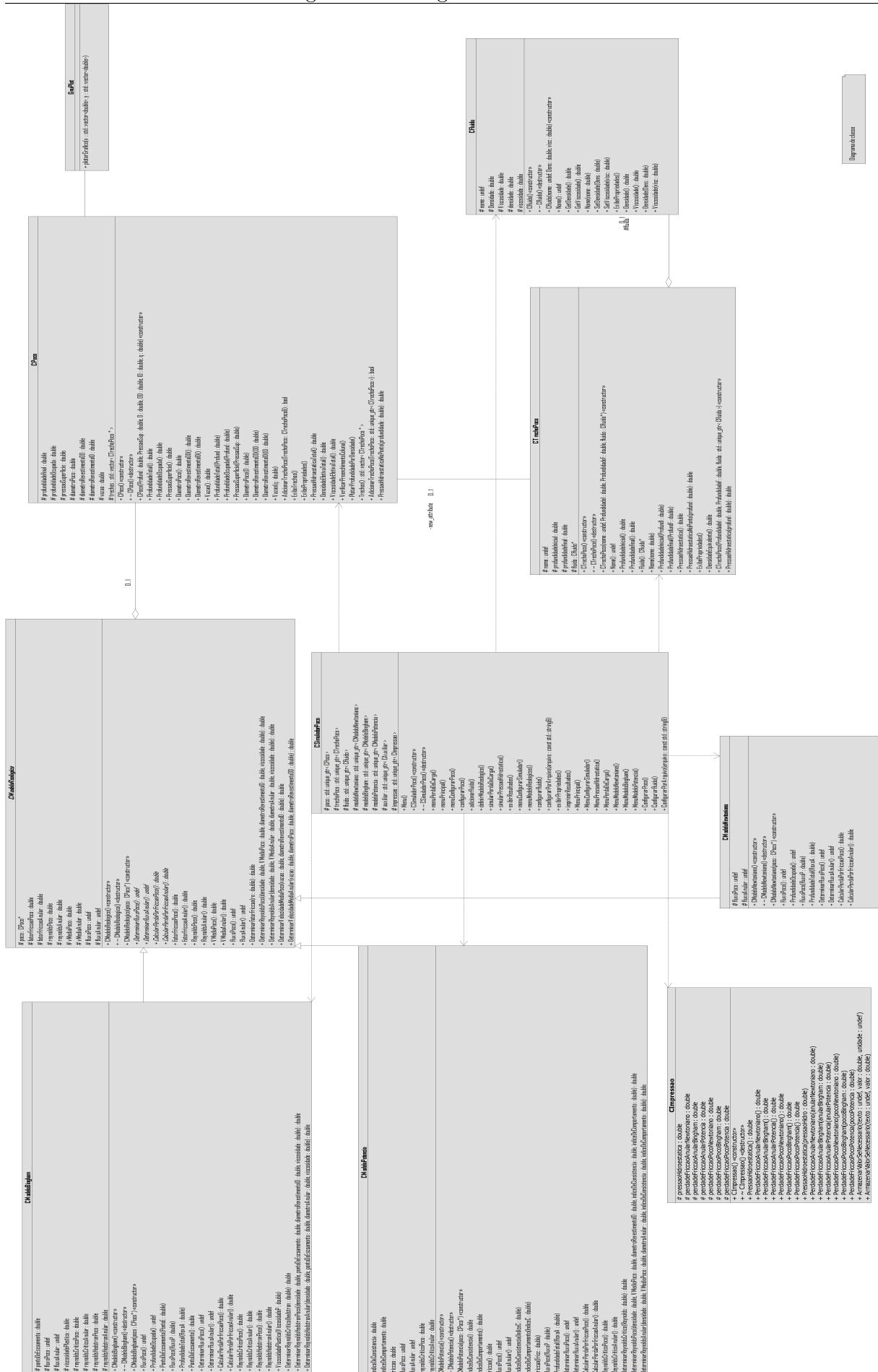
## AOO – análise orientada a objeto

Neste capítulo são apresentadas 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 BUENO (2003).

### 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 10 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.
- **CModeloReologico:** 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.
- **CImpressao:** Classe responsável por armazenar e salvar os resultados da simulação em um arquivo .dat.
- **CGnuplot:** Classe responsável por gerar gráficos utilizando a biblioteca *Gnuplot*, 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.

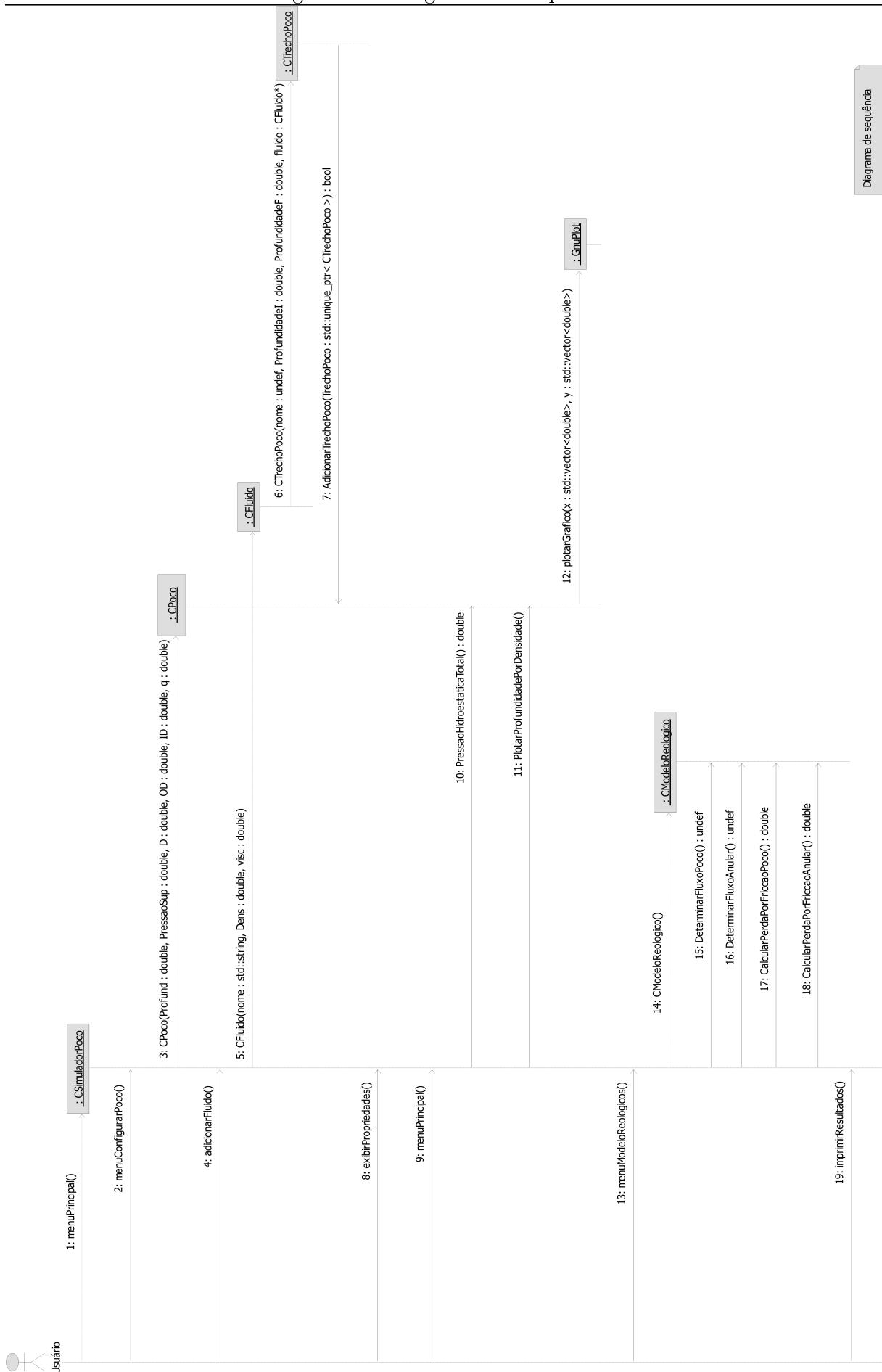
## 3.2 Diagrama de sequência – eventos e mensagens

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 sequência

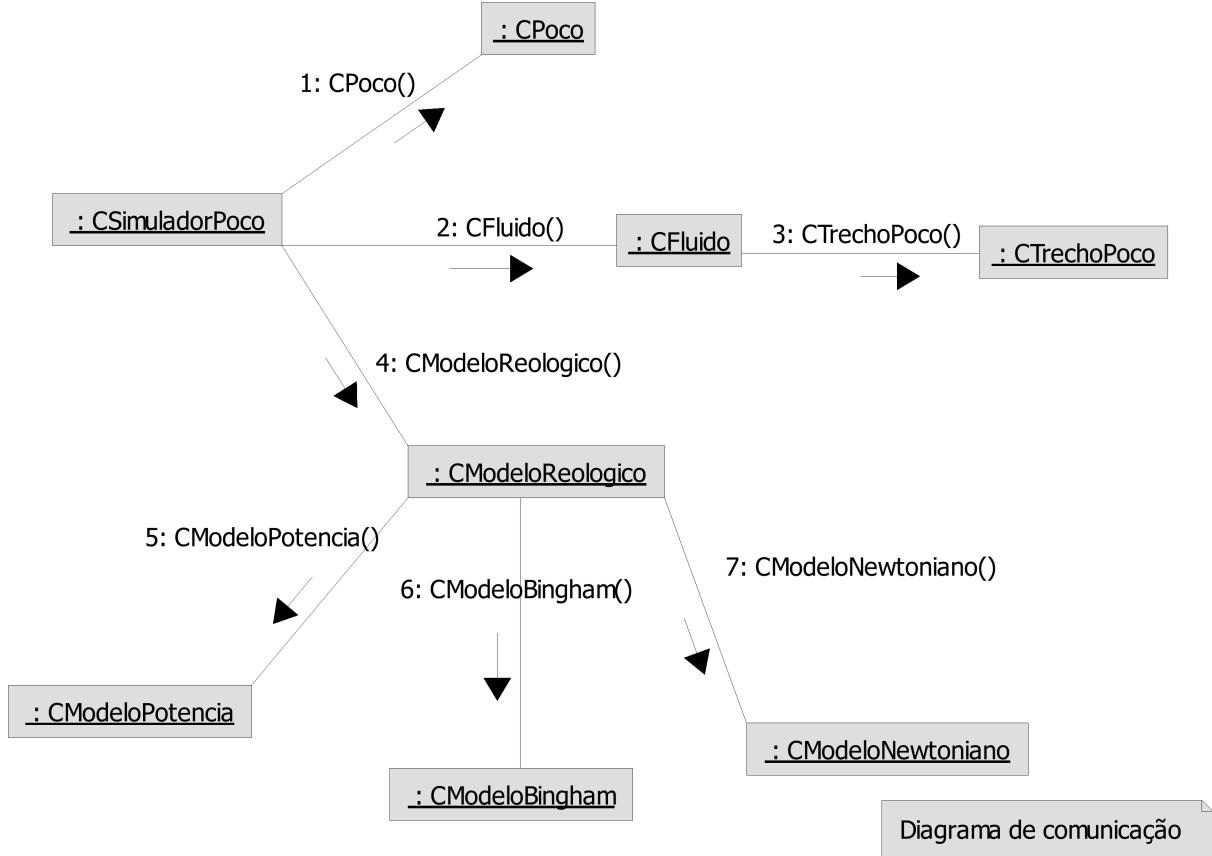


Fonte: Produzido pelo autor.

### 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



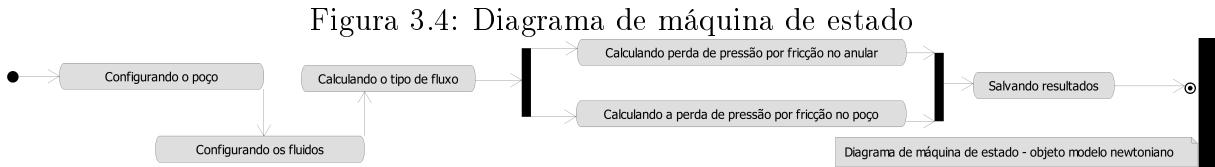
Fonte: Produzido pelo autor.

### 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.



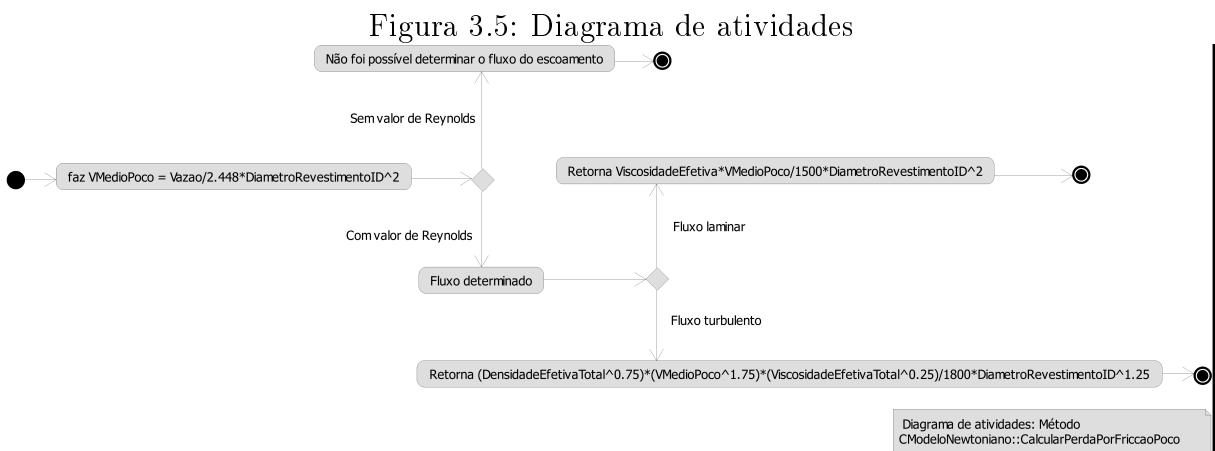
Fonte: Produzido pelo autor.

### 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.



Fonte: Produzido pelo autor.

# 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 a objeto.

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

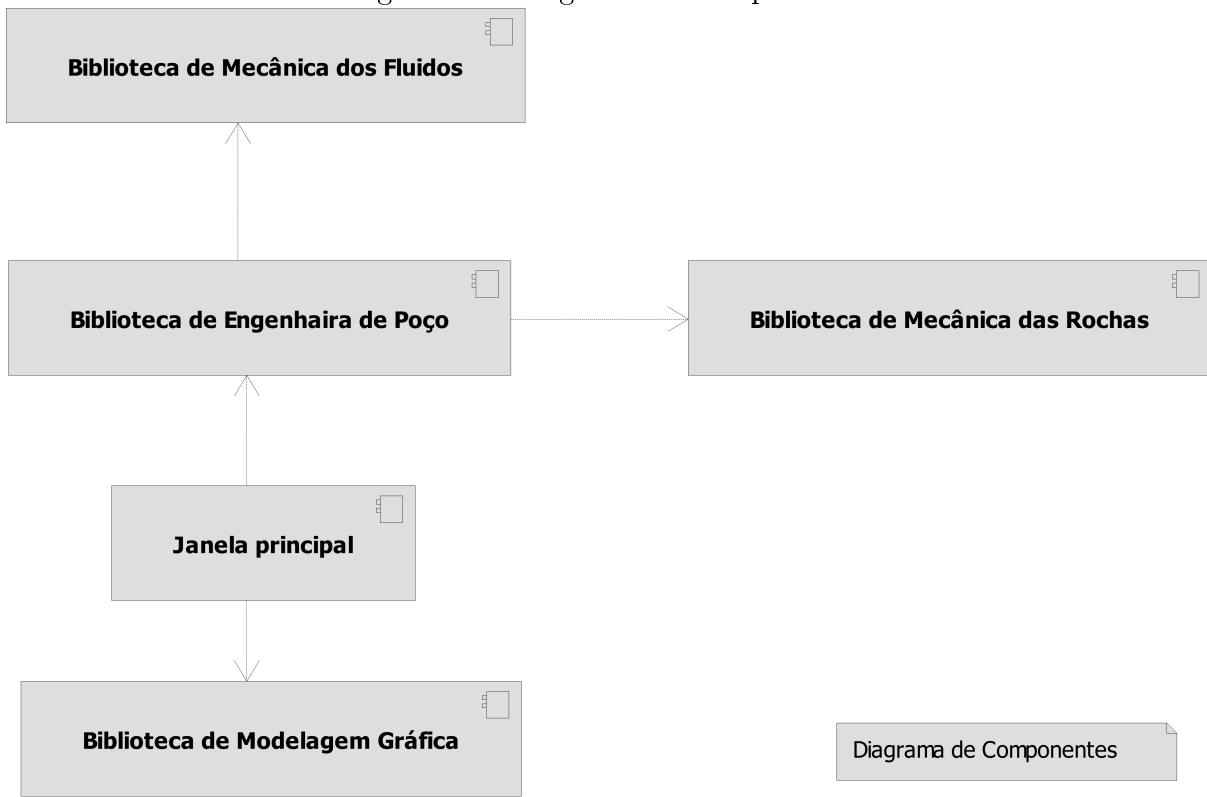
- Linguagem de programação de alto desempenho, adequada para cálculos numéricos intensivos.
- Amplo suporte a orientação a objeto e diversos vínculos com UML.
- Dezenas de bibliotecas de suporte prontas, gráficos (biblioteca *Gnuplot*) e bibliotecas para gerar saídas em arquivo .dat.
- Programação de alto nível com vários graus de abstração.
- Diversos ambientes de desenvolvimento - IDEs, compiladores, *debuggers*, *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

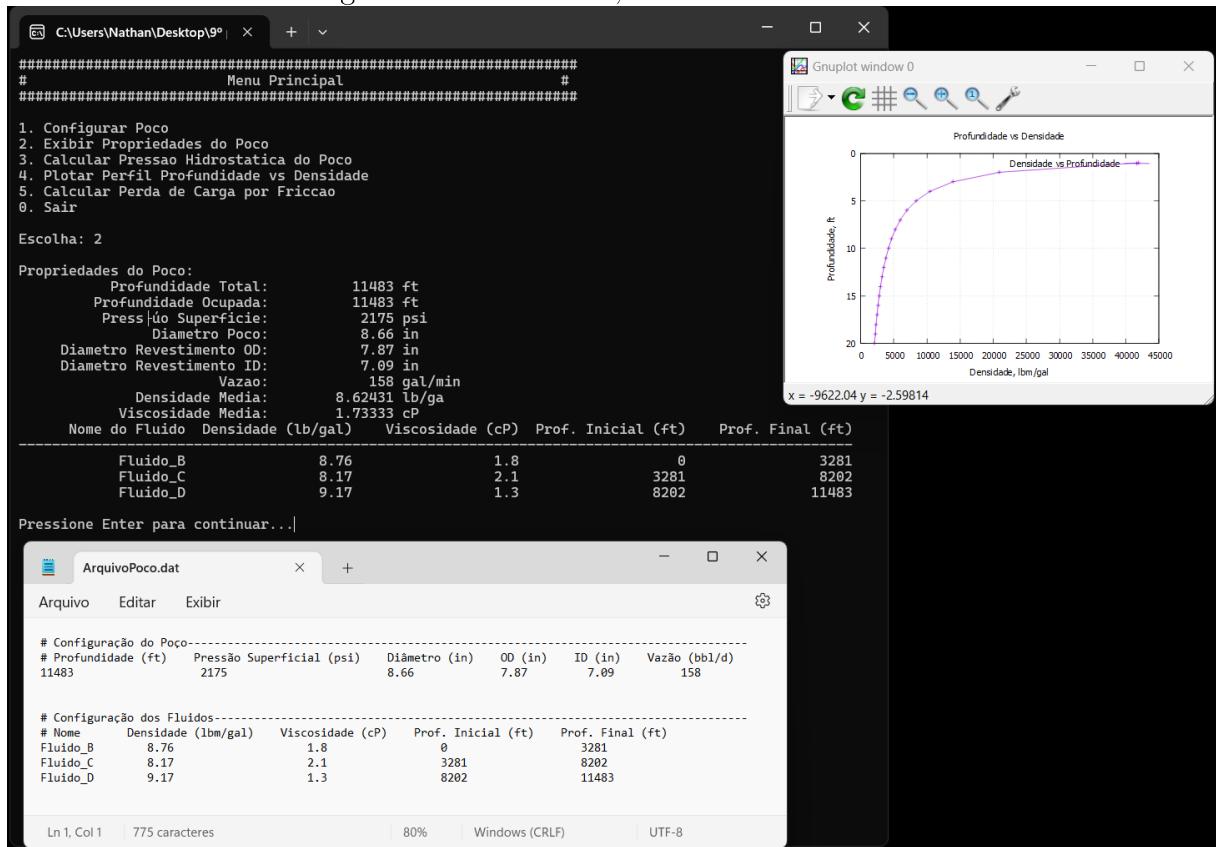
Apresenta-se neste capítulo as versões do software desenvolvido.

### 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 exibidas as classes necessárias para a interação via terminal.

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

Apresenta-se na listagem 6.1 o arquivo com código da função *main*.

---

Listing 6.1: Arquivo de implementação da função main

---

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

---

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 "CAuxiliar.h"
10 #include "CImpressao.h"
11 #include <memory>
12 #include <vector>
13
14 class CSimuladorPoco {
15 protected:
16     std::unique_ptr<CPoco> poco;
17     std::unique_ptr<CTrechoPoco> trechoPoco;
18     std::unique_ptr<CFluido> fluido;
19     std::unique_ptr<CModeloNewtoniano> modeloNewtoniano;
20     std::unique_ptr<CModeloBingham> modeloBingham;
21     std::unique_ptr<CModeloPotencia> modeloPotencia;
22     std::unique_ptr<CAuxiliar> auxiliar;
23     std::unique_ptr<CImpressao> impressao;
24
25
26 public:
27     // Construtor e destrutor
28     CSimuladorPoco() {}
29     ~CSimuladorPoco() {}
30
31     // Menus principais
32     void MenuPrincipal();
33     void MenuConfigurarSimulador();
34     void MenuPressaoHidrostatica();
35     void MenuPerdaDeCarga();
36     void MenuModeloNewtoniano();
37     void MenuModeloBingham();
38     void MenuModeloPotencia();
39
40
41     // Metodos auxiliares para configurar o poco e fluidos
42     void ConfigurarPoco();
43     void ConfigurarFluido();
44     void ConfigurarPorArquivo(const std::string& arquivo);
45     void ImprimirResultados();
```

```

46
47     // Metodos de simulacao
48     void ExibirPropriedades();
49
50 };
51
52 #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 #include <iostream>
2 #include <iomanip>
3 #include <string>
4 #include <memory>
5 #include <vector>
6 #include <fstream>
7 #include <sstream>
8 #include "CAuxiliar.h"
9 #include "CSimuladorPoco.h"

10
11 auto auxiliar = std::make_unique();
12
13 void CSimuladorPoco::ExibirPropriedades() {
14     // Exibir propriedades do pôco
15     std::cout << "\nPropriedades do Pôco:" << std::endl;
16     std::cout << std::setw(30) << "Profundidade Total:"
17         << std::setw(15) << poco->ProfundidadeTotal()
18             << " ft" << std::endl;
19
20     std::cout << std::setw(30) << "Profundidade Ocupada:"
21         << std::setw(15) << poco->ProfundidadeOcupada()
22             << " ft" << std::endl;
23
24     std::cout << std::setw(30) << "Pressão Superfície:"
25         << std::setw(15) << poco->PressaoSuperficie()
26             << " psi" << std::endl;
27
28     std::cout << std::setw(30) << "Diâmetro Pôco:"
29         << std::setw(15) << poco->DiâmetroPoco() << " "
30             in" << std::endl;

```



```
57             << std::setw(20) << trecho->Fluido()->
58                 Viscosidade()
59             << std::setw(20) << trecho->
60                 ProfundidadeInicial()
61             << std::setw(20) << trecho->
62                 ProfundidadeFinal() << std::endl;
63     }
64
65 void CSimuladorPoco::ConfigurarPoco() {
66
67     bool pocoConfigurado = (poco != nullptr);
68     bool criarNovoPoco = false;
69
70     double profundidade, pressaoSuperficie, diametro, OD, ID, vazao
71     ;
72
73     if (!pocoConfigurado) {
74         std::cout << "\nInforme a profundidade total do poço [ft]: "
75         ";
76         std::cin >> profundidade;
77     }
78     else {
79         std::cout << "\nInforme a profundidade total do poço [ft] "
80         "(0 para manter a profundidade anterior): ";
81         std::cin >> profundidade;
82         if (profundidade == 0) {
83             profundidade = poco->ProfundidadeTotal();
84         }
85         else{
86             criarNovoPoco = true;
87         }
88     }
89
90     std::cout << "Informe a pressão na superfície do poço [psia]: "
91     ;
92     std::cin >> pressaoSuperficie;
93
94     std::cout << "Informe o diâmetro do poço [in]: ";
95     std::cin >> diametro;
96
97     std::cout << "Informe o OD do poço [in]: ";
98 }
```

```
92     std::cin >> OD;
93
94     std::cout << "Informe o ID do pôco [in]: ";
95     std::cin >> ID;
96
97     std::cout << "Informe a vazão do pôco [gal/min]: ";
98     std::cin >> vazao;
99
100    if (criarNovoPoco)
101    {
102        // Cria um novo objeto CPoco
103        poco = std::make_unique<CPoco>(profundidade,
104                                         pressaoSuperficie, diametro, OD, ID, vazao);
105    }
106    else{
107        // Atualiza as propriedades do objeto CPoco
108        poco->PressaoSuperficie(pressaoSuperficie);
109        poco->DiametroPoco(diametro);
110        poco->DiametroRevestimentoOD(OD);
111        poco->DiametroRevestimentoID(ID);
112        poco->Vazao(vazao);
113    }
114
115 }
116
117 void CSimuladorPoco::ConfigurarFluido() {
118     std::vector<std::unique_ptr<CTrechoPoco>> trechos; // Vetor
119     para armazenar os trechos
120
121     char continuar;
122
123     do {
124         std::string nome;
125         double densidade, viscosidade, profInicial,
126             profFinal;
127
128         std::cout << "\nInforme um nome para o fluido: ";
129         std::cin >> nome;
```

```
130
131         std::cout << "Informe a viscosidade do fluido:"
132         " ";
133
134         std::cin >> viscosidade;
135
136
137         std::cout << "Informe a profundidade inicial do"
138         " fluido:" ;
139
140         std::cin >> profInicial;
141
142
143         // Cria um novo objeto CFluido e CTrechoPoco
144         auto fluido = std::make_unique<CFluido>(nome,
145             densidade, viscosidade);
146
147         auto trechoPoco = std::make_unique<CTrechoPoco>(profInicial, proffinal, std::move(fluido))
148             ;
149
150
151         // Armazena o trecho no vetor
152         trechos.push_back(std::move(trechoPoco));
153
154
155         std::cout << "Deseja adicionar outro fluido? (s/n): ";
156
157         std::cin >> continuar;
158
159     } while (continuar == 's' || continuar == 'S'); // Continua enquanto o usuário quiser
160
161
162         // Adiciona todos os trechos ao pôco
163         for (auto& trecho : trechos) {
164
165             poco->AdicionarTrechoPoco(std::move(trecho));
166
167         }
168
169     }
170
171
172 void CSimuladorPoco::ConfigurarPorArquivo(const std::string&
173     arquivo) {
174
175     std::ifstream file(arquivo);
176
177     if (!file) {
178
179         std::cerr << "Erro ao abrir o arquivo:" << arquivo << std
```

```
        :: endl;
163    return;
164 }
165
166 std::string linha;
167 bool lendoFluidos = false; // Comeca lendo dados do poco
168
169 while (std::getline(file, linha)) {
170     // Ignorar linhas vazias ou comentarios
171     if (linha.empty() || linha[0] == '#') {
172         if (linha.find("Fluidos") != std::string::npos) {
173             lendoFluidos = true; // Mudar para leitura de
174             fluidos
175         }
176     }
177
178     if (!lendoFluidos) {
179         // Ler os dados do poco
180         std::istringstream iss(linha);
181         double profundidade, pressaoSuperficie, diametro, OD,
182             ID, vazao;
183
184         if (iss >> profundidade >> pressaoSuperficie >>
185             diametro >> OD >> ID >> vazao) {
186             poco = std::make_unique<CPoco>(profundidade,
187                 pressaoSuperficie, diametro, OD, ID, vazao);
188             std::cout << "Poco configurado a partir do arquivo "
189             << arquivo << " com sucesso!" << std::endl;
190         } else {
191             std::cerr << "Erro ao ler dados do poco na linha: "
192             << linha << std::endl;
193         }
194     } else {
195         // Ler os dados dos fluidos
196         std::istringstream iss(linha);
197         std::string nome;
198         double densidade, viscosidade, profInicial, profFinal;
199
200         if (iss >> nome >> densidade >> viscosidade >>
201             profInicial >> profFinal) {
202             auto fluido = std::make_unique<CFluido>(nome,
```

```
    densidade, viscosidade);  
197    auto trechoPoco = std::make_unique<CTrechoPoco>(  
198        profInicial, profFinal, std::move(fluido));  
199    if (!poco->AdicionarTrechoPoco(std::move(trechoPoco))){  
200        std::cerr << "Erro ao adicionar trecho ao poco!"  
201        << std::endl;  
202    } else {  
203        std::cerr << "Erro ao ler linha de fluido:" <<  
204        linha << std::endl;  
205    }  
206}  
207    file.close();  
208}  
209 // Função para o menu principal  
210 void CSimuladorPoco::MenuPrincipal() {  
211  
212     auxiliar->cabecalho();  
213  
214     while (true) {  
215         int escolha;  
216  
217         auxiliar->limparTela();  
218         auxiliar->desenharBorda();  
219         auxiliar->desenharLinhaTexto("■■■Menu■■■Principal■■■");  
220         auxiliar->desenharBorda();  
221  
222         bool pocoConfigurado = (poco != nullptr);  
223  
224         std::cout << "\n1.■Configurar■Poco\n"  
225             << (pocoConfigurado ? "2.■Exibir■Propriedades■do■  
226                 Poco\n"  
227                     : "2.[X]■Exibir■Propriedades  
228                         ■do■Poco\n")  
229             << (pocoConfigurado ? "3.■Calcular■Pressão■  
230                 Hidrostática■do■Poco\n"  
231                     : "3.[X]■Calcular■Pressão■  
232                         Hidrostática■do■Poco\n")  
233             << (pocoConfigurado ? "4.■Plotar■Perfil■  
234                 Hidrostática■do■Poco\n"
```

```

    Profundidade vs Densidade\n"
230                           : "4.[X] Plotar Perfil\n"
                                Profundidade vs Densidade\
                                n")
231 << (pocoConfigurado ? "5.[X] Calcular Perda de Carga
                                por Friccao\n"
232                           : "5.[X] Calcular Perda de
                                Carga por Friccao\n")
233 << "0.[X] Sair\n\n";
234
235 if (!pocoConfigurado)
236     std::cout << "Nenhum p\u00f3co configurado. Por favor, u
                                configure um p\u00f3co antes de prosseguir.\n";
237
238 std::cout << "Escolha: ";
239 std::cin >> escolha;
240
241 switch (escolha) {
242     case 1:
243         MenuConfigurarSimulador();
244         break;
245
246     case 0:
247         std::cout << "Saindo...\n";
248         return;
249
250     default:
251         if (!pocoConfigurado) {
252             std::cout << "Op\u00e7ao invalida ou indisponivel!\n"
                                Configure o p\u00f3co primeiro.\n";
253         } else {
254             switch (escolha) {
255                 case 2:
256                     ExibirPropriedades();
257                     break;
258
259                 case 3:
260                     MenuPressaoHidrostatica();
261                     break;
262
263                 case 4:
264                     poco->PlotarProfundidadePorDensidade();
```

```
265                                     break;
266
267         case 5:
268             MenuPerdaDeCarga();
269             break;
270
271         default:
272             std::cout << "Opcao invalida! Tente novamente.\n";
273             break;
274         }
275     }
276 }
277 std::cout << "\nPressione Enter para continuar...";
278 std::cin.ignore().get();
279 }
280 }
281
282 void CSimuladorPoco::MenuConfigurarSimulador() {
283     int escolha;
284
285     while (true) {
286         auxiliar->limparTela();
287         auxiliar->desenharBorda();
288         auxiliar->desenharLinhaTexto("Configurar Poco");
289         auxiliar->desenharBorda();
290
291         std::cout << "\n1. Criar Poco\n"
292                     "2. Adicionar Fluido\n"
293                     "3. Carregar Dados a Partir de Arquivo (.dat)\n"
294                     "0. Voltar\n";
295         std::cout << "Escolha: ";
296         std::cin >> escolha;
297
298         auto impressao = std::make_unique<CImpressao>();
299         switch (escolha) {
300             case 1: {
301                 ConfigurarPoco();
302                 break;
303             }
304         }
305     }
306 }
```

```

305         case 2: {
306             auxiliar->desenharAviso("Voce\u201d pode\u201d adicionar\u201d
307                         multiplos\u201d fluidos");
308             ConfigurarFluido();
309             break;
310
311         case 3:
312             ConfigurarPorArquivo("ArquivoPoco.dat");
313             break;
314         }
315         case 0:
316             return; // Volta ao menu principal
317         default:
318             std::cout << "Opcao\u201d invalida!\u201d Tente\u201d novamente.\n";
319             break;
320
321         std::cout << "\nPressione\u201d Enter\u201d para\u201d continuar...";
322         std::cin.ignore().get(); // Pausa ate pressionar Enter
323     }
324 }
325
326 void CSimuladorPoco::MenuPressaoHidrostatica() {
327     int escolha;
328     double profundidade;
329
330     while (true) {
331         auxiliar->limparTela();
332         auxiliar->desenharBorda();
333         auxiliar->desenharLinhaTexto("Menu\u201d de\u201d Pressao\u201d Hidrostatica"
334                                         );
335         auxiliar->desenharBorda();
336
337         std::string texto;
338
339         std::cout << "1.\u201d Calcular\u201d Pressao\u201d Hidroestatica\u201d(Fundo\u201d de
340                         \u201d p\u00f3co)\n"
341                         "2.\u201d Calcular\u201d Pressao\u201d Hidroestatica\u201d em\u201d um\u201d Ponto
342                             \u201d do\u201d P\u00f3co\n"
343                         "0.\u201d Voltar\n";
344         std::cout << "Escolha:\u201d";
345         std::cin >> escolha;

```

```
343
344     switch (escolha) {
345
346         case 1:
347             std::cout << "\nPressao Hidrostatica Total:" <<
348                 poco->PressaoHidroestaticaTotal() << " psi\n";
349
350             texto = "O Valor da Pressao Hidrostatica:";
351             impressao ->ArmazenarValorSeNecessario(texto, poco->
352                 PressaoHidroestaticaTotal(), "psi");
353
354         break;
355
356         case 2:
357
358             std::cout << "\nProfundidade maxima" << poco->
359                 ProfundidadeTotal() << " ft\n";
360             std::cout << "Informe a profundidade que deseja "
361                 " calcular [ft]:";
362             std::cin >> profundidade;
363
364             std::cout << "\nPressao Hidrostatica:" << poco->
365                 PressaoHidroestaticaNoPonto(profundidade) << " "
366                 "psi\n";
367
368         break;
369
370         case 0:
371             return; // Volta ao menu principal
372
373     default:
374         std::cout << "Opcao invalida! Tente novamente.\n";
375     }
376
377     std::cout << "\nEnter para continuar... ";
378     std::cin.ignore().get(); // Pausa ate pressionar Enter
379 }
380
381 void CSimuladorPoco::MenuPerdaDeCarga() {
382     int escolha;
383
384     while (true) {
```

```
379     auxiliar->limparTela();
380     auxiliar->desenharBorda();
381     auxiliar->desenharLinhaTexto("Menu de Perda de Carga");
382     auxiliar->desenharBorda();
383
384     std::cout << "\n1. Modelo Newtoniano\n"
385                 "2. Modelo Bingham\n"
386                 "3. Modelo de Potencia\n"
387                 "0. Voltar\n";
388     std::cout << "Escolha: ";
389     std::cin >> escolha;
390
391     switch (escolha) {
392         case 1:
393             MenuModeloNewtoniano();
394             break;
395
396         case 2:
397             MenuModeloBingham();
398             break;
399
400         case 3:
401             MenuModeloPotencia();
402             break;
403
404         case 0:
405             return; // Volta ao menu principal
406
407         default:
408             std::cout << "Opcao invalida! Tente novamente.\n";
409             break;
410     }
411
412     std::cout << "\nPressione Enter para continuar...";
413     std::cin.ignore().get(); // Pausa ate pressionar Enter
414 }
415
416 void CSimuladorPoco::MenuModeloNewtoniano() {
417     int escolha;
418     std::string armazena;
419
420     while (true) {
421         auxiliar->limparTela();
422         auxiliar->desenharBorda();
```

```

421     auxiliar->desenharLinhaTexto("Menu de Perda de Carga -"
422                                     "Newtoniano");
423     auxiliar->desenharBorda();
424
425     std::string texto;
426
427     std::cout << "\n1. Determinar Perda de Carga no Poco\n"
428                     "2. Determinar Perda de Carga no Anular\n"
429                     "0. Voltar\n";
430     std::cout << "Escolha: ";
431     std::cin >> escolha;
432
433     modeloNewtoniano = std::make_unique<CModeloNewtoniano>(poco
434 .get());
435
436     switch (escolha) {
437         case 1:
438             std::cout << "\nVelocidade no Poco: " <<
439                 modeloNewtoniano->VMediaPoco() << " ft/s"
440                     << "\nReynolds no Poco: " <<
441                     modeloNewtoniano->ReynoldsPoco()
442                     << "\nTipo de Fluxo no Poco: " <<
443                     modeloNewtoniano->DeterminarFluxoPoco
444                     ()
445                     << "\nPerda Friccional no Poco: " <<
446                     modeloNewtoniano->
447                     CalcularPerdaPorFriccaoPoco() << " psi
448                     /ft\n";
449
450         if (modeloNewtoniano->FluxoPoco() == "Turbulento")
451         {
452             std::cout << "\nFator de Friccao no Poco: " <<
453                 modeloNewtoniano->FatorFriccaoPoco() << "\n"
454                 ;
455         }
456
457         texto = "0 Valor da Perda de Friccao no Poco para o"
458               " Modelo Newtoniano: ";
459         impressao->ArmazenarValorSeNecessario(texto, poco->
460             PressaoHidroestaticaTotal(), "psi/ft");
461
462         break;

```

```

449         case 2:
450             std::cout << "\nVelocidade no anular: " <<
451                 modeloNewtoniano->VMediaAnular() << " ft/s"
452                         << "\nReynolds no anular: " <<
453                             modeloNewtoniano->ReynoldsAnular()
454                             << "\nTipo de Fluxo no anular: " <<
455                                 modeloNewtoniano->
456                                     DeterminarFluxoAnular()
457                                     << "\nPerda Friccional no Anular: " <<
458                                         modeloNewtoniano->
459                                         CalcularPerdaPorFriccaoAnular() << " psi/ft\n";
460
461
462             if (modeloNewtoniano->FluxoAnular() == "Turbulento")
463                 {
464                     std::cout << "\nFator de Friccao no Anular: "
465                         << modeloNewtoniano->FatorFriccaoAnular() <<
466                             "\n";
467                 }
468
469             texto = "O Valor da Perda de Friccao no Anular para
470                 o Modelo Newtoniano: ";
471             impressao->ArmazenarValorSeNecessario(texto, poco->
472                 PressaoHidroestaticaTotal(), "psi/ft");
473
474             break;
475
476         case 0:
477             return; // Volta ao menu principal
478
479         default:
480             std::cout << "Opcao invalida! Tente novamente.\n";
481             break;
482
483     }
484
485     std::cout << "\nPressione Enter para continuar... ";
486     std::cin.ignore().get(); // Pausa ate pressionar Enter
487 }
488
489 }
490
491 void CSimuladorPoco::MenuModeloBingham() {
492     int escolha;
493     std::string armazena;
494
495 }
```

```
479     while (true) {
480         double pontoDeEscoamento, viscosidadePlastica;
481
482         auxiliar->limparTela();
483         auxiliar->desenharBorda();
484         auxiliar->desenharLinhaTexto("Menu de Perda de Carga -"
485                                         "Bingham");
486         auxiliar->desenharBorda();
487
488         std::string texto;
489
490         std::cout << "\n1. Determinar Perda de Carga no Poco\n"
491                         "2. Determinar Perda de Carga no Anular\n"
492                         "0. Voltar\n";
493         std::cout << "Escolha: ";
494         std::cin >> escolha;
495
496         modeloBingham = std::make_unique<CModeloBingham>(poco.get());
497
498         switch (escolha) {
499             case 1:
500                 std::cout << "\nInforme o valor do"
501                               "pontoDeEscoamento [lbf/100 sq.ft]: ";
502                 std::cin >> pontoDeEscoamento;
503                 modeloBingham->PontodeEscoamento(pontoDeEscoamento)
504                               ;
505
506                 std::cout << "Informe o valor da viscosidade"
507                               "Plastica [cP]: ";
508                 std::cin >> viscosidadePlastica;
509                 modeloBingham->ViscosidadePlastica(
510                               viscosidadePlastica);
511
512                 std::cout << "\nVelocidade no Poco: " <<
513                               modeloBingham->VMediaPoco() << " ft/s"
514                               << "\nReynolds no Poco: " <<
515                               modeloBingham->ReynoldsPoco()
516                               << "\nReynolds Hedstrom no Poco: " <<
517                               modeloBingham->ReynoldsHedstromPoco()
518                               << "\nReynolds Critico no Poco: " <<
519                               modeloBingham->ReynoldsCriticoPoco()
```

```

511             << "\nTipo de Fluxo no Poco: " <<
512                 modeloBingham->DeterminarFluxoPoco()
513
514             << "\nPerda Friccional no Poco: " <<
515                 modeloBingham->
516                     CalcularPerdaPorFriccaoPoco() << " psi
517                     /ft\n";
518
519             if (modeloBingham->FluxoPoco() == "Turbulento") {
520                 std::cout << "\nFator de Friccao: " <<
521                     modeloBingham->FatorFriccaoPoco() << "\n";
522             }
523
524             texto = "O Valor da Perda de Friccao no Poco para o
525             Modelo Bingham: ";
526             impressao->ArmazenarValorSeNecessario(texto, poco->
527                 PressaoHidroestaticaTotal(), "psi/ft");
528
529             break;
530
531         case 2:
532             std::cout << "\nInforme o valor do
533                 pontoDeEscoamento [lbf/100 sq.ft]: ";
534             std::cin >> pontoDeEscoamento;
535             modeloBingham->PontoDeEscoamento(pontoDeEscoamento)
536                 ;
537
538             std::cout << "Informe o valor da viscosidade
539                 Plastica [cP]: ";
540             std::cin >> viscosidadePlastica;
541             modeloBingham->ViscosidadePlastica(
542                 viscosidadePlastica);
543
544             std::cout << "\nVelocidade no anular: " <<
545                 modeloBingham->VMediaAnular() << " ft/s"
546                 << "\nReynolds no anular: " <<
547                     modeloBingham->ReynoldsAnular()
548                     << "\nReynolds de Hedstrom no anular: "
549                     << modeloBingham->
550                         ReynoldsHedstromAnular()
551                         << "\nReynolds Critico no anular: " <<
552                             modeloBingham->ReynoldsCriticoAnular()
553                             << "\nTipo de Fluxo no anular: " <<
554                                 modeloBingham->DeterminarFluxoAnular()

```

```

536                                     << "\nPerda\u00e1Frictional\u00e1no\u00e1Anular:\u00d7" <<
537                                     modeloBingham->
538                                     CalcularPerdaPorFriccaoAnular() << " \u00d7
539                                     psi/ft\n";
540
541
542             if (modeloBingham->FluxoAnular() == "Turbulento") {
543                 std::cout << "\nFator\u00e1de\u00e1Friccao:\u00d7" <<
544                 modeloBingham->FatorFriccaoAnular() << "\n";
545             }
546
547             texto = "O\u00d7Valor\u00e1da\u00e1Perda\u00e1de\u00e1Friccao\u00e1no\u00e1Anular\u00e1para
548             \u00e1Modelo\u00e1Bingham:\u00d7";
549             impressao->ArmazenarValorSeNecessario(texto, poco->
550                                         PressaoHidroestaticaTotal(), " \u00d7psi/ft");
551
552             break;
553         case 0:
554             return; // Volta ao menu principal
555         default:
556             std::cout << "Op\u00e7ao\u00e1invalida!\u00d7Tente\u00e1novamente.\n";
557             break;
558     }
559
560     std::cout << "\nPressione\u00e1Enter\u00e1para\u00e1continuar... ";
561     std::cin.ignore().get(); // Pausa ate pressionar Enter
562 }
563
564
565 void CSimuladorPoco::MenuModeloPotencia() {
566     int escolha;
567     std::string armazena;
568
569     while (true) {
570         double indiceDeConsistencia;
571
572         auxiliar->limparTela();
573         auxiliar->desenharBorda();
574         auxiliar->desenharLinhaTexto("Menu\u00e1de\u00e1Perda\u00e1de\u00e1Carga\u00e1-\u00e1
575                                     Potencia");
576         auxiliar->desenharBorda();
577
578         std::string texto;

```

```

571
572     std::cout << "\n1. Determinar Perda de Carga no Poco\n"
573                 "2. Determinar Perda de Carga no Anular\n"
574                 "0. Voltar\n";
575     std::cout << "Escolha:" ;
576     std::cin >> escolha;
577
578     modeloPotencia = std::make_unique<CModeloPotencia>(poco.get()
579                                         ());
580
581     switch (escolha) {
582         case 1:
583             std::cout << "\nInforme o valor do indice de
584                           consistencia [Cp_eq]: ";
585             std::cin >> indiceDeConsistencia;
586             modeloPotencia->IndiceDeConsistencia(
587                         indiceDeConsistencia);
588
589             std::cout << "\nVelocidade no Poco: " <<
590                         modeloPotencia->VMediaPoco() << " ft/s"
591                         << "\nReynolds no Poco: " <<
592                         modeloPotencia->ReynoldsPoco()
593                         << "\nReynolds Critico no Poco: " <<
594                         modeloPotencia->ReynoldsCriticoPoco()
595                         << "\nTipo de Fluxo no Poco: " <<
596                         modeloPotencia->DeterminarFluxoPoco()
597                         << "\nPerda Friccional no Poco: " <<
598                         modeloPotencia->
599                         CalcularPerdaPorFriccaoPoco() << " psi
/ft\n";
600
601         if (modeloPotencia->FluxoPoco() == "Turbulento") {
602             std::cout << "\nFator de Friccao: "
603                         << modeloPotencia->FatorFriccaoPoco() << "\n";
604         }
605
606         texto = "O Valor da Perda de Friccao no Poco para o
607                           Poco: ";
608         impressao->ArmazenarValorSeNecessario(texto, poco->
609                         PressaoHidroestaticaTotal(), "psi/ft");
610
611         break;

```

```

600         case 2:
601             std::cout << "\nInforme o valor do indice de
602             consistencia [Cp_eq]: ";
603             std::cin >> indiceDeConsistencia;
604             modeloPotencia->IndiceDeConsistencia(
605                 indiceDeConsistencia);
606
607             std::cout << "\nVelocidade no anular: " <<
608                 modeloPotencia->VMediaAnular() << " ft/s"
609                 << "\nReynolds no anular: " <<
610                     modeloPotencia->ReynoldsAnular()
611                     << "\nReynolds Critico no anular: " <<
612                         modeloPotencia->ReynoldsCriticoAnular
613                         ()
614
615             << "\nTipo de Fluxo no anular: " <<
616                 modeloPotencia->DeterminarFluxoAnular
617                 ()
618             << "\nPerda Friccional no Anular: " <<
619                 modeloPotencia->
620                     CalcularPerdaPorFriccaoAnular() << " psi /ft\n";
621
622             if (modeloPotencia->FluxoAnular() == "Turbulento")
623             {
624                 std::cout << "\nFator de Friccao: " <<
625                     modeloPotencia->FatorFriccaoAnular() << "\n"
626                     ;
627             }
628
629             texto = "O Valor da Perda de Friccao no Anular para
630             o Modelo Potencia: ";
631             impressao->ArmazenarValorSeNecessario(texto, poco->
632                 PressaoHidroestaticaTotal(), " psi /ft");
633
634             break;
635
636         case 0:
637             return; // Volta ao menu principal
638
639         default:
640             std::cout << "Opcao invalida! Tente novamente.\n";
641             break;
642
643     }
644
645

```

```

626     std::cout << "\nPressione Enter para continuar..." ;
627     std::cin.ignore().get(); // Pausa ate pressionar Enter
628 }
629 }
```

---

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; }
```

```

31     double DiametroPoco() const { return diametroPoco; }
32     double DiametroRevestimentoOD() const { return
33         diametroRevestimentoOD; }
34     double DiametroRevestimentoID() const { return
35         diametroRevestimentoID; }
36     double Vazao() const { return vazao; }
37     std::vector<CTrechoPoco*> Trechos() const;
38
39 // Setters
40     void ProfundidadeTotal( double Profund ) { profundidadeFinal =
41         Profund; }
42     void ProfundidadeOcupada( double Profund ) {
43         profundidadeOcupada = Profund; }
44     void PressaoSuperficie( double PressaoSup ) { pressaoSuperficie
45         = PressaoSup; }
46     void DiametroPoco( double D ) { diametroPoco = D; }
47     void DiametroRevestimentoOD( double OD ) {
48         diametroRevestimentoOD = OD; }
49     void DiametroRevestimentoID( double ID ) {
50         diametroRevestimentoID = ID; }
51     void Vazao( double q ) { vazao = q; }
52
53 // Metodos
54     bool AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco>
55         TrechoPoco);
56     double PressaoHidroestaticaTotal() const;
57     double PressaoHidroestaticaNoPonto(double profundidade) const;
58     double DensidadeEfetivaTotal() const;
59     double ViscosidadeEfetivaTotal() const;
60     bool VerificarPreenchimentoColuna();
61     void PlotarProfundidadePorDensidade();
62
63 };
64
65 #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 "CGnuplot.h"
3 #include <iostream>

```

```
4 #include <vector>
5 #include <fstream>
6 #include <cstdlib>
7
8 // Metodos
9
10 std::vector<CTrechoPoco*> CPoco::Trechos() const {
11     std::vector<CTrechoPoco*> trechosPonteiros;
12     for (const auto& trecho : trechos) {
13         trechosPonteiros.push_back(trecho.get()); // Adiciona o
14             ponteiro do trecho ao vetor
15     }
16     return trechosPonteiros; // Retorna o vetor de ponteiros
17 }
18 bool CPoco::AdicionarTrechoPoco(std::unique_ptr<CTrechoPoco>
19     TrechoPoco) {
20     double ProfundidadeFluido = TrechoPoco->ProfundidadeFinal() -
21         TrechoPoco->ProfundidadeInicial();
22
23     // Verifica se a profundidade total ocupada + profundidade do
24     // novo fluido excede a profundidade total do pôco
25     if (profundidadeOcupada + ProfundidadeFluido <=
26         profundidadeFinal) {
27         trechos.push_back(std::move(TrechoPoco));
28         profundidadeOcupada += ProfundidadeFluido;
29         return true;
30     } else {
31         std::cout << "Erro: o fluido excede a profundidade total do
32             pôco!\n";
33         return false;
34     }
35 }
36 double CPoco::PressaoHidroestaticaTotal() const {
37     double pressaoTotal = 0.0;
38
39     for (const auto& Trecho : trechos) {
40         pressaoTotal += Trecho->PressaoHidroestatica();
41     }
42     return pressaoTotal + pressaoSuperficie;
```

```
40 }
41
42 double CPoco::PressaoHidroestaticaNoPonto(double profundidade)
43     const {
44         double pressaoTotal = pressaoSuperficie;
45         double profundidadeAcumulada = 0.0;
46
47         for (const auto& Trecho : trechos) {
48             double profundidadeTrecho = Trecho->ProfundidadeFinal() -
49                             Trecho->ProfundidadeInicial();
50
51             // Verifica se a profundidade esta dentro do trecho atual
52             if (profundidade <= profundidadeAcumulada +
53                 profundidadeTrecho) {
54                 // Calcula a contribuicao do trecho ate a profundidade
55                 // desejada
56                 pressaoTotal += Trecho->PressaoHidroestatica(
57                     profundidade - profundidadeAcumulada);
58                 break;
59             } else {
60                 // Adiciona a pressao hidrostatica do trecho completo
61                 pressaoTotal += Trecho->PressaoHidroestatica();
62                 profundidadeAcumulada += profundidadeTrecho;
63             }
64
65         }
66
67         return pressaoTotal;
68     }
69
70     bool CPoco::VerificarPreenchimentoColuna() {
71         double ProfundidadeNaoOcupada = profundidadeFinal -
72             profundidadeOcupada;
73
74         if (ProfundidadeNaoOcupada > 0) {
75             std::cout << "Uma coluna de " << ProfundidadeNaoOcupada <<
76                 " litros fluido precisa ser adicionada!\n";
77             std::cout << std::endl;
78             return false; // Coluna nao preenchida
79         } else {
80             std::cout << "A coluna de fluidos equivale a profundidade da coluna do poco!\n";
81             std::cout << std::endl;
82         }
83     }
84 }
```

```
74         return true; // coluna preenchida
75     }
76 }
77
78 double CPoco::DensidadeEfetivaTotal() const {
79     double densidadeTotal = 0.0;
80     double comprimentoTotal = 0.0;
81     double comprimentoTrecho = 0.0;
82
83     for (const auto& Trecho : trechos) {
84         comprimentoTrecho = Trecho->ProfundidadeFinal() - Trecho->
85             ProfundidadeInicial();
86         densidadeTotal += Trecho->DensidadeEquivalente() *
87             comprimentoTrecho;
88         comprimentoTotal += comprimentoTrecho;
89     }
89
90     return densidadeTotal / comprimentoTotal;
91 }
91
92 double CPoco::ViscosidadeEfetivaTotal() const {
93
94     double viscosidadeTotal = 0.0;
95
96     for (const auto& Trecho : trechos) {
97         viscosidadeTotal += Trecho->Fluido()->Viscosidade();
98     }
99     return viscosidadeTotal / trechos.size();
100 }
101
102 void CPoco::PlotarProfundidadePorDensidade() {
103     std::vector<double> Profundidade;
104     std::vector<double> Densidade;
105
106     double profundidadeTotal = 0;
107
108     // Coletar dados para a profundidade e densidade
109     for (const auto& trecho : trechos) {
110
111         double Intervalo = trecho->ProfundidadeFinal() - trecho->
112             ProfundidadeInicial();
```

```
113     for (double i = 0; i <= Intervalo; i += 1) { // Usando um
114         incremento menor
115         double ProfundidadeAtual = profundidadeTotal + i; // Atualiza a profundidade em cada iteracao
116         double Dens = PressaoHidroestaticaNoPonto(
117             ProfundidadeAtual) / (ProfundidadeAtual * 0.05195);
118         Densidade.push_back(Dens);
119         Profundidade.push_back(ProfundidadeAtual); // Armazena
120             a profundidade atual
121     }
122
123     // Escrever dados em arquivo
124     std::ofstream outputFile("dadosSimulacaoPoco_Gnuplot.dat");
125
126     for (size_t j = 0; j < Profundidade.size(); ++j) {
127         outputFile << Profundidade[j] << "\t" << Densidade[j] <<
128             std::endl;
129     }
130     outputFile.close();
131
132     // Comando Gnuplot para plotar os dados
133     std::ofstream gnuplotFile("plot_script.gp");
134     if (!gnuplotFile.is_open()) {
135         std::cerr << "Erro ao abrir ou arquivo plot_script.gp para "
136             "escrita." << std::endl;
137         return;
138     }
139     gnuplotFile << "set title 'Profundidade vs Densidade'\n";
140     gnuplotFile << "set xlabel 'Densidade, lbm/gal'\n"; //
141         Corrigido o label
142     gnuplotFile << "set ylabel 'Profundidade, ft'\n"; // Corrigido
143         o label
144     gnuplotFile << "set xrange [20:0]\n"; // Inverter o eixo Y
145     gnuplotFile << "set grid\n"; // Adicionar grade ao grafico
146     gnuplotFile << "set style data linespoints\n"; // Estilo de
147         linha com pontos
```

```

145     // Plota apenas uma curva
146     gnuplotFile << "plot 'dados.txt' using 2:1 with linespoints"
147         title 'Densidade vs Profundidade '\n";
148     gnuplotFile << "set terminal pngcairo size 1920,1080\n";
149     gnuplotFile << "set output 'Profundidade_vs_densidade.png'\n";
150     gnuplotFile << "pause -1\n"; // Pausa para que voce possa ver o
151         grafico
152     gnuplotFile.close();
153
154 }

---



```

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 CModeloReológico.

Listing 6.8: Arquivo de implementação da classe CModeloReológico

```

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     // Métodos
38     double DeterminarFatorFriccao(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     double VMedioPoco, double diametroRevestimentoID, double
9     viscosidade) {
10     reynoldsPoco = (928 * densidade * VMedioPoco *
11         diametroRevestimentoID) / viscosidade;
12 }
```

```
9     return reynoldsPoco;
10 }
11
12 double 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
20
21
22 double CModeloReologico::DeterminarVelocidadeMediaPoco(double vazao,
23     double diametroRevestimentoID){
24     vMediaPoco = vazao / (2.448 * std::pow(diametroRevestimentoID,
25                         2));
26     return vMediaPoco;
27 }
28
29
30
31
32
33
34
35
36
37
38
39
40
41
```

```

        ;
42    };
43
44    // Metodo 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
59    // Estimar o chute inicial usando o fator laminar
60    double x0 = laminar_fator(re);
61
62    // Resolver a equacao de Fanning usando Newton-Raphson
63    return newtonRaphson(x0);
64}

```

---

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 "CModeloReologico.h"
5
6
7 class CModeloNewtoniano : public CModeloReologico {
8
9
10 public:
11     //Construtor
12     CModeloNewtoniano() {}
13     ~CModeloNewtoniano() {}
14     CModeloNewtoniano(CPoco* poco) : CModeloReologico(poco){}

```

```

15
16     //Metodos
17     std::string DeterminarFluxoPoco() override;
18     std::string DeterminarFluxoAnular() override;
19     double CalcularPerdaPorFriccaoPoco() override;
20     double CalcularPerdaPorFriccaoAnular() 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 // Funcao para determinar o tipo de fluxo no poco
6 std::string CModeloNewtoniano::DeterminarFluxoPoco() {
7
8     DeterminarVelocidadeMediaPoco(poco->Vazao(), poco->
9         DiametroRevestimentoID());
10    DeterminarReynoldsPoco(poco->DensidadeEfetivaTotal(),
11        vMediaPoco, poco->DiametroRevestimentoID(), poco->
12        ViscosidadeEfetivaTotal());
13    fluxoPoco = (reynoldsPoco <= 2100) ? "Laminar" : "Turbulento";
14        // Determinacao do fluxo
15
16    return fluxoPoco;
17 }
18
19 // Funcao para determinar o tipo de fluxo no espaco anular
20 std::string CModeloNewtoniano::DeterminarFluxoAnular() {
21
22     double diametroAnular = poco->DiametroPoco() - poco->
23         DiametroRevestimentoOD();
24
25     DeterminarVelocidadeMediaAnular(poco->Vazao(), poco->
26         DiametroPoco(), poco->DiametroRevestimentoOD());
27     DeterminarReynoldsAnular(poco->DensidadeEfetivaTotal(),
28         vMediaAnular, diametroAnular, poco->ViscosidadeEfetivaTotal)

```

```
(());
22     this->fluxoAnular = (reynoldsAnular <= 2100) ? "Laminar" : "
23         Turbulento"; // Determinacao do fluxo
24
25 }
26
27 // Funcao para calcular a perda de carga por friccao no poco
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 // Funcao para calcular a perda de carga por friccao no espaco
45 // anular
46
47
48     double diametroAnular = poco->DiametroPoco() - poco->
49         DiametroRevestimentoOD();
50     this->fatorFriccaoAnular = DeterminarFatorFriccao(
51         reynoldsAnular);
52
53     if (fluxoAnular == "Laminar") {
54         return (poco->ViscosidadeEfetivaTotal() * vMediaAnular) /
55             (1000 * std::pow(diametroAnular, 2));
56     } else { // Fluxo turbulento
57         return (fatorFriccaoAnular * poco->DensidadeEfetivaTotal()
```

```

    * std::pow(vMediaAnular, 2) ) / (21.1 * diametroAnular);
55 }
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; }

33    // Setters
34    void PontoDeEscoamento( double PontoE ) { pontoDeEscoamento =

```

```

        PontoE; }

32    void ViscosidadePlastica( double ViscosidadeP ) {
33        viscosidadePlastica = ViscosidadeP; }

34    // Metodos
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    std::string DeterminarFluxoPoco() override;
44    std::string DeterminarFluxoAnular() override;
45    double CalcularPerdaPorFriccaoPoco() override;
46    double CalcularPerdaPorFriccaoAnular() override;
47
48};

49#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 Reynolds
11     // crítico
12     auto func = [hedstron](double x) {
13         return ((x / std::pow(1 - x, 3)) * 16800) - hedstron;
14     };
15
16     // Define a derivada de f(x)
17     auto derivadaFunc = [] (double x) {

```

```

17     return (16800 * (1 + x) / std::pow(1 - x, 4));
18 }
19
20 // Metodo 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             convergencia
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 critico usando o valor
34 // encontrado
35
36 double CModeloBingham::DeterminarReynoldsHedstronPoco(double
37     densidade, double pontoDeEscoamento, double
38     diametroRevestimentoID, double viscosidade) {
39     reynoldsCriticoPoco = DeterminarReynoldsCritico(
40         reynoldsHedstronPoco);
41     return (37100 * densidade * pontoDeEscoamento * std::pow(
42         diametroRevestimentoID, 2))/ (std::pow(viscosidade, 2));
43 }
44
45 double CModeloBingham::DeterminarReynoldsHedstronAnular(double
46     densidade, double pontoDeEscoamento, double diametroAnular,
47     double viscosidade) {
48     reynoldsCriticoAnular = DeterminarReynoldsCritico(
49         reynoldsHedstronAnular);
50     return (24700 * densidade * pontoDeEscoamento * std::pow(
51         diametroAnular, 2))/ (std::pow(viscosidade, 2));
52 }
53
54 // Funcao para determinar o tipo de fluxo no poto
55 std::string CModeloBingham::DeterminarFluxoPoco() {
56

```

```
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"; // Determinacao do fluxo
61
62     return fluxoPoco;
63 }
64
65 // Funcao para determinar o tipo de fluxo no espaco 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"; // Determinacao do fluxo
82
83     return fluxoAnular;
84 }
85
86 // Funcao para calcular a perda de carga por friccao no poco
87 double CModeloBingham::CalcularPerdaPorFriccaoPoco() {
88
89     if (fluxoPoco.empty()) {
90         DeterminarFluxoPoco();
91     }
92
93     fatorFriccaoPoco = DeterminarFatorFriccao(reynoldsPoco);
```

```

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

---

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     //Metodos
43     double DeterminarReynoldsCritico(double Reynolds);
44     double DeterminarReynoldsPoco(double densidade, double
45         VMedioPoco, double diametroRevestimentoID, double
46         indiceDeConsistencia, double indiceDeComportamento);
47     double DeterminarReynoldsAnular(double densidade, double
```

```

    VMedioPoco, double diametroAnular, double
    indiceDeConsistencia, double indiceDeComportamento);
40     std::string DeterminarFluxoPoco() override;
41     std::string DeterminarFluxoAnular() override;
42     double CalcularPerdaPorFriccaoPoco() override;
43     double CalcularPerdaPorFriccaoAnular() override;
44
45 };
46
47 #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     }
22     return reynoldsPoco;
23 }
24
25 double CModeloPotencia::DeterminarReynoldsAnular(double densidade,
26         double VMedioPoco, double diametroAnular, double
27         indiceDeConsistencia, double indiceDeComportamento) {

```

```

20     reynoldsAnular = ((109000 * densidade * std::pow(VMedioPoco, 2-
21         indiceDeComportamento)) / indiceDeConsistencia) * (std::pow(
22             ((0.0208 * diametroAnular)/(2+(1/indiceDeComportamento)),
23                 indiceDeComportamento));
24
25
26     if (indiceDeComportamento < 0.4){
27         reynoldsCriticoAnular = DeterminarReynoldsCritico(
28             reynoldsAnular);
29     }
30
31
32     return reynoldsAnular;
33 }
34
35 // Funcao para determinar o tipo de fluxo no poco
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"; // Determinacao do fluxo
47     return fluxoPoco;
48 }
49
50 // Funcao para determinar o tipo de fluxo no espaco anular
51 std::string CModeloPotencia::DeterminarFluxoAnular() {
52
53     double diametroAnular = poco->DiametroPoco() - poco->
54         DiametroRevestimentoOD();
55     vMediaAnular = DeterminarVelocidadeMediaAnular(poco->Vazao(),
56         poco->DiametroPoco(), poco->DiametroRevestimentoOD());
57     reynoldsAnular = DeterminarReynoldsAnular(poco->
58         DensidadeEfetivaTotal(), vMediaAnular, diametroAnular,
59         indiceDeConsistencia, indiceDeComportamento);
60
61     fluxoAnular = (reynoldsAnular <= reynoldsCriticoAnular) ? "
62         Laminar" : "Turbulento"; // Determinacao do fluxo
63     return fluxoAnular;
64 }
```

```
48 }
49
50 // Funcao para calcular a perda de carga por friccao no poco
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 // Funcao para calcular a perda de carga por friccao no espaco
73 // anular
74 double CModeloPotencia::CalcularPerdaPorFriccaoAnular() {
75     if (fluxoAnular.empty()) {
76         DeterminarFluxoAnular();
77     }
78
79     double diametroAnular = poco->DiametroPoco() - poco->
80         DiametroRevestimentoOD();
81     fatorFriccaoAnular = DeterminarFatorFriccao(reynoldsAnular);
82
83     if (fluxoAnular == "Laminar") {
84         return ((indiceDeConsistencia * std::pow(vMediaAnular, -
85             indiceDeComportamento)) * std::pow(( (2+(1/
86             indiceDeComportamento) ) / 0.0208),
87             indiceDeComportamento))
88         / (144000 * std::pow(diametroAnular, (1+
89             indiceDeComportamento)));
90 }
```

```

78     } else { // Fluxo turbulento
79         return (fatorFriccaoAnular * poco->DensidadeEfetivaTotal()
80             * std::pow(vMediaAnular, 2) ) / (21.1 * diametroAnular);
81     }

```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.16 a implementação da classe CImpressao.

Listing 6.16: Arquivo de implementação da classe CImpressao

```

1 #ifndef CIMPRESSAO_H
2 #define CIMPRESSAO_H
3
4 #include <string>
5
6 class CImpressao {
7 protected:
8     double pressaoHidroestatica = 0.0;
9     double perdadeFriccaoAnularNewtoniano = 0.0;
10    double perdadeFriccaoAnularBingham = 0.0;
11    double perdadeFriccaoAnularPotencia = 0.0;
12    double perdadeFriccaoPocoNewtoniano = 0.0;
13    double perdadeFriccaoPocoBingham = 0.0;
14    double perdadeFriccaoPocoPotencia = 0.0;
15
16 public:
17     // Construtor
18     CImpressao() {}
19     ~CImpressao() {}
20
21     // Getters
22     double PressaoHidroestatica() const { return
23         pressaoHidroestatica; }
24     double PerdadeFriccaoAnularNewtoniano() const { return
25         perdadeFriccaoAnularNewtoniano; }
26     double PerdadeFriccaoAnularBingham() const { return
27         perdadeFriccaoAnularBingham; }
28     double PerdadeFriccaoAnularPotencia() const { return
29         perdadeFriccaoAnularPotencia; }
30     double PerdadeFriccaoPocoNewtoniano() const { return
31         perdadeFriccaoPocoNewtoniano; }
32     double PerdadeFriccaoPocoBingham() const { return
33         perdadeFriccaoPocoBingham; }

```

```

28     double PerdadeFriccaoPocoPotencia() const { return
29         perdadeFriccaoPocoPotencia; }
30
31     // Setters
32     void PressaoHidroestatica(double pressaoHidro) {
33         pressaoHidroestatica = pressaoHidro; }
34     void PerdadeFriccaoAnularNewtoniano(double anularNewtoniano) {
35         perdadeFriccaoAnularNewtoniano = anularNewtoniano; }
36     void PerdadeFriccaoAnularBingham(double anularBingham) {
37         perdadeFriccaoAnularBingham = anularBingham; }
38     void PerdadeFriccaoAnularPotencia(double anularPotencia) {
39         perdadeFriccaoAnularPotencia = anularPotencia; }
40     void PerdadeFriccaoPocoNewtoniano(double pocoNewtoniano) {
41         perdadeFriccaoPocoNewtoniano = pocoNewtoniano; }
42     void PerdadeFriccaoPocoBingham(double pocoBingham) {
43         perdadeFriccaoPocoBingham = pocoBingham; }
44     void PerdadeFriccaoPocoPotencia(double pocoPotencia) {
45         perdadeFriccaoPocoPotencia = pocoPotencia; }
46
47     // Metodos
48     void ArmazenarValorSeNecessario(std::string texto, double valor
49         , std::string unidade);
50     void ArmazenarValorSeNecessario(std::string texto, double valor
51         );
52
53 };
54
55 #endif

```

---

Fonte: produzido pelo autor.

Apresenta-se na listagem 6.17 a implementação da classe CImpressao.

Listing 6.17: Arquivo de implementação da classe CImpressao

```

1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <string>
5 #include <iomanip>
6
7 #include "CImpressao.h"
8
9 void CImpressao::ArmazenarValorSeNecessario(std::string texto,
10     double valor) {

```

```
10
11     ArmazenarValorSeNecessario(texto, valor, "");
12 }
13 }
14
15 void CImpressao::ArmazenarValorSeNecessario(std::string texto,
16     double valor, std::string unidade) {
17
18     char resposta;
19     std::cout << "Gostaria de armazenar o valor (s/n)? ";
20     std::cin >> resposta;
21
22     if (resposta == 's' || resposta == 'S') {
23
24         std::ofstream arquivo("ResultadosSimulacaoPoco.dat", std::
25             ios::app); // Cria/abre o arquivo para escrita
26         arquivo << texto << valor << unidade << std::endl;
27         arquivo.close(); // Fecha o arquivo
28     }
29 }
```

---

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

### 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 (lbm/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 seguintes cálculos:

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

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

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

Na Figura 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 seguintes cálculos:

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

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

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

Na Figura 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 seguintes cálculos:

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

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

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

Na Figura 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 seguintes cálculos:

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

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

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

Na Figura 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 poço

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

Para o fluxo laminar no poço realizamos os seguintes cálculos:

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

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

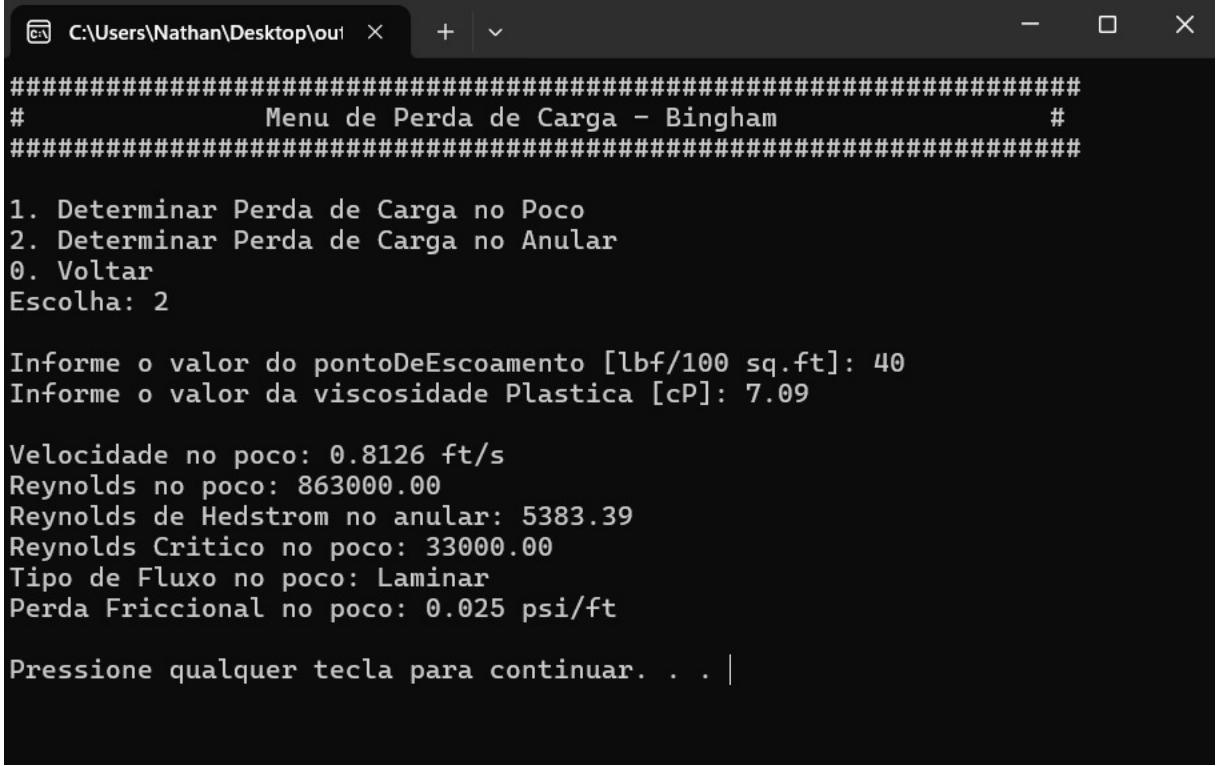
$$N_{rec} = 33000$$

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

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

Na Figura 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



```
C:\Users\Nathan\Desktop\out X - □ ×
#####
#      Menu de Perda de Carga - Bingham      #
#####
1. Determinar Perda de Carga no Poco
2. Determinar Perda de Carga no Anular
0. Voltar
Escolha: 2

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

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

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

Fonte: Produzido pelo autor.

Para o fluxo turbulento no poço realizamos os seguintes cálculos:

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

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

$$N_{rec} = 27000$$

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

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

Na Figura 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 seguintes cálculos:

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

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

$$N_{rec} = 6000$$

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

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

Na Figura 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

```
C:\Users\Nathan\Desktop\out
#####
#           Menu de Perda de Carga - Bingham      #
#####

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

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

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

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

Fonte: Produzido pelo autor.

Para o fluxo turbulento no anular realizamos os seguintes cálculos:

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

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

$$N_{rec} = 5000$$

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

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

Na Figura 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 poço

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

Para o fluxo laminar no poço realizamos os seguintes cálculos:

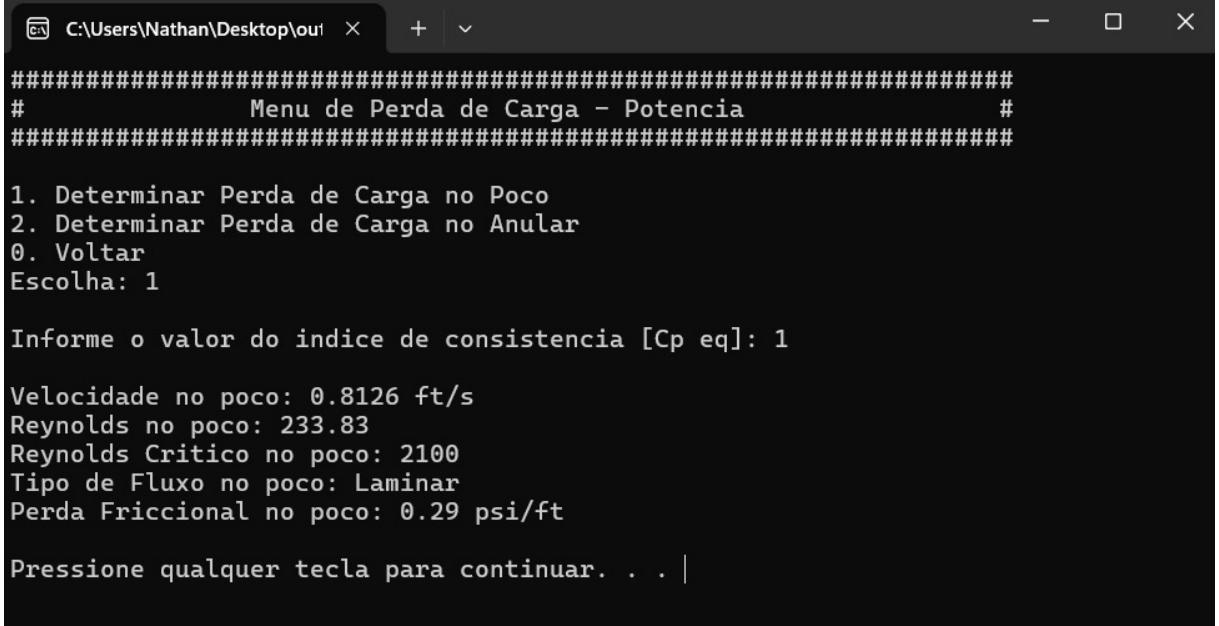
$$\bar{v} = \frac{100}{2.448 \times 7.09^2} = 0.8126$$

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

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

Na Figura 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



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

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

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

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

Fonte: Produzido pelo autor.

Para o fluxo turbulento no poço realizamos os seguintes cálculos:

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

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

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

Na Figura 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 seguintes cálculos:

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

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

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

Na Figura 7.11 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

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

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

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

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

Fonte: Produzido pelo autor.

Para o fluxo turbulento no poço realizamos os seguintes cálculos:

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

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

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

Na Figura 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.

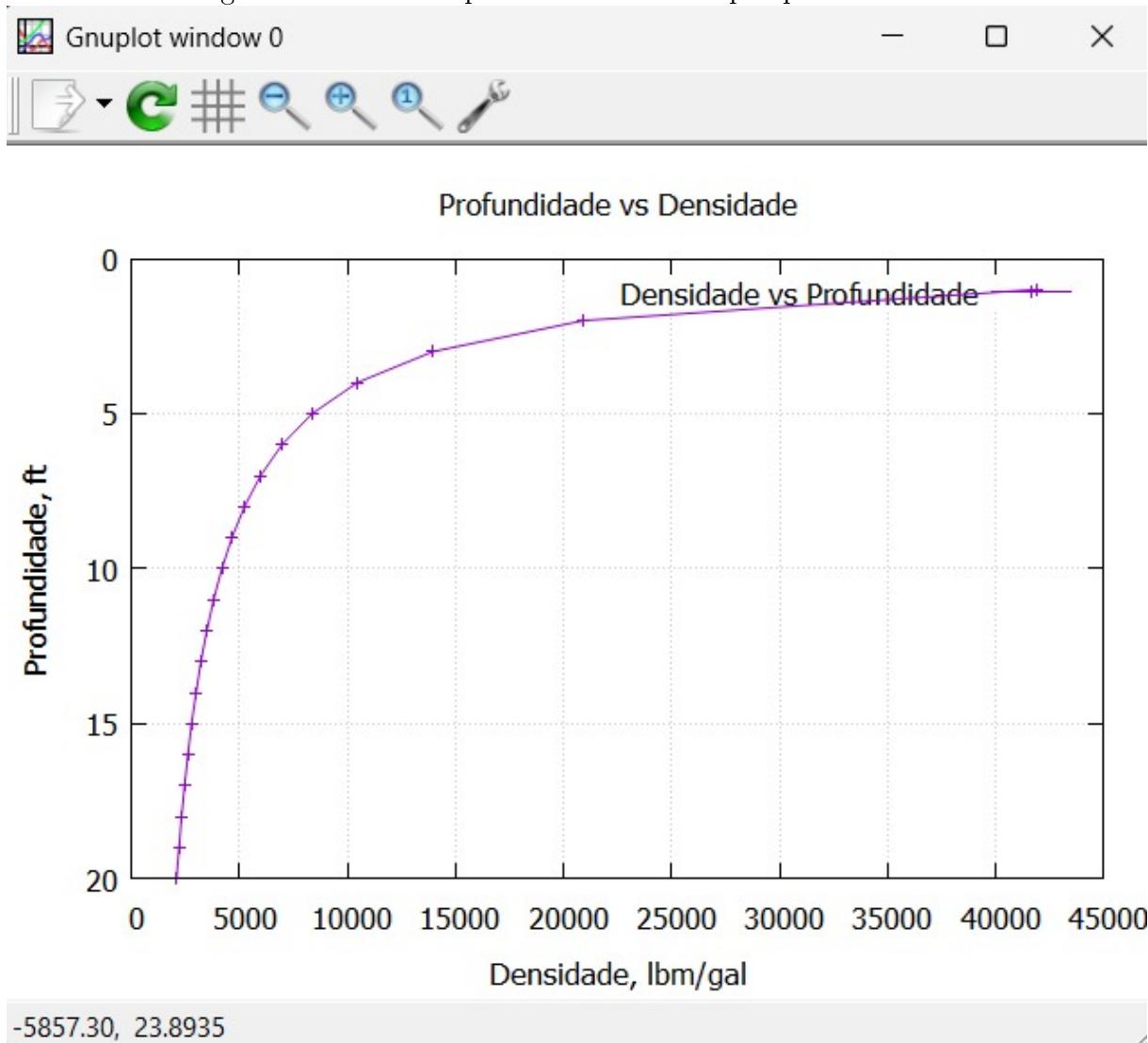
Figura 7.13: Soluções geradas pelo código para pressão hidrostática no fundo do poço

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

Fonte: Produzido pelo autor.

Na Figura 7.14 temos um gráfico com a variação da pressão hidrostática em função da profundidade.

Figura 7.14: Gráfico pressão hidrostática por profundidade

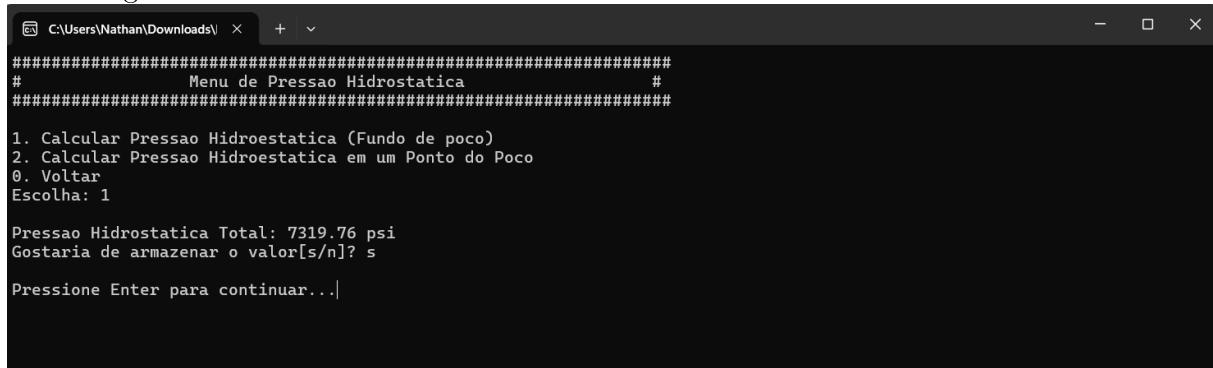


Fonte: Produzido pelo autor.

### 7.2.8 Validação da funcionalidade impressão

Nesta seção, será validada a classe CImpressao que tem a função de armazenar os resultados da simulação em um arquivo de saída .dat. Pode-se observar na Figura 7.15 a funcionalidade armazenar os resultados obtidos durante a simulação.

Figura 7.15: Terminal mostrando a funcionalidade armazenar os resultados



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

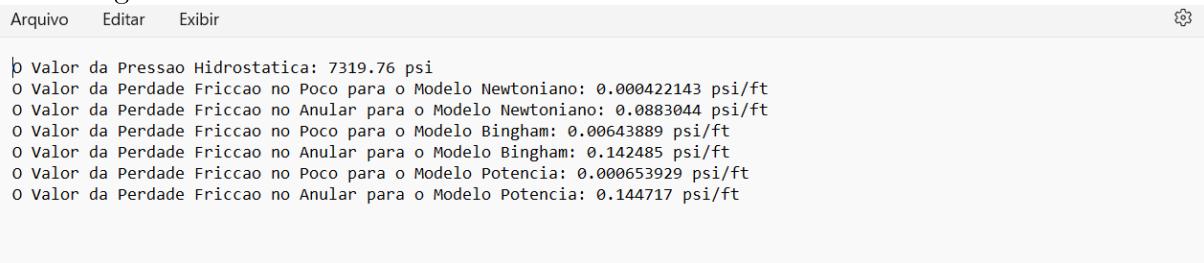
Pressao Hidrostatica Total: 7319.76 psi
Gostaria de armazenar o valor[s/n]? s

Pressione Enter para continuar...|
```

Fonte: Produzido pelo autor.

Na Figura 7.16 podemos ver todos os resultados gerados durante a simulação que forma armazenados no arquivo de saída .dat.

Figura 7.16: Terminal mostrando a funcionalidade armazenar os resultados



```
Arquivo Editar Exibir

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

Fonte: Produzido pelo autor.

# Capítulo 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 9 - 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.

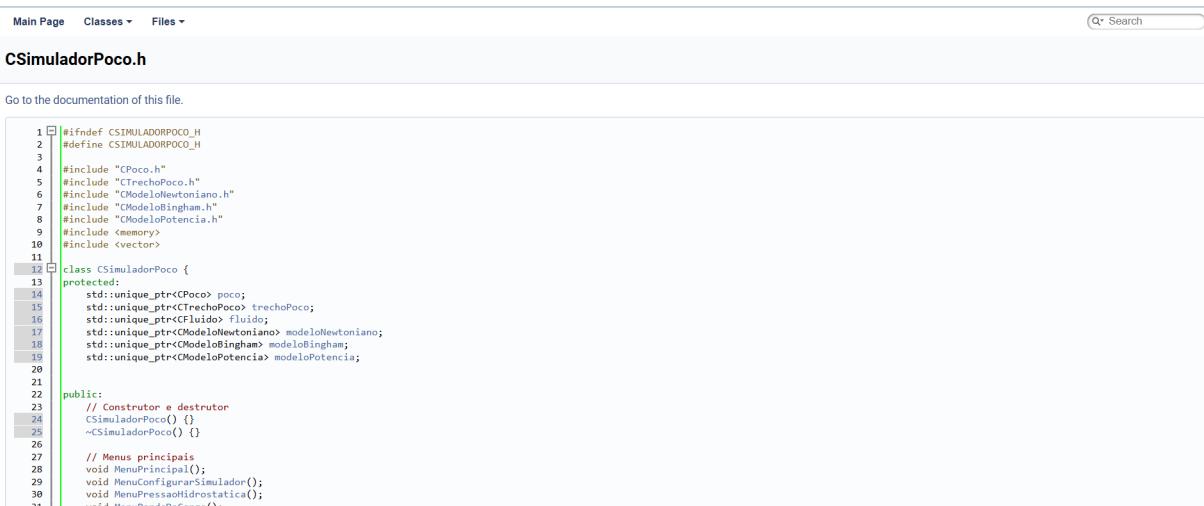
Figura 8.1: Logo e documentação do *software*  
**SoftwareEducacionalParaAnaliseESolucaoDeProblemasEmEngenhariaDePoco**



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 CSimuladorPoco, no *Doxygen*  
**SoftwareEducacionalParaAnaliseESolucaoDeProblemasEmEngenhariaDePoco**



Fonte: Produzido pelo autor.

# Capítulo 9

## Manual do usuário

### 9.1 Instalação

O software foi disponibilizado no site GitHub - Software Educacional Para Análise de Poço

Lá você encontra instruções atualizadas para baixar e instalar.

#### 9.1.1 Dependências

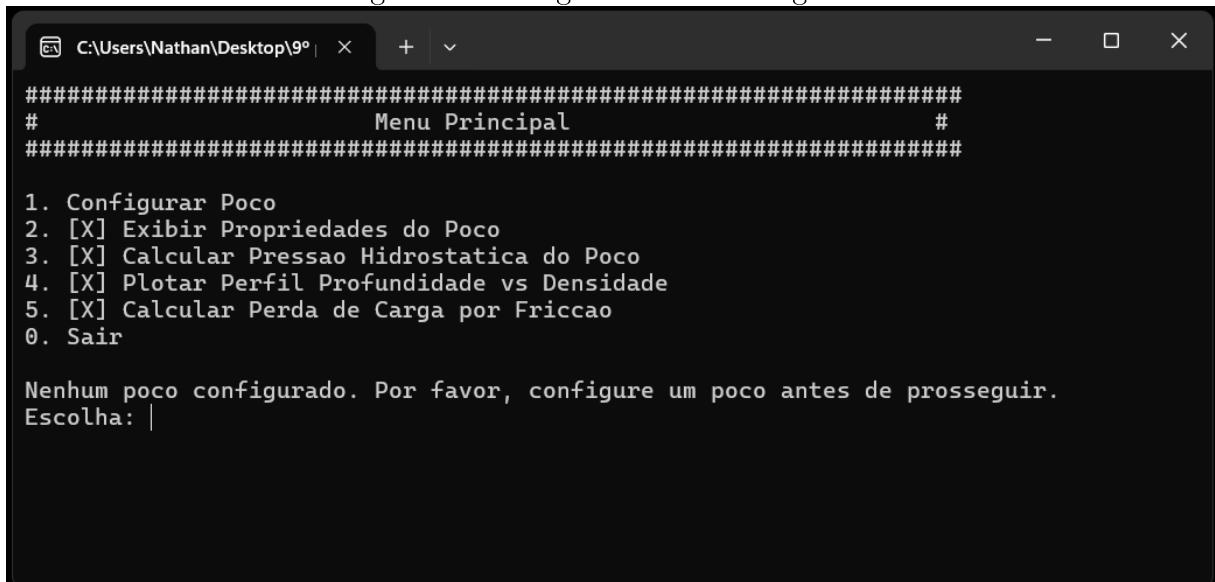
Para compilar o software é necessário atender as seguintes dependências:

- Instalar o compilador g++ da GNU disponível em <http://gcc.gnu.org>.

### 9.2 Interface gráfica

A interface do programa é apresentada na Figura 9.1.

Figura 9.1: Imagem da interface gráfica



Fonte: Produzido pelo autor.

A Figura 9.1 mostra as principais opções de configuração e operação disponíveis ao usuário no menu do programa, organizadas em funcionalidades específicas para a simulação de poços de petróleo.

### 9.3 Funcionalidades

- Configurar Poço: permite ao usuário definir e ajustar as principais propriedades do poço e fluido, oferecendo três opções:
  - Criar Poço: para configurar propriedades iniciais, como profundidade, diâmetro e revestimento.
  - Adicionar Fluido: para definir as características dos fluidos presentes no poço, incluindo densidade e viscosidade.
  - Carregar Dados a Partir de Arquivo (.dat): para importar as configurações do poço e fluidos a partir de um arquivo .dat, facilitando a inicialização da simulação com dados externos.
- Exibir Propriedades do Poço: exibe as características e parâmetros do poço, incluindo profundidade total, diâmetro, e propriedades dos fluidos, organizadas para fácil visualização.
- Calcular Pressão Hidrostática do Poço: permite ao usuário calcular a pressão exercida pelo fluido em diferentes pontos do poço, com duas opções:
  - Calcular Pressão Hidrostática (Fundo de Poço): exibe a pressão no fundo do poço considerando a profundidade total e a densidade do fluido.

- Calcular Pressão Hidrostática em um Ponto do Poço: permite calcular a pressão hidrostática em uma profundidade específica, escolhida pelo usuário.
- Plotar Perfil Profundidade por Densidade: gera um gráfico que mostra a variação de densidade dos fluidos em relação à profundidade, facilitando a visualização da distribuição dos fluidos ao longo do poço.
- Calcular Perda de Carga por Fricção: oferece cálculo detalhado da perda de carga utilizando três modelos reológicos distintos, com opções para poço e espaço anular.
  - Modelo Newtoniano
    - \* Determinar Perda de Carga no Poço: realiza o cálculo de perda de carga considerando um modelo Newtoniano no espaço do poço.
    - \* Determinar Perda de Carga no Anular: cálculo de perda de carga utilizando um modelo Newtoniano para o espaço anular.
  - Modelo Bingham
    - \* Determinar Perda de Carga no Poço: cálculo para perda de carga aplicando o modelo Bingham no espaço do poço.
    - \* Determinar Perda de Carga no Anular: utiliza o modelo Bingham para calcular a perda de carga no espaço anular.
  - Modelo de Potência
    - \* Determinar Perda de Carga no Poço: realiza o cálculo com o modelo de potência no espaço do poço.
    - \* Determinar Perda de Carga no Anular: cálculo da perda de carga no espaço anular com o modelo de potência.

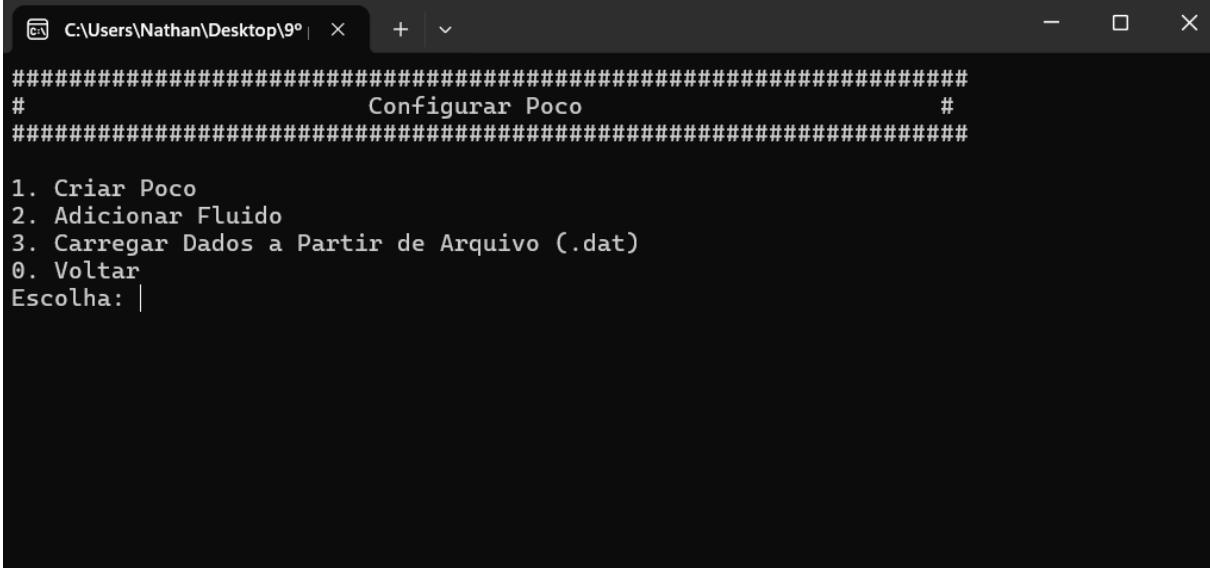
## 9.4 Como adicionar informações do poço/fluido

Apresenta-se neste apêndice instruções para configuração do Poço/Fluido.

Para configurar um poço ou fluido no software, o usuário tem duas opções: realizar a configuração manualmente acessando **Configurar Poço** → **Criar Poço** para definir as propriedades do poço, ou **Configurar Poço** → **Adicionar Fluido** para especificar as características dos fluidos.

A Figura 9.2 ilustra o local do menu que deve ser selecionado para adicionar o material.

Figura 9.2: Configurar o poço no simulador: funções de criar o poço e impostar o dados a partir de um arquivo .dat



```

C:\Users\Nathan\Desktop\9º | × + ▾
#####
#          Configurar Poco      #
#####

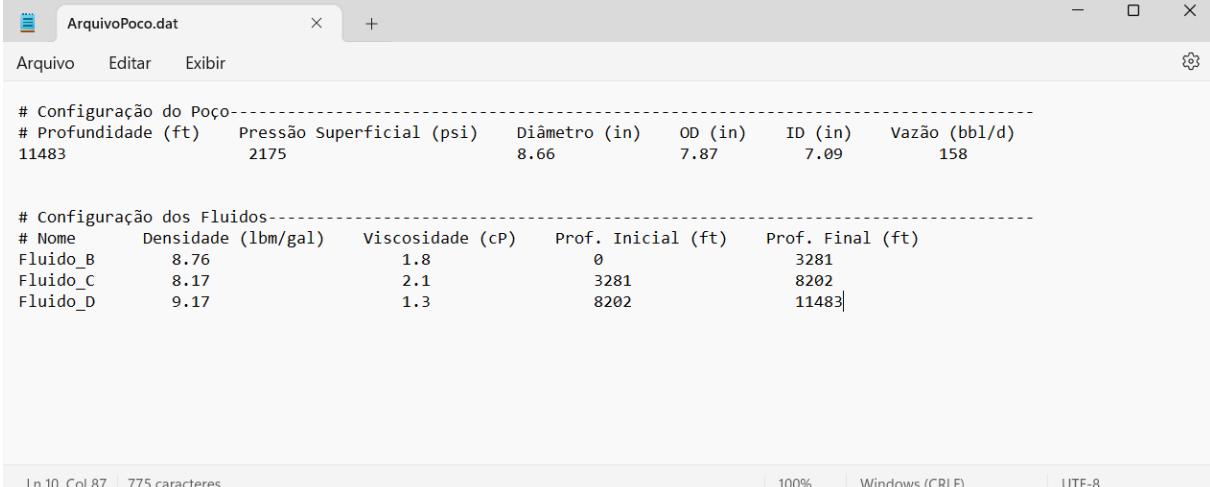
1. Criar Poco
2. Adicionar Fluido
3. Carregar Dados a Partir de Arquivo (.dat)
0. Voltar
Escolha: |

```

Fonte: Produzido pelo autor.

Além disso, se preferir, o usuário pode carregar os dados diretamente a partir de um arquivo .dat. Nesse arquivo, as configurações devem seguir uma ordem específica, onde apenas um poço e os fluidos são listados, um abaixo do outro, conforme a sequência mostrada na Figura 9.3.

Figura 9.3: Arquivo .dat para adicionar propriedades do poço e dos fluidos no simulador



```

ArquivoPoco.dat | × + | - □ × | ⚙
Arquivo   Editar   Exibir | Ln 10, Col 87 | 775 caracteres | 100% | Windows (CRLF) | UTF-8

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

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

```

Fonte: Produzido pelo autor.

Dessa forma após configurar o poço o usuário poderá explorar as funcionalidade do software descritas na Seção 9.3.



# Referências Bibliográficas

- BUENO, André Duarte. 2003. *Programação orientada a objeto com c++*. Novatec. 32
- HALLIDAY, David, & RESNICK, Jearld Walker. 2009. *Fundamentos de física, volume 2: gravitação, ondas e termodinâmica*. 24
- Jr., Adam T. Bourgoyné, Millheim, Keith E., Chenevert, Martim E., & Jr., F. S. Young. 1991. *Applied drilling engineering*. Society of Petroleum Engineers. 9, 20, 21, 22, 26
- Mitchell, Robert F., & Miska, Stefam Z. 2011. *Fundamentals of drilling engineering*. Society of Petroleum Engineers. 9, 23, 24, 25