UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE
LABORATÓRIO DE ENGENHARIA E EXPLORAÇÃO DE PETRÓLEO

PROJETO DE ENGENHARIA
DESENVOLVIMENTO DO SOFTWARE:
SIMULADOR DE CURVAS IPR UTILIZANDO MODELOS EMPÍRICOS EM POÇOS VERTICAIS

TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA

Versão 1:

ATILA JUNIOR
GIOVANNA MASSARDI
MARCELO BERNARDO
Prof. André Duarte Bueno

MACAÉ - RJ

Dezembro - 2023

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

No presente projeto de engenharia, desenvolve-se o software de Simulação de Curvas IPR (Inflow Performance Relationship) utilizando Modelos Empíricos em Poços verticais, um software aplicado a engenharia de petróleo e que utiliza o paradigma da orientação a objetos.

Este software tem como finalidade obter curvas de IPR no regime pseudopermanente a partir de dados inseridos pelo usuário. Para isso, é necessário que o índice de produtividade seja calculado com base nos dados de pressão e vazão ou propriedades do reservatório. Dessa forma, os modelos empíricos Linear, Fetkovich, Vogel e Vogel Generalizado poderão ser escolhidos para realização do cálculo da pressão de fundo e respectivas vazões bem como avaliar reservatórios estratificados a partir dos cálculos resultantes. Tais valores serão mostrados ao usuário juntamente com um gráfico com a curva de IPR e estas informações poderão ser salvas em disco.

1.1 Escopo do problema

A curva de IPR, que pode ser chamada de curva de influxo, curva de pressão disponível ou curva do índice de produtividade, é uma representação gráfica que descreve como a pressão disponível no fundo de um poço de petróleo ou gás varia em relação à taxa de fluxo de fluidos. Essas medidas são tomadas na profundidade em que o reservatório foi perfurado em um momento específico durante a vida útil do campo. Em outras palavras, essa curva fornece informações cruciais sobre como a pressão no poço reage quando se está produzindo hidrocarbonetos.

Através da análise dessa curva, as decisões estratégicas podem ser tomadas para maximizar a eficiência e recuperação de petróleo ou gás, além de demonstrar os cenários nos quais intervenções deverão ser realizadas assim como definir quando perfurar novos poços. Com isso, é possível analisar quando há possíveis limitações no desempenho do poço. Ademais, esta ferramenta é frequentemente utilizada em simulações de reservatório a fim de prever o comportamento futuro e auxiliar em um planejamento de longo prazo.

O cálculo da IPR pode sofrer variações dependendo das características do reservatório e se estes dados são conhecidos. Logo, alguns modelos matemáticos podem ser aplicados para obter a curva de IPR e inúmeros métodos podem ser escolhidos com base nos regimes de escoamento, tipos de fluxo e fluidos existentes. Portanto, estas equações e suas respectivas análises podem ser encontradas na literatura e, facilitar o cálculo e plotagem das curvas é algo extremamente relevante para simulação de reservatórios e para o entedimento de elevação e escoamento de fluidos.

1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivos gerais:
  - Utilizar modelos empíricos e equações matemáticas propostas na literatura para o cálculo das pressões de poço e suas respectivas vazões ao longo da vida útil de um reservatório.
  - Plotar curvas de IPR em escoamentos monofásico e bifásico utilizando os modelos Linear, Vogel, Fetkovich e Vogel Generalizado considerando regime pseudopermanente a partir do sofware externo Gnuplot.
- Objetivos específicos:
  - Permitir que o usuário escolha qual modelo irá utilizar para realizar o cálculo dos parâmetros da curva de IPR.
  - Criar o software de modo que a curva de IPR possa ser plotada independente da existência dos parâmetros de reservatório.
  - A partir do cálculo dos modelos empíricos, analisar os tipos de IPR: Linear, Generalizada e Reservatórios Estratificados.

Capítulo 2

Especificação

Nesta seção do projeto de engenharia, é apresentada a especificação do software a ser desenvolvido para aplicação em sistemas de modelagem de curvas IPR, utilizado modelos empíricos em poços verticais. Para a indústria de Óleo e Gás, este processo é altamente relevante, pois ao dispor de um software capaz de construir curvas IPR (Figura), será possível estimar características importantes acerca dos poços analisados, dentre elas a sua produtividade. Além disso, o projeto visa estimar e fornecer ao usuário o momento em que será necessário utilizar-se de métodos de recuperação terciária, como a elevação artificial.

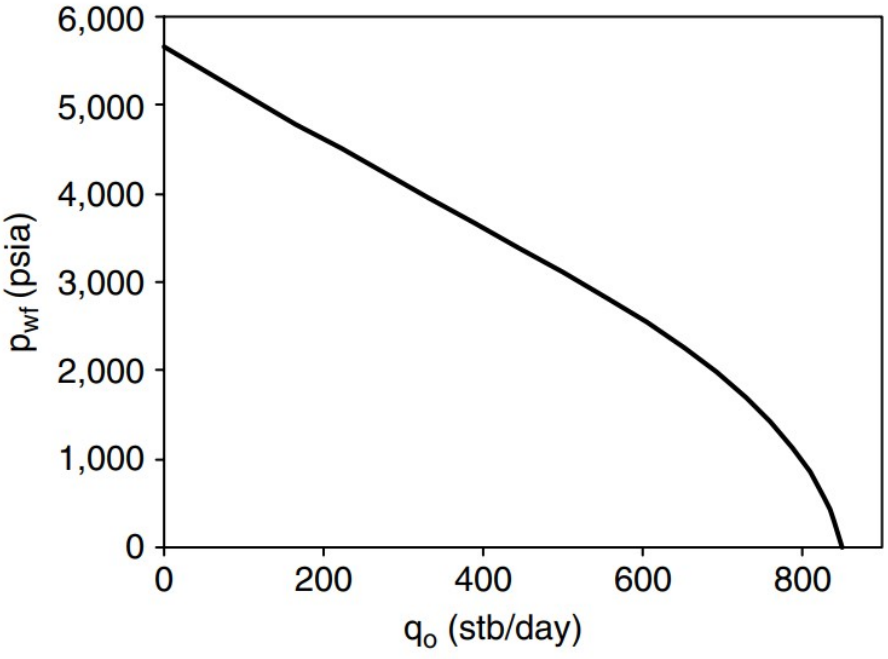

Figura 2.1: Curva IPR típica de um reservatório de óleo

2.1 Nome do sistema/produto

Nome	Simulação de Curvas IPR Utilizando Modelos Empíricos para Poços Verticais
Componentes principais	- Implementação de Modelos Empíricos de IPR: Será desenvolvida uma gama diversificada de modelos empíricos, os quais representarão com precisão os padrões de produção em poços verticais de petróleo. - Entrada Customizada: O software possibilitará aos usuários inserirem parâmetros específicos do poço, tais como propriedades dos fluidos e do reservatório, para conduzir simulações altamente personalizadas. - Cálculos de Produtividade: O software executará cálculos complexos para estimar a produtividade prevista em um poço vertical, com base nos modelos empíricos selecionados e nos dados fornecidos. - Visualização Gráfica: O software será equipado com recursos de visualização gráfica, a fim de exibir de forma clara e compreensível os resultados, facilitando a análise e interpretação dos resultados.
Missão	A missão subjacente a este software é disponibilizar aos estudantes do curso de Elevação e Escoamento e à indústria uma ferramenta de simulação de curvas IPR de alta qualidade. Por meio de uma interface intuitiva e da capacidade de fornecer estimativas precisas de produtividade, o software tem por objetivo apoiar engenheiros, pesquisadores e profissionais do setor em suas decisões e na otimização da produção de poços verticais. Fundindo a precisão dos modelos com a flexibilidade de entrada dos dados, essa solução almeja tornar-se uma ferramenta indispensável para avaliação e planejamento de operações em poços de petróleo.

2.2 Especificação

O presente projeto tem como objetivo a criação de um software avançado de simulação de curvas IPR (Inflow Performance Relationship), utilizando modelos empíricos para poços verticais de petróleo. A simulação de curvas IPR desempenha um papel crucial na indústria de petróleo, permitindo estimativas precisas da produtividade dos poços e auxiliando na tomada de decisões estratégicas. O software a ser desenvolvido busca proporcionar uma ferramenta poderosa para engenheiros e profissionais do setor, que desejam analisar e otimizar a produção de poços verticais de maneira eficiente e eficaz.

O fundamento deste projeto conta com a aplicação de modelos empíricos conhecidos, incluindo as abordagens de Fetkovich, Vogel, e outras metodologias de IPR generalizada. Esses modelos têm sido fundamentais na avaliação da performance de poços de petróleo, considerando variáveis complexas como propriedades do reservatório, características dos fluidos e condições operacionais. O software que está sendo desenvolvido permitirá aos usuários empregarem esses modelos com facilidade, fornecendo uma plataforma de simulação confiável e precisa.

Os modelos empíricos como Fetkovich e Vogel têm sido amplamente utilizados para representar a relação entre a vazão de óleo e a pressão na entrada do poço. O Fetkovich é reconhecido pela sua aplicabilidade a diferentes regimes de fluxo, enquanto o modelo Vogel oferece uma abordagem simplificada para estimar a produtividade. Além disso, abordar-se-á a adaptação desses modelos para situações específicas, como poços estratificados, onde a heterogeneidade do reservatório é levada em consideração para uma simulação mais precisa.

Em suma, o projeto em questão assume uma importância significativa na indústria de petróleo, fornecendo uma solução de simulação avançada para a análise de curvas IPR em poços verticais. Com modelos empíricos consagrados como Fetkovich e Vogel, juntamente com adaptações para poços estratificados e outras variações, o software busca simplificar e aprimorar a avaliação da produtividade de poços de petróleo. Com isso, engenheiros e profissionais do setor terão uma ferramenta valiosa para otimizar a produção e tomar decisões embasadas no setor de exploração e produção de petróleo.

O software será desenvolvido utilizando o conceito de programação orientada a objeto, interface intuitiva e utilizará o software externo Gnuplot para gerar e salvar os gráficos.

2.2.1 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

RF-01	O usuário deve ter liberdade para escolher quais parâmetros de entrada utilizar, utilizando teclado ou um arquivo .txt.

RF-02	O usuário deve escolher o modelo adequado para seu problema de engenharia específico.

RF-03	O usuário poderá plotar suas curvas em um gráfico utilizando o Gnuplot. O gráfico poderá ser salvo como imagem.

2.2.2 Requisitos não funcionais

RNF-01	Os cálculos devem ser feitos utilizando-se formulações/modelos matemáticos conhecidos na literatura.

RNF-02	Usuário deve ter conhecimento básico e prévio de assuntos sobre Elevação e Escoamento para escolha do método mais adequado.

RNF-03	O programa deverá ser multi-plataforma, podendo ser executado em *Windows*, *GNU/Linux* ou *Mac*.

2.3 Casos de uso

Nesta seção, apresenta-se uma tabela que especifica um caso de uso para o software, o diagrama de caso de uso geral e o diagrama de caso de uso específico para casos onde o reservatório é estratificado, isto é, cada camada possui diferentes propriedades.

Tabela 2.1: Caso de uso

Nome do caso de uso:	Simulação de curvas IPR de um reservatório.
Resumo/descrição:	Determinação das curvas IPR através das propriedades dos fluidos, rocha e poço para análise do comportamento do reservatório ao longo do tempo.
Etapas:	1. Definição do número de camadas. 2. Entrada de dados do reservatório, fluidos e poço via teclado ou arquivo .txt. 3. Definir método adequado para a simulação. 4. Cálculo da vazão ao longo do tempo. 5. Gerar gráfico. 6. Analisar resultados.
Cenários alternativos:	Inserir valores negativos ou incompatíveis com a ordem de grandeza de um reservatório real.

2.3.1 Diagrama de caso de uso geral

O diagrama de caso de uso geral da Figura 2.2 mostra o usuário interagindo com o software para obter as curvas de IPR do reservatório. Nesse caso, o usuário insere os dados do problema (propriedades dos fluidos, reservatório e poço), selecionando o modelo mais adequado para o problema que ele possuir e analisando os resultados obtidos.

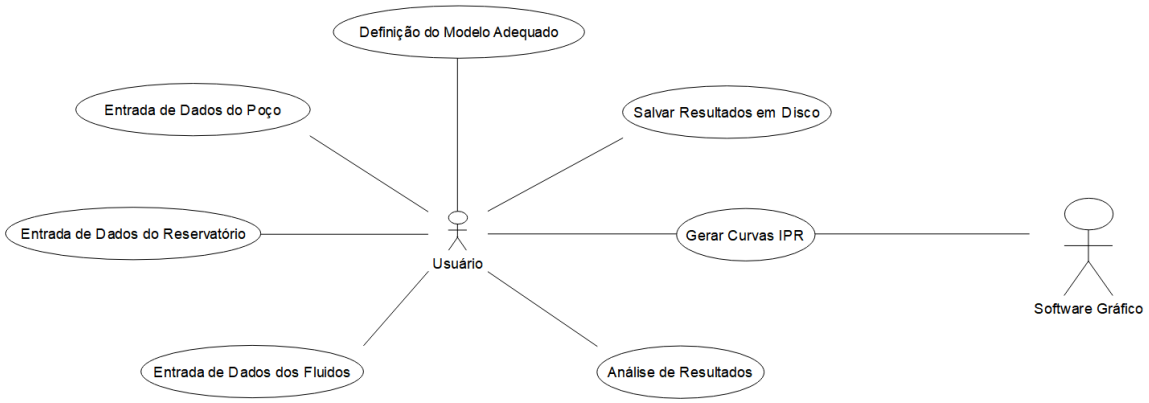

Figura 2.2: Diagrama de caso de uso – Caso de uso geral

2.3.2 Diagrama de caso de uso específico

O caso de uso para reservatórios homogêneos é descrito na Figura 2.2 e na Tabela 2.1. Para reservatórios estratificados, o processo é detalhado na Figura 2.3. O usuário entrará com o número de camadas do reservatório e com as propriedades de cada camada do reservatório para obtenção das curvas.

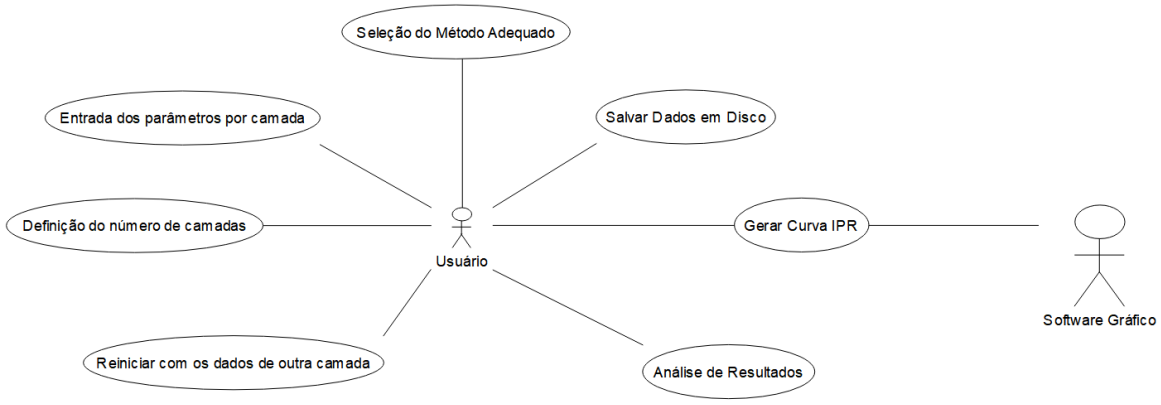

Figura 2.3: Diagrama de caso de uso específico – Reservatórios Estratificados

Capítulo 3

Elaboração

Neste capítulo, após a definição dos objetivos, da especificação do software e da montagem dos primeiros diagramas de caso de uso, será apresentada a etapa da elaboração que aborda estudos e análises de conceitos relacionados ao sistema desenvolvido, isto é, análise de domínio e identificação de pacotes. Neste sentido, será empregada uma análise de requisitos de modo a ajustá-los aos requisitos iniciais a fim de desenvolver um sistema útil, que atenda às necessidades do usuário e que possa ser, possivelmente, reutilizada e estendida.

3.1 Análise de domínio

O Software a ser desenvolvido aborda formas de estimar a produtividade de um poço vertical de petróleo através de modelagem de curvas IPR, utilizando modelos empíricos. O fato de se explorar este tema é de fundamental relevância no campo da produção de petróleo, pois traz informações importantes em relação ao rendimento e vida útil que o poço analisado terá. A capacidade de entrega do reservatório é definida como a taxa de produção alcançável de óleo ou gás a partir do reservatório em uma dada pressão no fundo do poço. É um fator importante que afeta a sua capacidade de entrega. Por sua vez, a capacidade de entrega do reservatório determina os tipos de completação e os métodos de elevação artificial a serem utilizados. Dessa forma, um conhecimento completo da produtividade do reservatório é essencial para engenheiros de produção. A produtividade do reservatório depende de diversos fatores, como:

- Pressão do Reservatório: A pressão do reservatório é um fator crítico que influencia diretamente a produtividade. Uma pressão mais alta pode impulsionar a migração do petróleo ou gás em direção ao poço, aumentando a taxa de produção.

- Espessura de *pay zone*: A *pay zone* se refere à camada do reservatório que contém o petróleo ou gás. Quanto mais espessa essa camada, mais espaço há para armazenar hidrocarbonetos, o que pode aumentar a produtividade.

- Permeabilidade: A permeabilidade está relacionada à capacidade do reservatório de permitir que o petróleo ou gás fluam através dele. Uma maior permeabilidade facilita a movimentação dos fluidos, contribuindo para uma maior produtividade.

- Propriedades dos fluidos do reservatório: As propriedades dos fluidos, como a viscosidade do petróleo e a composição do gás, influenciam a facilidade com que os fluidos podem ser extraídos. Fluidos mais viscosos podem dificultar o fluxo, reduzindo a produtividade.

- Permeabilidade Relativa do Reservatório: A permeabilidade relativa é a medida de quão facilmente os diferentes fluidos (por exemplo, óleo, gás e água) se movem no reservatório. Se a permeabilidade relativa do óleo for baixa em relação ao gás ou água, o petróleo pode ter dificuldades em ser deslocado, afetando a produtividade.

O software desenvolvido terá a capacidade de fornecer ao cliente resultadosda produtividade e vida útil do reservatório, levando em consideração todos os requisitos, especificações e conceitos de Engenharia de Petróleo apresentados na Introdução.

3.2 Formulação teórica

Nesta seção, apresenta-se a formulação teórica dos conceitos fundamentais abordados ao longo deste projeto. A mesma tem como base o livro[**?**]

3.2.1 *Inflow Performance Relationship*

A curva IPR (*Inflow Performance Relationship*) é usada para avaliar a capacidade de entrega do reservatório na engenharia de produção. A curva IPR é uma apresentação gráfica da relação entre a pressão de fluxo no fundo do poço e a taxa de produção de líquidos. A magnitude da inclinação da curva IPR é chamada de "índice de produtividade" (J):

$$J = \frac{q}{(p_e - p_{wf})} \tag{3.1}$$

.

As curvas IPR de poços são geralmente construídas usando modelos de vazão de reservatórios, que podem ser de base teórica ou empírica.

3.2.2 IPR Linear

A IPR Linear é aplicada na suposição de fluxo líquido monofásico e, dessa forma, funciona para zonas do reservatório acima do ponto de bolha ou para óleos subsaturados.

As equações definem o índice de produtividade (J) (equação 3.2) para a pressão de fundo de poço acima da pressão do ponto de bolha:

$$J^* = \frac{q}{(p_e - p_{wf})} = \frac{kh}{141.2B_o\left(\frac{1}{2}ln\frac{4A}{\gamma C_A r_w^2 + S}\right)} \tag{3.2}$$

Como o índice de produtividade (J) acima da pressão do ponto de bolha é independente da taxa de produção, a curva IPR para um reservatório monofásico (líquido) é uma linha reta traçada da pressão do reservatório até a pressão do ponto de bolha. Se a pressão do ponto de bolha for 0 $psig$, o fluxo aberto absoluto (AOF) é o índice de produtividade (J) vezes a pressão do reservatório .

$$AOF = JPe$$

3.2.3 Reservatórios Bifásicos

Acima da pressão de bolha, o óleo contido no reservatório se encontra subsaturado, ou seja, o gás contido no óleo se encontra todo dissolvido, fazendo com que a IPR apresente um comportamente linear. Abaixo da pressão do ponto de bolha, o gás em solução escapa do óleo e se torna gás livre.

O gás livre ocupa alguma porção do espaço poroso, o que reduz o fluxo de óleo. Este efeito é quantificado pela permeabilidade relativa reduzida. Além disso, a viscosidade do óleo aumenta à medida que o conteúdo do gás em solução diminui. A combinação do efeito de permeabilidade relativa e do diminuição da viscosidade resulta em menor taxa de produção de petróleo a uma determinada pressão de fundo de poço. Isto faz com que a curva IPR se desvie da tendência linear abaixo da pressão do ponto de bolha. Quanto menor a pressão, maior o desvio. Se a pressão do reservatório estiver abaixo da pressão inicial do ponto de bolha, existe fluxo bifásico de petróleo e gás em todo o domínio do reservatório.

Apenas equações empíricas estão disponíveis para modelar IPR de reservatórios bifásicos. Essas equações empíricas incluem a equação de Vogel (1968) e a equação de Fetkovich (1973). A equação de Vogel ainda é amplamente utilizada na indústria.

3.2.4 Equação de Vogel

A equação de Vogel é descrita por:

$q = q_{max}\left[1 - 0.2\left(\frac{p_{wf}}{\overline{p}}\right) - 0.8\left(\frac{p_{wf}}{\overline{p}}\right)^2\right]$ ou $p_{wf} = 0.125\overline{p}\left[\sqrt{81 - 80\left(\frac{q}{q_{max}}\right)} - 1\right]$,

onde $qmax$ é uma constante empírica e seu valor representa a capacidade máxima que o reservatório pode entregar, ou AOF. Para fluxo pseudo-permanente pode ser calculado por:

$q_{max} = \frac{J*\overline{p}}{1.8}$.

3.2.5 Equação de Fetkovich

A equação de Fetkovich é definida por:
$q = q_{max}\left[1 - \left(\frac{p_{wf}}{\overline{p}}\right)^2\right]^n$ ou $q = C\left(\overline{p}^2 - p_{wf}^2\right)^n$,
onde C e n são constantes empíricas relacionadas a vazão máxima, dada por:
$C = \frac{q_{max}}{\overline{p^{2n}}}$.

3.2.6 IPR Generalizada

Acima do ponto de bolha a IPR apresenta comportamento linear e a vazão na pressão de bolha é dada por:
$q = J^*(\overline{p} - p_b)$.
Baseado na equação de Vogel, a vazão adicional causada pela queda de pressão abaixo do ponto de bolha é expressada como:
$\Delta q = q_v\left[1 - 0.2\left(\frac{p_{wf}}{p_b}\right) - 0.8\left(\frac{p_{wf}}{p_b}\right)^2\right]$, onde $q_v = \frac{J^* p_b}{1.8}$.
Portanto, a vazão para uma determinada pressão abaixo da pressão de bolha é:
$q = J^*\left(\overline{p} - p_b\right) + \frac{J^* p_b}{1.8} * \left[1 - 0.2\left(\frac{p_{wf}}{p_b}\right) - 0.8\left(\frac{p_{wf}}{p_b}\right)^2\right]$.

3.2.7 Fluxo Monofásico

Para camadas de reservatório contendo óleos subsaturados, se a pressão de fluxo no fundo do poço estiver acima das pressões do ponto de bolha dos óleos em todas as camadas, é esperado fluxo monofásico.
$\sum J^*(\overline{p}_i - p_{wf}) = q_{wh}$
$AOF = \sum J_i^* \overline{p}_i = \sum AOF_i$
Também é possível se chegar a pressão de fundo. É importante ressaltar que Pwfo é uma pressão dinâmica de fundo de poço devido ao fluxo cruzado entre as camadas.
$p_{wfo} = \frac{\sum J_i^* \overline{p_i}}{\sum J_i^*}$.

3.2.8 Fluxo Bifásico

Para camadas de reservatório contendo óleos saturados, é esperado um fluxo bifásico. Utilizando Vogel:
$\sum \frac{J_i^* \overline{p}_i}{1.8}\left[1 - 0.2\left(\frac{p_{wf}}{\overline{p_i}}\right) - 0.8\left(\frac{p_{wf}}{\overline{p_i}}\right)^2\right] = q_{wh}$
$AOF = \sum \frac{J_i^* \overline{p}_i}{1.8} = \sum AOF_i$
Também é possível se chegar a pressão de fundo:
$p_{wfo} = \frac{\sqrt{80 \sum J_i^* \overline{p}_i \sum \frac{J_i^*}{\overline{p}_i} + \left(\sum J_i^*\right)^2} - \sum J_i^*}{8 \sum \frac{J_i^*}{\overline{p}_i}}$.

3.3 Identificação de pacotes – assuntos

A partir da análise dos modelos apresentados, identifica-se ose seguintes assuntos/pacotes:

- Pacote Reservatório:
  - Composto pelos parâmetros da rocha-reservatório(como porosidade, permeabilidade), fluidos(como viscosidade) e do próprio reservatório(como , extensão, fator volume-formação) e outros.
- Pacote Dados de Produção:
  - Composto por dados de pressão de fundo e inicial do reservatório de modo que o cálculo da IPR seja possível sem necessitar dos parâmetros do pacote acima descrito.
- Pacote Modelos Empíricos:
  - Calcula os parâmetros necessários para as curvas utilizando os modelos listados. O usuário poderá escolher qual modelo deseja usar.
- Pacote Regime Pseudopermanente:
  - Permite calcular o parâmetro de índice de produtividade o qual será utilizado nos cálculos de vazão posteriormente.
- Pacote Simulador:
  - Relaciona os pacotes listados, sendo responsável por interagir com o usuário através de um interface via texto para definir as ações a serem tomadas.
- Pacote Gráficos:
  - Utilizando um software externo, será possível gerar gráficos de IPR.

3.4 Diagrama de pacotes – assuntos

O diagrama de pacotes da Figura 3.1 mostra as relações e dependências existentes entre os pacotes deste software.

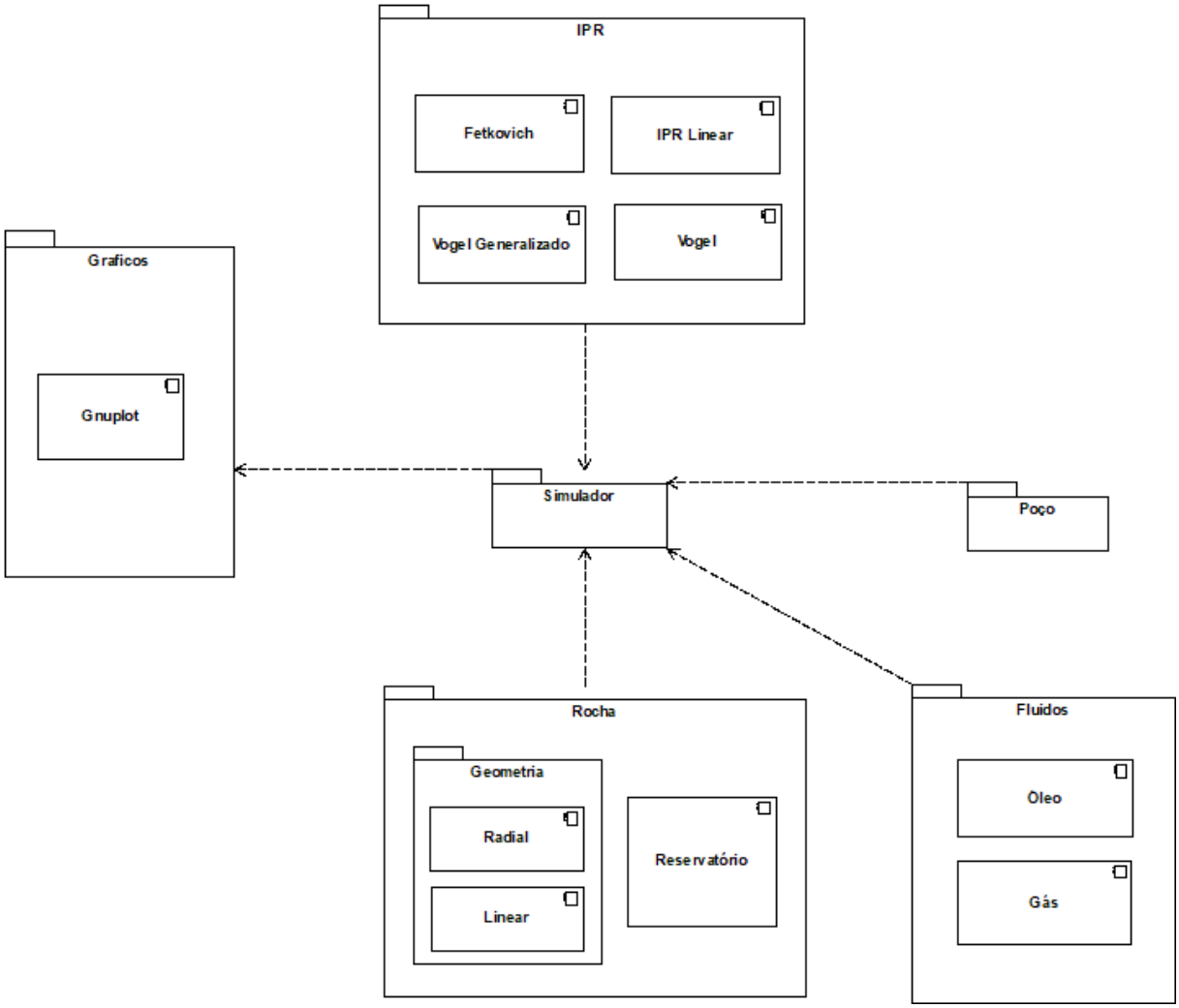

Figura 3.1: Diagrama de Pacotes

Capítulo 4

AOO – Análise Orientada a Objeto

A terceira etapa do desenvolvimento de um projeto de engenharia, no nosso caso um software aplicado a engenharia de petróleo, é a AOO – Análise Orientada a Objeto. A AOO utiliza algumas regras para identificar os objetos de interesse, as relacões entre os pacotes, as classes, os atributos, os métodos, as heranças, as associações, as agregações, as composições e as dependências.

4.1 Diagramas de classes

O diagrama de classes é apresentado na Figura 4.1.

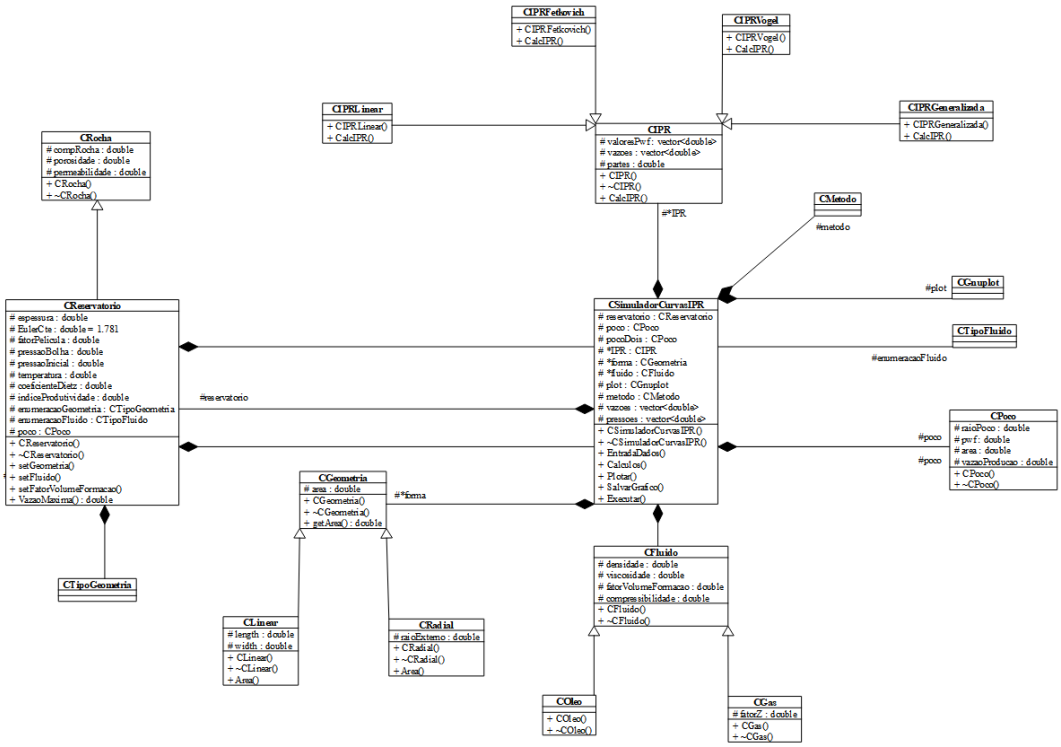

Figura 4.1: Diagrama de classes

4.1.1 Dicionário de classes

- Classe CPoco: representa a classe responsável por receber os parâmetros do poço
- Classe CFluido: representa a classe responsável por receber as propriedades do fluido.
- Classe COleo: representa a classe responsável por receber os parâmetros do óleo quando a intenção do usuário é simular curvas IPR de óleo.
- Classe CGas: representa a classe responsável por receber os parâmetros do gás quando a intenção do usuário é simular curvas IPR de gás.
- Classe CRocha: representa a classe responsável por receber os parâmetros de rocha.
- Classe CReservatorio: representa a classe responsável por receber todos os parâmetros referentes ao reservatório.
- Classe CGeometria: representa a classe responsável por definir a geometria do reservatório.
- Classe CRadial: representa a classe responsável por receber e calcular dimensões do reservatório de geometria radial.
- Classe CLinear: representa a classe responsável por receber e calcular dimensões do reservatório de geometria linear.
- ClasseCIPR: representa a classe base que recebe os vetores de pressão de fundo e vazão.
- Classe CIPRLinear: representa a classe responsável por calcular a curva IPR para reservatórios acima da pressão de bolha.
- Classe CIPRFetkovich: representa a classe responsável pelo cálculo da curva IPR utilizando o modelo de Fetkovich.
- Classe CIPRVogel: representa a classe responsável pelo cálculo da curva IPR utilizando o modelo de Vogel.
- Classe CIPRGeneralizada: representa a classe responsável pelo cálculo da curva IPR utilizando o modelo de Vogel generalizado para reservatórios bifásicos.
- Classe CMetodo: classe que representa a enumeração dos tipos de métodos de cálculo de IPR.
- Classe CTipoGeometria: classe que representa a enumeração dos tipos de geometria do reservatório.

- Classe CTipoFluido: classe que representa a enumeração dos tipos de fluidos.
- Classe CSimuladorCurvasIPR: representa a classe responsável pela simulação do software. Recebe as escolhas do usuário quanto aos parâmetros do poço, reservatório e fluidos, além da escolha do modelo adequado para a obtenção das curvas de IPR.
- Classe CGrafico: representa a classe responsável pela parte gráfica do programa, a partir do uso do software externo Gnuplot.

4.2 Diagrama de seqüência – eventos e mensagens

O diagrama de seqüência enfatiza a troca de eventos e mensagens e sua ordem temporal. Portanto, representa uma forma de diagrama interativo que delineia a maneira e a sequência em que um conjunto de objetos colabora.

4.2.1 Diagrama de sequência geral

Veja o diagrama de seqüência geral na Figura 4.2. Notar que a entrada de dados pode ser realizada tanto via disco como via teclado, deixando a escolha do usuário

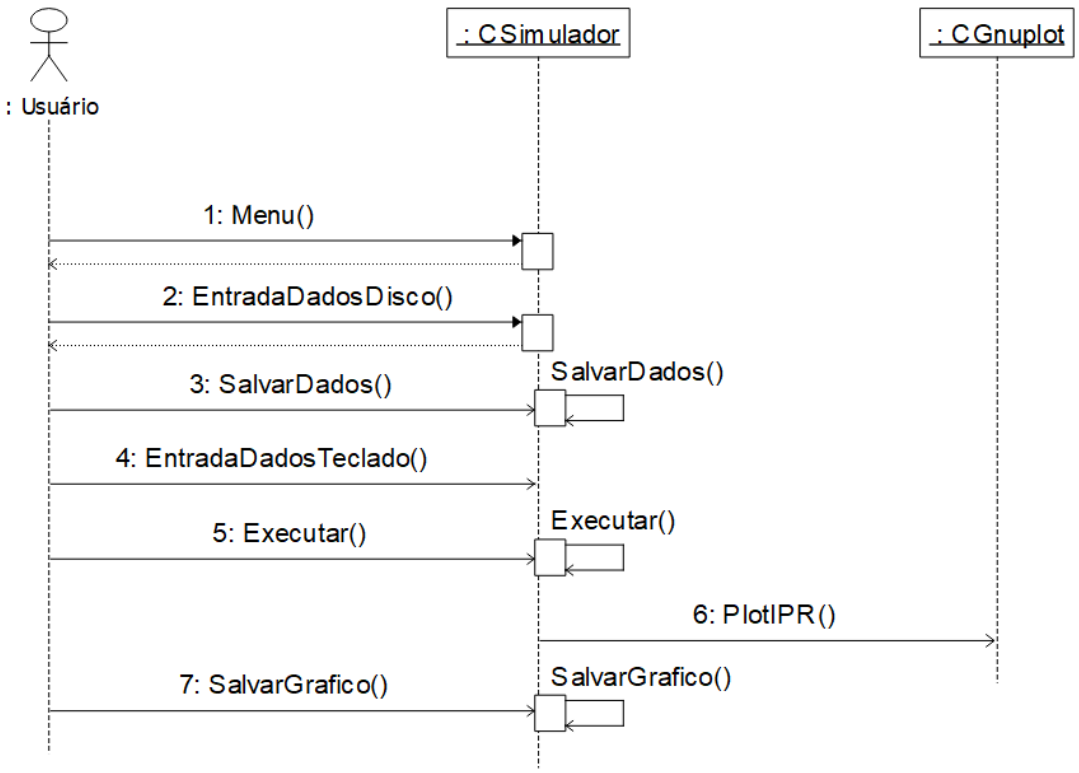

Figura 4.2: Diagrama de seqüência

4.2.2 Diagrama de sequência específico

Para o caso de um diagrama de sequência específico para cálculo dos parâmetros das curvas de IPR, é necessário que o usuário forneça os dados de fluido, reservatório e poço através do teclado, bem como escolha o método que deseja utilizar para calcular e obter os resultados que serão utilizados no gráfico final. Veja o diagrama de sequência na Figura 4.3.

4.3 Diagrama de comunicação – colaboração

No diagrama de comunicação o foco é a interação e a troca de mensagens e dados entre os objetos.

•

Veja na Figura 4.3 o diagrama de comunicação mostrando a sequência de como as classes se comunicam entre si para o funcionamento do software. Observe que o método EntradaDados() pode ser representado através dos métodos EntradaDadosDisco() ou EntradaDadosTeclado(), ficando a critério do usuário selecionar qual modo de entrada de dados atende melhor à sua demanda.

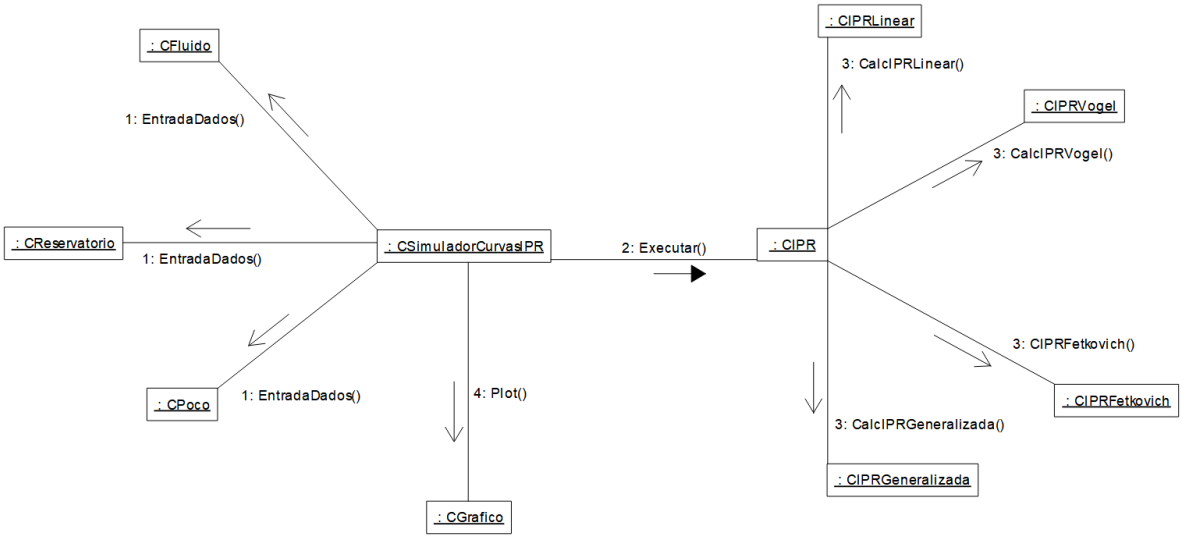

Figura 4.4: Diagrama de comunicação

4.4 Diagrama de máquina de estado

Um diagrama de máquina de estado, também conhecido como diagrama de estados ou diagrama de transição de estados, é uma representação visual que descreve o comportamento de um sistema ou entidade em relação aos diferentes estados em que pode estar e às transições entre esses estados. Essa ferramenta é amplamente utilizada na engenharia

de software, engenharia de sistemas e em outros campos para modelar o comportamento de sistemas complexos.

É possível observar na figura 4.5 o diagrama de máquina de estado para os objetos da classe CsimuladorCurvasIPR.

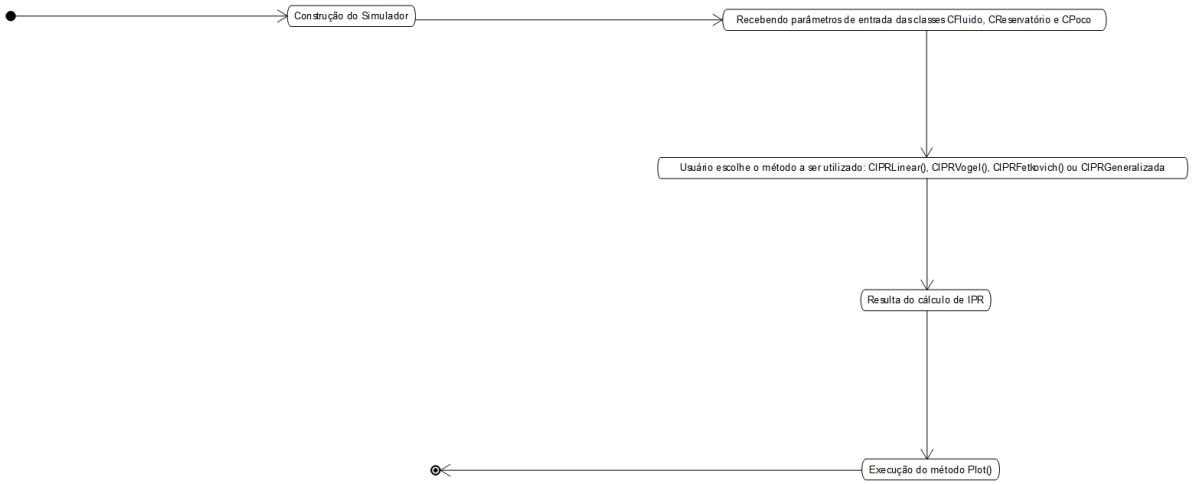

Figura 4.5: Diagrama de máquina de estado

4.5 Diagrama de atividades

Um diagrama de atividades é um tipo de diagrama da Linguagem de Modelagem Unificada (UML) que representa o fluxo de controle de atividades em um sistema. Ele é essencialmente um gráfico de fluxo que mostra o fluxo de controle de uma atividade para outra.

Veja na Figura 4.6 o diagrama de atividades correspondente a uma atividade específica do diagrama de máguina de estado. Neste caso específico, é realizado o cálculo da produtividade de um poço vertical, utilizando-se de modelos teóricos e empíricos.

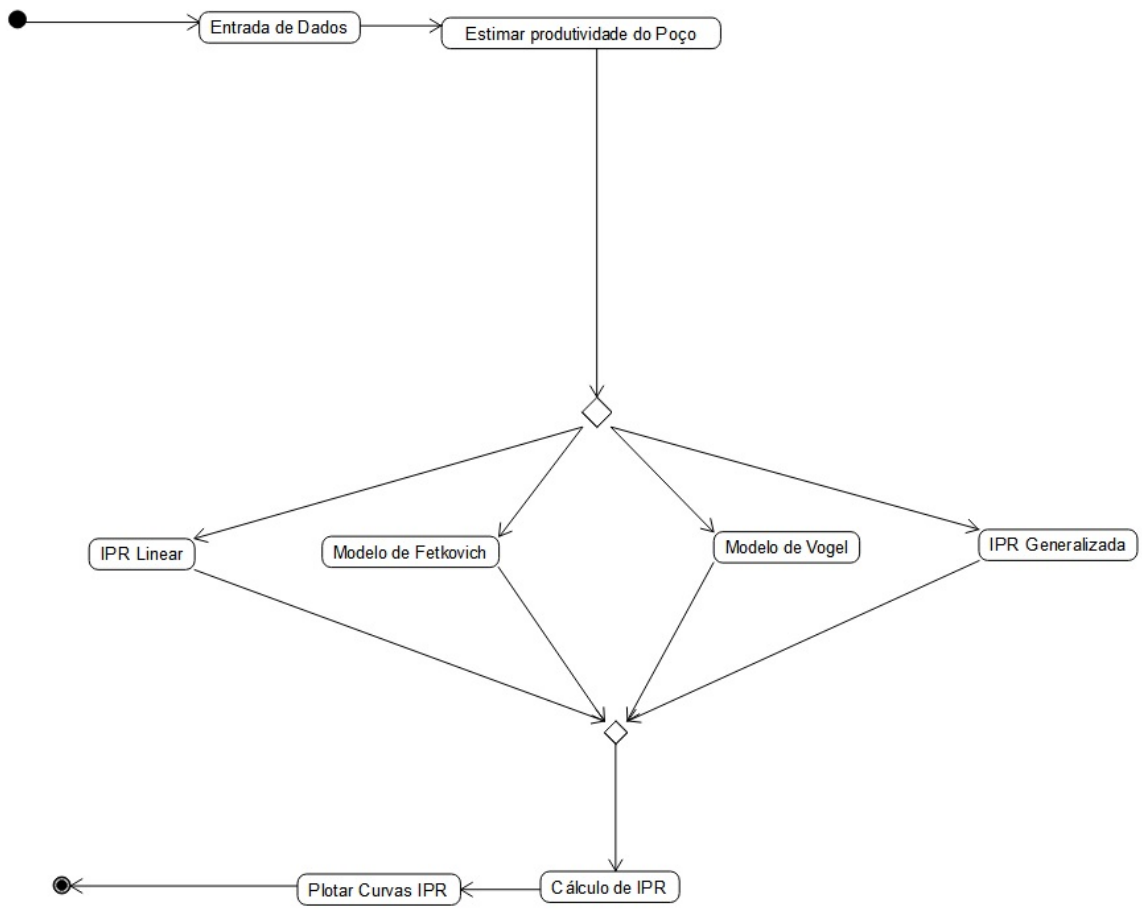

Figura 4.6: Diagrama de atividades

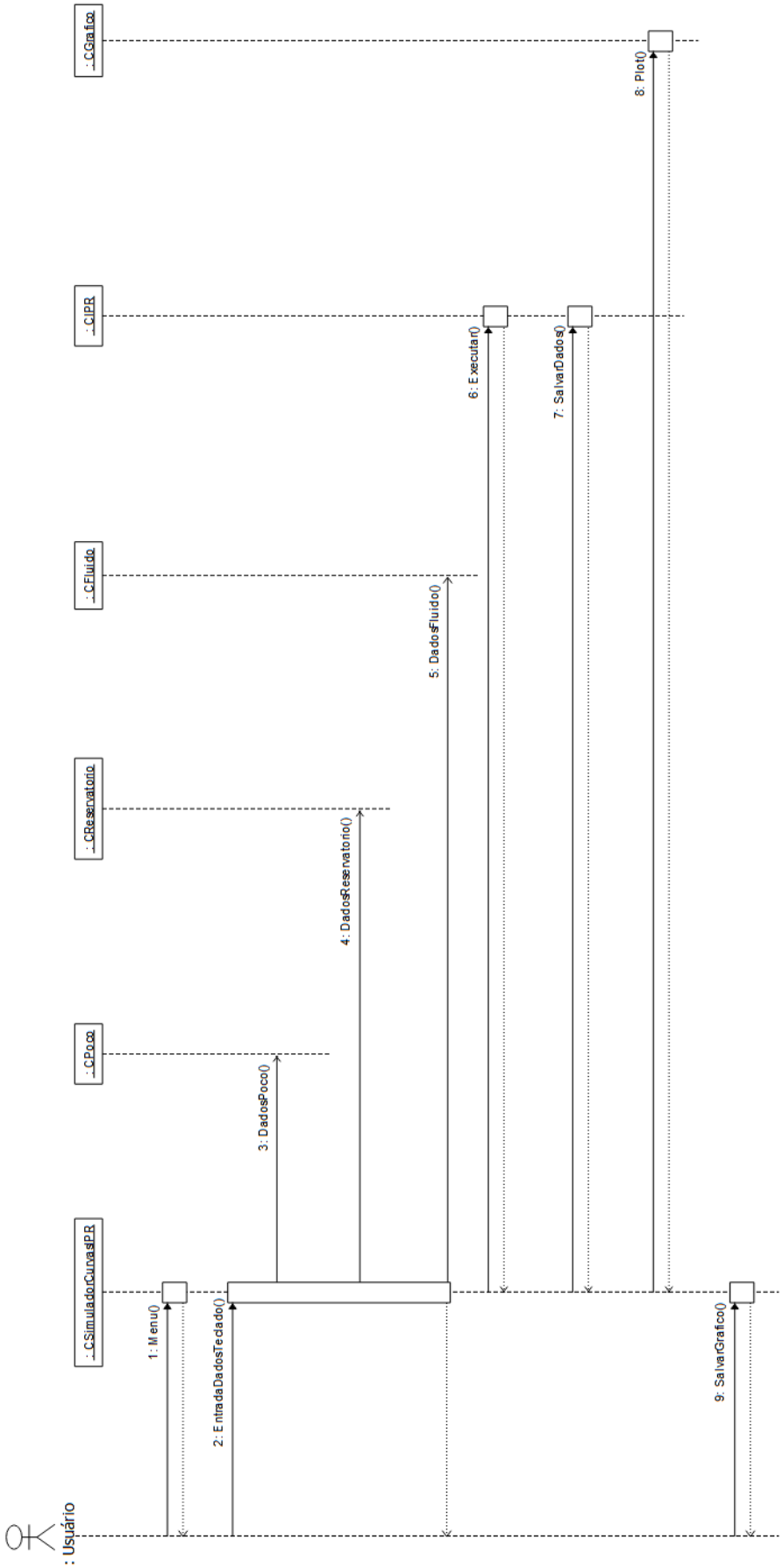

Figura 4.3: Diagrama de seqüência

Capítulo 5

Projeto

Neste capítulo do projeto de engenharia veremos questões associadas ao projeto do sistema, incluindo protocolos, recursos, plataformas suportadas, implicações nos diagramas feitos anteriormente, diagramas de componentes e implantação. Na segunda parte revisamos os diagramas levando em conta as decisões do projeto do sistema.

5.1 Projeto do sistema

Depois da análise orientada a objeto desenvolve-se o projeto do sistema, qual envolve etapas como a definição dos protocolos, da interface API, o uso de recursos, a subdivisão do sistema em subsistemas, a alocação dos subsistemas ao hardware e a seleção das estruturas de controle, a seleção das plataformas do sistema, das bibliotecas externas, dos padrões de projeto, além da tomada de decisões conceituais e políticas que formam a infraestrutura do projeto.

Deve-se definir padrões de documentação, padrões para o nome das classes, padrões de retorno e de parâmetros em métodos, características da interface do usuário e características de desempenho.

Segundo [**?**, **?**], o projeto do sistema é a estratégia de alto nível para resolver o problema e elaborar uma solução. Você deve se preocupar com itens como:

1. Protocolos

    - Definição dos protocolos de comunicação entre os diversos elementos externos
        - Neste projeto o software irá se comunicar com o componente externo Gnuplot.
    - Definição do formato dos arquivos gerados pelo software. Por exemplo: prefira formatos abertos, como arquivos txt e xml.
        - Neste projeto será gerado um arquivo .txt com os valores de pressão e vazão obtidos dos resultados.

  - Neste projeto será gerado um arquivo .png com as curvas IPR geradas.

2. Recursos

   - Identificação e alocação dos recursos globais, como os recursos do sistema serão alocados, utilizados, compartilhados e liberados. Implicam modificações no diagrama de componentes.
     - Neste projeto serão utilizados todos os componentes do computador que estão dentro do gabinete: HD, processador, memória;
     - Neste projeto serão utilizados todos os componentes :teclado, mouse etela.

3. Controle

   - Identificação da necessidade de otimização.
     - Neste projeto não haverá a necessidade de grande quantidade de memória visto que não haverá grande quantidade de dados.
   - Identificação de concorrências – quais algoritmos podem ser implementados usando processamento paralelo.
     - O software não necessitará de uma grande escala de processamento, logo, não haverá necessidade de processamento paralelo.

4. Plataformas

   - Identificação e definição das plataformas a serem suportadas: hardware, sistema operacional e linguagem de software.
     - Neste projeto será utilizado a linguagem de programação C++.
     - O software deverá ser multiplataforma, podendo ser executado em Windows e GNU/Linux.
     - O software será desenvolvido no sistema operacional Windows 11 em máquinas com processador Intel Core i5-11ª geração e Intel Core i3-10ª geração.
   - Seleção das bibliotecas externas a serem utilizadas.
     - Neste projeto será utilizada a biblioteca padrão da linguagem C++, incluindo vector, string, iostream, fstream, sstream e iremos utilizar a classe CGnuplot, que fornece acesso ao programa externo Gnuplot (link).
   - Seleção do ambiente de desenvolvimento para montar a interface de desenvolvimento – IDE.
     - Neste projeto a IDE utilizada será o software Embarcadero na versão 6.3 (link) e o DEV C++ na versão 5.11 (link)..

5.2 Projeto orientado a objeto – POO

O projeto orientado a objeto é a etapa posterior ao projeto do sistema. Baseia-se na análise, mas considera as decisões do projeto do sistema. Acrescenta a análise desenvolvida e as características da plataforma escolhida (hardware, sistema operacional e linguagem de software). Passa pelo maior detalhamento do funcionamento do software, acrescentando atributos e métodos que envolvem a solução de problemas específicos não identificados durante a análise.

Envolve a otimização da estrutura de dados e dos algoritmos, a minimização do tempo de execução, de memória e de custos. Existe um desvio de ênfase para os conceitos da plataforma selecionada.

Por exemplo, na análise você define que existe um método para salvar um arquivo em disco, define um atributo nomeDoArquivo, mas não se preocupa com detalhes específicos da linguagem. Já no projeto, você inclui as bibliotecas necessárias para acesso ao disco, cria um objeto específico para acessar o disco, podendo, portanto, acrescentar novas classes àquelas desenvolvidas na análise.

5.2.0.1 Efeitos do projeto no modelo estrutural

- Adicionar nos diagramas de pacotes as bibliotecas e subsistemas selecionados no projeto do sistema (exemplo: a biblioteca gráfica selecionada).
  - Neste projeto, foi utilizada a classe `CGnuplot` a fim de gerar as curvas para realização de análises gráficas. Para isto, é necessário que haja a instalação do software Gnuplot para que o funcionamento do Simulador seja totalmente garantido.
- Novas classes e associações oriundas das bibliotecas selecionadas e da linguagem escolhida devem ser acrescentadas ao modelo.
  - Neste projeto, foram feitas associações entre as classes CGnuplot e CSimuladorCurvasIPR para que ocorra a geração de gráficos dos parâmetros calculados;
  - As classes CRocha, CReservatorio, CPoco e CFluido estão associadas a CSimuladorCurvasIPR pois configuram os dados fundamentais que serão utilizados nos modelos empíricos;
  - Por fim, as classes CIPRVogel, CIPRGeneralizada,CIPRLinear e CIPRFetkovich também estão relacionadas a CSimuladorCurvasIPR visto que são os modelos empíricos que serão utilizados nos cálculos que resultarão nos parâmetros principais das curvas de IPR (pressão e vazão).
- Estabelecer as dependências e restrições associadas à plataforma escolhida.

  - O software utiliza o HD, processador e o teclado do computador;
  - Pode ser executado nas plataformas GNU/Linux, Windows ou Mac;
  - Nos Sistemas Operacionais citados, há necessidade de instalação do software Gnuplot para o funcionamento total do programa.

5.2.0.2 Efeitos do projeto no modelo dinâmico

- Revisar os diagramas de seqüência e de comunicação considerando a plataforma escolhida.
  - Os diagramas de sequência e comunicação serão revisados, se necessário, durante as etapas de desenvolvimento do código.
- Verificar a necessidade de se revisar, ampliar e adicionar novos diagramas de máquinas de estado e de atividades.
  - Os diagramas de máquina de estado e atividades sofrerão correções se surgir a necessidade à medida que o código for desenvolvido..

5.2.0.3 Efeitos do projeto nos atributos

- Atributos novos podem ser adicionados a uma classe, como, por exemplo, atributos específicos de uma determinada linguagem de software (acesso a disco, ponteiros, constantes e informações correlacionadas).
  - Neste projeto, os atributos fin e fout deverão ser criados a fim de possibilitar a leitura dos dados a partir de um arquivo de disco bem como criar um arquivo com a saída de dados.

5.2.0.4 Efeitos do projeto nos métodos

- Em função da plataforma escolhida, verifique as possíveis alterações nos métodos. O projeto do sistema costuma afetar os métodos de acesso aos diversos dispositivos (exemplo: hd, rede).
  - Neste projeto, além da possibilidade de entrada de dados por um arquivo de disco, estes poderão também ser digitados pelo usuário utilizando o teclado do computador de maneira a não se limitar somente na utilização de uma das vias.
- Revise os diagramas de classes, de sequência e de máquina de estado.

  - As classes deverão ser revisadas a medida que for notada a necessidade de adicionar uma ou mais classes durante o desenvolvimento do código. Atualmente, foram adicionadas as classes CRocha, CGeometria, CRadial, CLinear, CIPR, COleo e CGas, com o intuito solucionar uma maior gama de problemas relacionados a IPR de gás e para reservatórios de diferentes geometrias.

5.2.0.5 Efeitos do projeto nas heranças

- Reorganização das classes e dos métodos (criar métodos genéricos com parâmetros que nem sempre são necessários e englobam métodos existentes).

  - Foi criada uma classe-base CIPR para as classes CIPRLinear, CIPRVogel, CIPRFetkovich e CIPRGeneralizada, duas classes herdeiras da classe CFluido, COleo e CGas, e as classes foram reordenadas de modo que a classe CRocha seja uma classe-base para CReservatorio, essa seja uma classe-base para CGeometria e, por fim, essa seja uma classe-base para duas classes herdeiras, CRadial e CLinear.

5.2.0.6 Efeitos do projeto nas associações

- Deve-se definir na fase de projeto como as associações serão implementadas, se obedecerão um determinado padrão ou não.

  - O projeto, ao referir-se a classes e associações, sofrerá mudanças futuras quando o código começar a ser desenvolvido.

5.2.0.7 Efeitos do projeto nas otimizações

- A ordem de execução pode ser alterada.

  - Um menu com opções distintas irá aparecer na tela de modo que o usuário poderá escolher como gostaria de entrar com os dados, via teclado ou disco, o modelo empírico que gostaria de utilizar para realização dos cálculos de pressão e vazão. Logo após, o gráfico será gerado e o usuário poderá salvá-lo ou não.

As dependências das bibliotecas e arquivos são demonstrados pelo diagrama de componentes e as dependências entre o sistema e o hardware são ilustradas pelo diagrama de implantação.

5.3 Diagrama de componentes

O diagrama de componentes mostra a forma como os componentes do software se relacionam, suas dependências. Inclui itens como: componentes, subsistemas, executáveis,

nós, associações, dependências, generalizações, restrições e notas. Exemplos de componentes são bibliotecas estáticas, bibliotecas dinâmicas, dlls, componentes Java, executáveis, arquivos de disco, código-fonte.

Veja na Figura 5.1 o de diagrama de componentes. Dessa forma, um diagrama de componentes é uma ferramenta valiosa para o seu projeto de desenvolvimento de software para simulação de curvas IPR na engenharia de petróleo por várias razões:

- Visualiza a estrutura do sistema;
- Mostra as dependências entre os elementos;
- Organiza o código em módulos;
- Facilita a comunicação na equipe;
- Identifica áreas críticas do sistema;
- Serve como documentação.

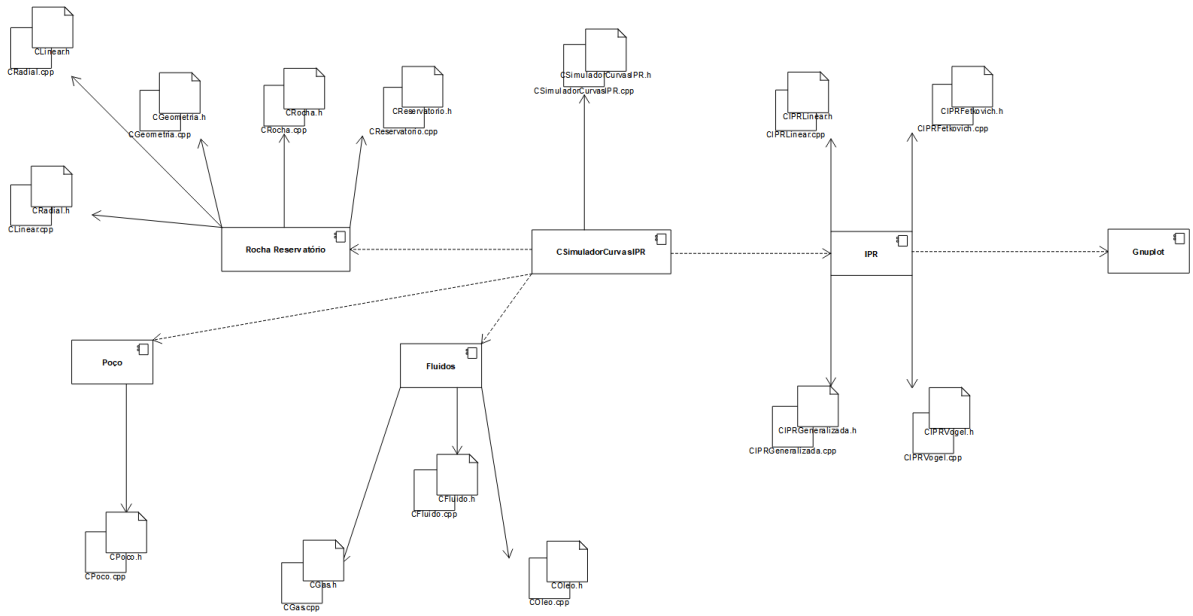

Figura 5.1: Diagrama de componentes

5.4 Diagrama de implantação

O diagrama de implantação é um diagrama de alto nível que inclui relações entre o sistema e o hardware e que se preocupa com os aspectos da arquitetura computacional escolhida. Seu enfoque é o hardware, a configuração dos nós em tempo de execução. O diagrama de implantação deve incluir os elementos necessários para que o sistema seja colocado em funcionamento: computador, periféricos, processadores, dispositivos, nós, relacionamentos de dependência, associação, componentes, subsistemas, restrições e

notas. O diagrama de implantação desempenha um papel crucial no projeto de engenharia de petróleo com simulações de curvas IPR para Poços Verticais. Ele se aplica de forma significativa ao ajudar a visualizar como os diversos componentes de software e hardware interagem em seu ambiente de execução. Neste contexto, o diagrama de implantação permite a:

- Visualização da Infraestrutura;
- Modelagem de Conexões;
- Planejamento de Recursos;
- Identificação de Pontos de Falha;
- Escalabilidade.

Veja na Figura 5.2 um exemplo de diagrama de implantação.

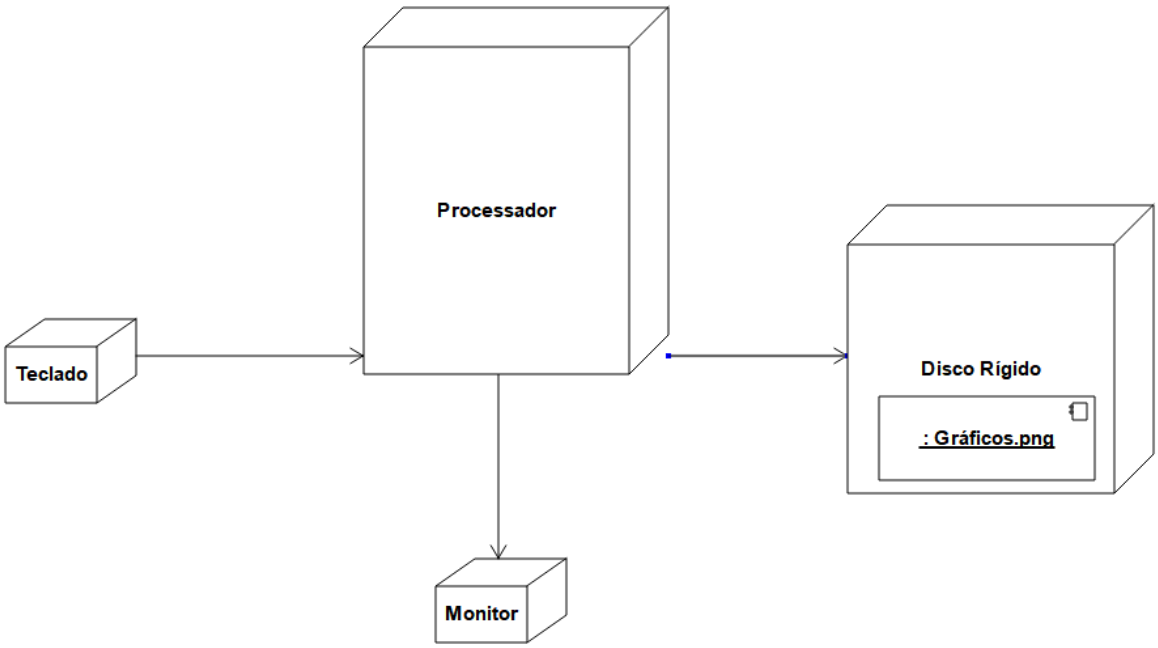

Figura 5.2: Diagrama de implantação

5.4.1 Lista de características <<features>>

No final do ciclo de concepção e análise chegamos a uma lista de características *<<features>>* que teremos de implementar.

Após a análises desenvolvidas e considerando o requisito de que este material deve ter um formato didático, chegamos a seguinte lista:

- v0.1
  - Lista de classes a serem implementadas: CFluido, COleo, CGas, CPoco, CRocha, CReservatorio, CGeometria, CLinear, CRadial, CIPR, CIPRLinear, CIPRVogel, CIPRFetkovich, CIPRGeneralizada, CSimuladorCurvasIPR e CGnuplot.
  - Testes:
    - O software deve ser capaz de exibir um menu com uma lista de opções ao usuário;
    - O software deve carregar informações via disco ou ler as informações inseridas pelo usuário via teclado;
    - O software deve disponibilizar uma opção para que o usuário selecione o modelo empírico a ser utilizado, desde que seja adequado para seu problema
    - O software deve, a partir dos dados inseridos, realizar cálculos iterativos de vazões e exportar os resultados para um arquivo externo;
    - O software deve plotar, utilizando a biblioteca externa Gnuplot, gráfico com a curva IPR do modelo selecionado;
    - O software deve oferecer ao usuário a possibilidade de salvar o gráfico gerado caso seja de necessidade do mesmo;
- v0.3
  - Lista de classes a serem implementadas
  - Testes
- v0.5
  - Lista de classes a serem implementadas
  - Testes

5.4.2 Tabela classificação sistema

A Tabela a seguir é utilizada para classificação do sistema desenvolvido. Deve ser preenchida na etapa de projeto e revisada no final, quando o software for entregue na sua versão final.

Licença:	[X] livre GPL-v3 [ ] proprietária
Engenharia de software:	[ ] tradicional [X] ágil [ ] outras
Paradigma de programação:	[ ] estruturada [X] orientado a objeto - POO [ ] funcional
Modelagem UML:	[X] básica [X] intermediária [ ] avançada
Algoritmos:	[X] alto nível [X] baixo nível
	implementação: [ ] recursivo ou [X] iterativo; [X] determinístico ou [ ] não-determinístico; [ ] exato ou [X] aproximado
	concorrências: [X] serial - síncrona [] concorrente [] paralelo
	paradigma: [X] dividir para conquistar [ ] programação linear [ ] transformação/ redução [ ] busca e enumeração [ ] heurístico e probabilístico [ ] baseados em pilhas
Software:	[ ] de base [X] aplicados [ ] de cunho geral [X] específicos para determinada área [X] educativo [X] científico
	instruções: [X] alto nível [ ] baixo nível
	otimização: [X] serial não otimizado [X] serial otimizado [] concorrente [] paralelo [ ] vetorial
	interface do usuário: [ ] kernel numérico [ ] linha de comando [ ] modo texto [X] híbrida (texto e saídas gráficas) [ ] modo gráfico (ex: Qt) [ ] navegador
Recursos de C++:	[X] C++ básico (FCC): variáveis padrões da linguagem, estruturas de controle e repetição, estruturas de dados, struct, classes(objetos, atributos, métodos), funções; entrada e saída de dados (*streams*), funções de cmath
	[X] C++ intermediário: funções lambda. Ponteiros, referências, herança, herança múltipla, polimorfismo, sobrecarga de funções e de operadores, tipos genéricos (templates), *smarth pointers*. Diretrizes de pré-processador, classes de armazenamento e modificadores de acesso. Estruturas de dados: enum, uniões. Bibliotecas: entrada e saída acesso com arquivos de disco, redirecionamento. Bibliotecas: *filesystem*
	[] C++ intermediário 2: A biblioteca de gabaritos de C++ (a STL), containers, iteradores, objetos funções e funções genéricas. Noções de processamento paralelo (múltiplas threads, uso de *thread, join* e *mutex*). Bibliotecas: *random, threads*

	[ ] C++ avançado: Conversão de tipos do usuário, especializações de templates, exceções. Cluster de computadores, processamento paralelo e concorrente, múltiplos processos (pipes, memória compartilhada, sinais). Bibliotecas: *expressões regulares, múltiplos processos*
Bibliotecas de C++:	[X] Entrada e saída de dados (*streams*) [X] cmath [X] *filesystem* [ ] *random* [X] *threads* [ ] *expressões regulares* [ ] *múltiplos processos [X] Vector [X] Locale [X] String*
Bibliotecas externas:	[X] `CGnuplot` [] `QCustomPlot` [ ] `Qt diálogos` [ ] QT Janelas/menus/BT_____
Ferramentas auxiliares:	Montador: [] make [] cmake [] qmake
IDE:	[ ] Editor simples: kate/gedit/emacs [ ] kdevelop [ ] QT-Creator [X] Embarcadero [X] Dev C++
SCV:	[ ] cvs [ ] svn [X] git
Disciplinas correlacionadas	[ ] estatística [ ] cálculo numérico [ ] modelamento numérico [ ] análise e processamento de imagens [X] elevação e escoamento [X] engenharia de reservatório

Capítulo 6

Ciclos Construção - Implementação

Neste capítulo, são apresentados os códigos fonte implementados.

Nota: os códigos devem ser documentados usando padrão **javadoc**. Posteriormente usar o programa **doxygen** para gerar a documentação no formato html.

- Veja informações gerais aqui http://www.doxygen.org/.
- Veja exemplo aqui http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html.

Nota: ao longo deste capítulo usamos inclusão direta de arquivos externos usando o pacote *listings* do LaTeX. Maiores detalhes de como a saída pode ser gerada estão disponíveis nos links abaixo.

- http://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings.
- http://mirrors.ctan.org/macros/latex/contrib/listings/listings.pdf.

6.1 Código fonte

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

Apresenta-se na listagem 6.1o arquivo com código da classe `CFluido`.

Listing 6.1: Arquivo de cabeçalho da classe CFluido

```
#ifndef CFluido_h
#define CFluido_h

#include <iostream>

class CFluido
{

        protected:
```

```
        double densidade; // Densidade do fluido
        double viscosidade; // Viscosidade do fluido
        double fatorVolumeFormacao; //Fator Volume formacao do fluido
        double compressibilidade; // Compressibilidade do fluido

        public:

                CFluido(){}; // Construtor
                virtual ~CFluido(){}; // Destrutor

                double Densidade() const {return densidade;} // Metodo
                   get para a densidade do fluido
                void Densidade (double _densidade) {densidade =
                   _densidade;} // Metodo set para a densidade do fluido

                double Viscosidade() const {return viscosidade;} //
                   Metodo get para a viscosidade do fluido
                void Viscosidade (double _viscosidade) {viscosidade =
                   _viscosidade;} // Metodo set para a viscosidade do
                   fluido

                double FatorVolumeFormacao() const {return
                   fatorVolumeFormacao;} // Metodo get para o fator
                   volume-formacao do oleo
                void FatorVolumeFormacao (double _fatorVolumeFormacao) {
                   fatorVolumeFormacao = _fatorVolumeFormacao;} //
                   Metodo set para o fator volume-formacao do oleo

        double getCompressibilidade() {return compressibilidade; } //
           Metodo get para a compressibilidade do oleo
        void setCompressibilidade(double _compressibilidade) {
           compressibilidade = _compressibilidade; } // Metodo set para
           a compressibilidade do oleo

};

#endif
```

Apresenta-se na listagem 6.2o arquivo com código da classe `COleo`.

Listing 6.2: Arquivo de cabeçalho da classe COleo

```
#ifndef COleo_h
#define COleo_h

#include "CFluido.h"

#include <iostream>
```

```
//Declaracao de COleo que herda de CFluido

class COleo : public CFluido {


        public:

                COleo(){}; // Construtor
                virtual ~COleo() = default; // Destrutor

};

#endif
```

Apresenta-se na listagem 6.3o arquivo com código da classe `CGas`.

Listing 6.3: Arquivo de cabeçalho da classe CGas

```
#ifndef CGas_h
#define CGas_h

#include "CFluido.h"

#include <iostream>

//Declaracao de CGas que herda de CFluido

class CGas : public CFluido {

        protected:

                double fatorZ; // Constante dos gases

        public:

                CGas(){}; // Construtor
                ~CGas(){}; // Destrutor

                double FatorZ() const {return fatorZ;} // Metodo get
                   para a constante dos gases
                void FatorZ (double _fatorZ) {fatorZ = _fatorZ;} //
                   Metodo set para a constante dos gases

};

#endif
```

Apresenta-se na listagem 6.4o arquivo com código da classe `CTipoFluido`.

Listing 6.4: Arquivo de cabeçalho da classe CTipoFluido

```
#ifndef CTipoFluido_h
#define CTipoFluido_h

#include <iostream>

enum class CTipoFluido {

        none = 0, oleo, gas // Definicao da geometria por enumeracao

        // O usuario fica aberto a adicionar mais geometrias

};

#endif
```

Apresenta-se na listagem 6.5o arquivo com código da classe `CRocha`.

Listing 6.5: Arquivo de cabeçalho da classe CRocha.

```
#ifndef CRocha_h
#define CRocha_h

#include <iostream>

// Classe CRocha que representa as propriedades de rocha

class CRocha {

        protected:

                double compRocha; // Compressibilidade da rocha
                double porosidade; // Porosidade da rocha
                double permeabilidade; // Permeabilidade da rocha


        public:

                CRocha(){}; // Construtor
                ~CRocha(){}; // Destrutor

                double CompRocha() const {return compRocha;} // Metodo
                    get para a compressibilidade da rocha
                void CompRocha (double _compRocha) {compRocha =
                    _compRocha;} // Metodo set para a compressibilidade
                    da rocha

                double Porosidade() const {return porosidade;} // Metodo
                     get para a porosidade da rocha
```

```
                void Porosidade (double _porosidade) {porosidade =
                    _porosidade;} // Metodo set para a porosidade da
                    rocha

                double Permeabilidade() const {return permeabilidade;}
                    // Metodo get para a permeabilidade da rocha
                void Permeabilidade (double _permeabilidade) {
                    permeabilidade = _permeabilidade;} // Metodo set para
                     a permeabilidade da rocha

};

#endif
```

Apresenta-se na listagem 6.26o arquivo com código da classe `CReservatorio`.

Listing 6.6: Arquivo de cabeçalho da classe CReservatorio

```
#ifndef CReservatorio_h
#define CReservatorio_h

// Inclusao dos arquivos de cabecalho

#include "CRocha.h"
#include "CGeometria.h"
#include "CLinear.h"
#include "CRadial.h"
#include "CTipoGeometria.h"
#include "CTipoFluido.h"
#include "CFluido.h"
#include "COleo.h"
#include "CGas.h"
#include "CPoco.h"

#include <iostream>

// Classe CReservatorio que representa o reservatorio

class CReservatorio: public CRocha {

        protected:

                double espessura; // Espessura do reservatorio
                double EulerCte = 1.781; // Constante de Euler
                double fatorPelicula; // Fator de pelicula do
                    reservatorio
                double pressaoBolha; // Pressao de bolha do reservatorio
                double pressaoInicial; // Pressao media inicial do
                    reservatorio
                double temperatura; // Temperatura
```

```
        double coeficienteDietz; // Coeficiente de Dietz
        double indiceProdutividade; // Indice de Produtividade
            do reservatorio
        CTipoGeometria enumeracaoGeometria; // Enumeracao para
            escolha do tipo de geometria
        CTipoFluido enumeracaoFluido; // Enumeracao para escolha
             do tipo de fluido
        CPoco poco; // Objeto de CPoco associado a reservatorio

public:

        CReservatorio () {} // Construtor default

        ~CReservatorio () {} // Destrutor

        double Espessura () const {return espessura;} // Metodo
            get para a espessura
        void Espessura (double _espessura) {espessura =
            _espessura;} // Metodo set para a espessura

        double FatorPelicula() const {return fatorPelicula;} //
            Metodo get para a fatorPelicula
        void FatorPelicula (double _fatorPelicula) {
            fatorPelicula = _fatorPelicula;} // Metodo set para a
             fatorPelicula

        double PressaoBolha() const {return pressaoBolha;} //
            Metodo get para a pressaoBolha
        void PressaoBolha (double _pressaoBolha) {pressaoBolha =
             _pressaoBolha;} // Metodo set para a pressaoBolha

        double PressaoInicial() const {return pressaoInicial;}
            // Metodo get para a pressaoInicial
        void PressaoInicial (double _pressaoInicial) {
            pressaoInicial = _pressaoInicial;} // Metodo set para
             a pressaoInicial

        double Temperatura() const {return temperatura;} //
            Metodo get para a temperatura
        void Temperatura (double _temperatura) {temperatura =
            _temperatura;} // Metodo set para a temperatura

        void setGeometria(CTipoGeometria _enumeracaoGeometria);
            // Metodo set para o tipo de geometria

        void setFluido(CTipoFluido _enumeracaoFluido); // Metodo
             set para o tipo de fluido
```

```
                void setFatorVolumeFormacao(double _fatorVolumeFormacao)
                    ; // Metodo set para o fator volume-formacao

                void CoeficienteDietz(CTipoGeometria
                    _enumeracaoGeometria); // Metodo para calculo do
                    Coeficiente de Dietz a ser utilizado
        double getCoeficienteDietz() { return coeficienteDietz; }

                void IndiceProdutividade(CFluido* _fluido, CGeometria*
                    _geometria, CPoco& _poco); // Metodo para calculo do
                    Indice de Produtividade
        double getIndiceProdutividade() {return indiceProdutividade; }

                double VazaoMaxima() {return ((indiceProdutividade *
                    pressaoInicial)/1.8);} // Metodo para calculo da
                    maxima vazao obtida pelo reservatorio (AOF)

};

#endif
```

Apresenta-se na listagem 6.7 o arquivo de implementação da classe `CReservatorio`.

Listing 6.7: Arquivo de implementação da classe CReservatorio

```
#include <iostream>
#include <fstream>
#include <string>
#include <cmath>

#include "CReservatorio.h"

using namespace std;

// Implementacao do metodo CoeficienteDietz da classe CReservatorio
void CReservatorio::CoeficienteDietz(CTipoGeometria _enumeracaoGeometria
    ) {

// Switch para atribuir o coeficiente de Dietz baseado no tipo de
    geometria
        switch (_enumeracaoGeometria) {

                case CTipoGeometria::linear:
                        coeficienteDietz = 30.8828;
                        break;

                case CTipoGeometria::radial:
                        coeficienteDietz = 31.62;
                        break;
```

```
                default:
                        cout << "Geometria␣nao␣encontrada";
                        break;

        }

}

// Implementacao do metodo IndiceProdutividade da classe CReservatorio
void CReservatorio::IndiceProdutividade(CFluido* _fluido, CGeometria*
    _geometria, CPoco& _poco) {

        // Calculo do Indice de Produtividade usando os parametros do
            reservatorio
        indiceProdutividade = ((permeabilidade * espessura) / (141.2 *
            _fluido->FatorVolumeFormacao() * _fluido->Viscosidade() *
            (0.5 * log((4 * _geometria->getArea()) / (EulerCte *
            coeficienteDietz * _poco.getRaioPoco() * _poco.getRaioPoco())
            ) + fatorPelicula)));

}
```

Apresenta-se na listagem 6.8o arquivo com código da classe `CGeometria`.

Listing 6.8: Arquivo de cabeçalho da classe CGeometria

```
#ifndef CGeometria_h
#define CGeometria_h

#include <iostream>

//Classe CGeometria que representa a geometria da rocha reservatorio

class CGeometria{

        protected:

                double area;

        public:

                CGeometria(){};
                virtual ~CGeometria() = default; //Destrutor

                virtual void Area() {}

                double getArea () {return area;} // Puxar area da
                    geometria em questÃ£o

};
```

```
#endif
```

Apresenta-se na listagem 6.9o arquivo com código da classe `CLinear`.

Listing 6.9: Arquivo de cabeçalho da classe CLinear

```
#ifndef CLinear_h
#define CLinear_h

#include <iostream>
#include <fstream>

#include "CGeometria.h"

class CLinear: public CGeometria {

        protected:

        double length; // Comprimento da rocha reservatorio
        double width; // Largura da rocha reservatorio

        public:

                CLinear(); // Construtor default
                ~CLinear() {}; // Destrutor

                double Length() {return length;} // Metodo get para o
                    comprimento do reservatorio
                void Length (double _length) {length = _length;} //
                    Metodo set para o comprimento do reservatorio

                double Width() const {return width;} // Metodo get para
                    a largura do reservatorio
                void Width (double _width) {width = _width;} // Metodo
                    set para a largura do reservatorio

                virtual void Area() override;

};

#endif
```

Apresenta-se na listagem 6.10 o arquivo de implementação da classe `CLinear`.

Listing 6.10: Arquivo de implementação da classe CLinear

```
#include <iostream>
#include <fstream>

#include "CLinear.h"
```

```
using namespace std;

CLinear::CLinear(): CGeometria() {}

void CLinear::Area() {

    area = width * length;

}
```

Apresenta-se na listagem 6.11o arquivo com código da classe `CRadial`.

Listing 6.11: Arquivo de cabeçalho da classe CRadial

```
#ifndef CRadial_h
#define CRadial_h

#include <iostream>
#include <cmath>
#include <fstream>

#include "CGeometria.h"

class CRadial: public CGeometria {

        protected:

                double raioExterno; // Raio da rocha reservatorio

        public:

                CRadial() {}; // Construtor default
                ~CRadial() {}; // Destrutor

                double RaioExterno() const {return raioExterno;} //
                   Metodo get para o raio do reservatorio
                void RaioExterno (double _raioExterno) {raioExterno =
                   _raioExterno;} // Metodo set para o raio do
                   reservatorio

                virtual void Area() override;

};

#endif
```

Apresenta-se na listagem 6.12 o arquivo de implementação da classe `CRadial`.

Listing 6.12: Arquivo de implementação da classe CRadial

```
#include <iostream>
#include <fstream>

#include <cmath>

#include "CRadial.h"

using namespace std;

void CRadial::Area() {

        area = M_PI*pow(raioExterno, 2.0);

}
```

Apresenta-se na listagem 6.13o arquivo com código da classe `CTipoGeometria`.

Listing 6.13: Arquivo de cabeçalho da classe CTipoGeometria

```
#ifndef CTipoGeometria_h
#define CTipoGeometria_h

#include <iostream>

enum class CTipoGeometria {

        none = 0, linear, radial // Definicao da geometria por
            enumeracao

        //O usuario fica aberto a adicionar mais geometrias

};

#endif
```

Apresenta-se na listagem 6.14o arquivo com código da classe `CPoco`.

Listing 6.14: Arquivo de cabeçalho da classe CPoco

```
#ifndef CPoco_h
#define CPoco_h

#include <iostream>

class CPoco{

        protected:

                double raioPoco; // Raio do poco
                double pwf; // Pressao de fundo
```

```
        double area; // Area do poco
        double vazaoProducao; // Vazao de producao

        public:

                CPoco(){}; // Construtor
                ~CPoco(){}; // Destrutor

                void CalcArea(); // Metodo de calculo da area
        double getArea() {return area; }

                double getRaioPoco() const {return raioPoco;} // Metodo
                   get para o raio do poco
                void setRaioPoco (double _raioPoco) {raioPoco =
                   _raioPoco;} // Metodo set para o raio do poco

                double getPressao() const {return pwf;} // Metodo get
                   para a pressao de fundo
                void setPressao (double _pwf) {pwf = _pwf;} // Metodo
                   set para a pressao de fundo

        double getVazaoProducao() {return vazaoProducao;} // Metodo get
           para a vazao de producao
        void setVazaoProducao(double _vazaoProducao) { vazaoProducao =
           _vazaoProducao;} // Metodo set para a vazao de producao

};
#endif
```

Apresenta-se na listagem 6.15 o arquivo de implementação da classe `CPoco`.

Listing 6.15: Arquivo de implementação da classe CPoco

```
#include "CPoco.h"
#include<iostream>
#include<cmath>

void CPoco::CalcArea(){

        area = (2.0 * M_PI * raioPoco * raioPoco) / 4.0; // CÃ¡lculo
           area do poco

};
```

Apresenta-se na listagem 6.16o arquivo com código da classe `CIPR`.

Listing 6.16: Arquivo de cabeçalho da classe CIPR

```
#ifndef CIPR_H
#define CIPR_H
```

```
#include "CFluido.h"
#include "CReservatorio.h"
#include "CPoco.h"
#include <vector>

class CIPR
{

public:

    CIPR();
    virtual void CalcIPR(CFluido* fluido, CReservatorio& reservatorio,
       CPoco& poco, CPoco& parametrosSegundoPoco);
    virtual void CalcIPR(CFluido* fluido, CReservatorio& reservatorio,
       CPoco& poco);
    std::vector<double> getPWF() {return valoresPwf;}
    void setPWF(std::vector<double> _valoresPwf) { valoresPwf =
       _valoresPwf; }
    std::vector<double> getVazao() {return vazao; }
    void iprVariacaoPwf(double _pwf);
    void setPartes(double _partes) { partes = _partes; }

protected:

    std::vector<double> valoresPwf;
    std::vector<double> vazao ;
    double partes;


};

#endif
```

Apresenta-se na listagem 6.17 o arquivo de implementação da classe `CIPR`.

Listing 6.17: Arquivo de implementação da classe CIPR

```
#include "CIPR.h"
#include <vector>

CIPR::CIPR()
{
}

// Declarando e definindo o metodo CalcIPR na classe CIPR
void CIPR::CalcIPR(CFluido* fluido, CReservatorio& reservatorio, CPoco&
   poco, CPoco& parametrosSegundoPoco) {

}
```

```
void CIPR::CalcIPR(CFluido *fluido, CReservatorio &reservatorio, CPoco &
    poco)
{
}

void CIPR::iprVariacaoPwf(double _pwf) // Funcao responsavel pela
    variacao de Pwf
{

  double deltaP = _pwf/partes; // seta o Delta P que eh obtido pelo
     usuario. Divisao de Pwf e q sao obtidas pelo usuario

  while ( _pwf >= 0)
  {
      valoresPwf.push_back(_pwf); //Pwf armazenado eh decrementado em
         Delta P
      _pwf -= deltaP;

   if(_pwf < 0)
      valoresPwf.push_back(0.0);

  }

}
```

Apresenta-se na listagem 6.18o arquivo com código da classe `CIPRLinear`.

Listing 6.18: Arquivo de cabeçalho da classe CIPRLinear

```
#ifndef MODELOIPRLINEAR_H
#define MODELOIPRLINEAR_H

#include "CIPR.h"

class CIPRLinear : public CIPR
{

public:
    CIPRLinear(){}; // Construtor padrao
    void CalcIPR(CFluido* fluido, CReservatorio& reservatorio, CPoco&
       poco) override;

};

#endif
```

Apresenta-se na listagem 6.19 o arquivo de implementação da classe `CIPRLinear`.

Listing 6.19: Arquivo de implementação da classe CIPRLinear

```
#include "CIPRLinear.h"
```

```
#include <iostream>
#include <cmath>

void CIPRLinear::CalcIPR(CFluido* fluido, CReservatorio& reservatorio,
    CPoco& poco) {

    double j = reservatorio.getIndiceProdutividade();
    double pi = reservatorio.PressaoInicial();
    double pb = reservatorio.PressaoBolha();

    iprVariacaoPwf(pi);  // Inicializa a classe CIPRVariacaoPwf com
        parametro pi

        // Uma estrutura condicional verifica se a pressao inicial eh
            maior que a pressao de bolha
    if (pi > pb) {
        // Loop para calcular a vazao para cada valor de Pwf
        for (double pwf : valoresPwf) {
            vazao.push_back(j * (pi - pwf));

            std::cout << "Pwf:␣" << pwf << ",␣Vazao:␣" << j * (pi - pwf)
                << std::endl;
        }
    } else {
        // Mostrar mensagem de erro e encerrar o codigo
        std::cout << "A␣pressao␣do␣reservatorio␣(pi)␣eh␣menor␣ou␣igual␣a
            ␣pressao␣de␣bolha␣(pb).␣Selecione␣um␣valor␣acima␣de␣pb." <<
            std::endl;
        exit(1); // Encerra o programa com um codigo de erro
    }

}
```

Apresenta-se na listagem 6.20o arquivo com código da classe `CIPRFetkovich`.

Listing 6.20: Arquivo de cabeçalho da classe CIPRFetkovich

```
#ifndef CFetkovich_h
#define CFetkovich_h

#include "CIPR.h"

class CIPRFetkovich : public CIPR{
public:
    CIPRFetkovich(){};
    virtual void CalcIPR(CFluido* fluido, CReservatorio& reservatorio,
        CPoco& poco, CPoco& pocoDois) override;

};
```

```
#endif
```

Apresenta-se na listagem 6.21 o arquivo de implementação da classe `CIPRFetkovich`.

Listing 6.21: Arquivo de implementação da classe CIPRFetkovich

```
#include "CIPRFetkovich.h"
#include <cmath>
#include <iostream>

using namespace std;

void CIPRFetkovich::CalcIPR(CFluido* fluido, CReservatorio& reservatorio
    , CPoco& poco, CPoco& pocoDois)
{

    double pi = reservatorio.PressaoInicial();
        double pwf = pi; // Inicializa o valor de pwf como pi

    // Solicita os valores de teste do usuario
    double pwf1, pwf2, q1, q2;


    pwf1 = poco.getPressao();


    pwf2 = pocoDois.getPressao();


    q1 = poco.getVazaoProducao();

    q2 = pocoDois.getVazaoProducao();

        //Definicao das constantes empiricas n e C
    double n = ( log(q1/q2) ) / ( log ( ( pow (pi,2) - pow(pwf1,2) )/ (
       pow(pi,2) - pow(pwf2,2) ) ) );
    double C = q1 / pow( (pow(pi,2) - pow(pwf1,2)), n);

        // Calculo de qmax com os valores de teste
    double qmax = C * pow(pi, 2*n);


    iprVariacaoPwf(pwf);

    // Loop para variar pwf
    for (double pwf : valoresPwf) {
        if (pwf == pi) {
            vazao.push_back(0); // Vazao zero quando pwf eh igual a pi
        } else if (pwf == 0) {
```

```
            vazao.push_back(qmax); // Vazao maxima quando pwf eh igual a
                zero
        } else {
            vazao.push_back(qmax * (1 - pow(pwf/pi, 2 ) ));
        }
        cout << "Pwf:␣" << pwf << ",␣Vazao:␣" << qmax * (1 - pow(pwf/pi,
            2 ) ) << endl;
    }
}
```

Apresenta-se na listagem 6.22o arquivo com código da classe `CIPRVogel`.

Listing 6.22: Arquivo de cabeçalho da classe CIPRVogel

```
#ifndef CVogel_H
#define CVogel_H

#include "CFluido.h"
#include "CReservatorio.h"
#include "CPoco.h"
#include "CIPR.h"

class CIPRVogel : public CIPR {
public:
    CIPRVogel(){};
    virtual void CalcIPR(CFluido* fluido, CReservatorio& reservatorio,
       CPoco& poco, CPoco& parametrosSegundoPoco) override;

};

#endif
```

Apresenta-se na listagem 6.23 o arquivo de implementação da classe `CIPRVogel`.

Listing 6.23: Arquivo de implementação da classe CIPRVogel

```
#include "CIPRVogel.h"
#include <cmath>
#include <iostream>

using namespace std;


void CIPRVogel::CalcIPR(CFluido* fluido, CReservatorio& reservatorio,
   CPoco& poco, CPoco& parametrosSegundoPoco) {
    double pi = reservatorio.PressaoInicial(); // Pressao inicial do
       reservatorio
    double j = reservatorio.getIndiceProdutividade(); //Indice de
       Produtividade

    // Solicita os valores de teste do usuario
```

```
    double pwf1, pwf2, q1, q2;

    // Calculo de qmax com os valores de teste
    double qmax = j*(pi)/1.8;

    // Inicializa a classe CIPRVariacaoPwf
    iprVariacaoPwf(pi);

    // Loop para variar pwf
    for (double pwf : valoresPwf) {
        double q;
        if (pwf == pi) {
            vazao.push_back(0); // Vazao zero quando pwf eh igual a pi
        } else if (pwf == 0) {
            vazao.push_back(qmax); // Vazao maxima quando pwf eh igual a
                zero
        } else {
            vazao.push_back(qmax * (1 - 0.2 * (pwf / pi) - 0.8 * pow(pwf
                / pi, 2)));
        }
    }

}
```

Apresenta-se na listagem 6.24o arquivo com código da classe `CIPRGeneralizada`.

Listing 6.24: Arquivo de cabeçalho da classe CIPRGeneralizada

```
#ifndef CIPRGeneralizada_H
#define CIPRGeneralizada_H

#include "CIPR.h"

class CIPRGeneralizada : public CIPR {
public:
    CIPRGeneralizada(){};
    virtual void CalcIPR(CFluido* fluido, CReservatorio& reservatorio,
       CPoco& poco);

};

#endif // MODELOIPRGENERALIZADA_H
```

Apresenta-se na listagem 6.25 o arquivo de implementação da classe `CIPRGeneralizada`.

Listing 6.25: Arquivo de implementação da classe CIPRGeneralizada

```
#include "CIPRGeneralizada.h"
#include<iostream>
#include <cmath>
```

```
void CIPRGeneralizada::CalcIPR(CFluido* fluido, CReservatorio&
    reservatorio, CPoco& poco) {
     double j = reservatorio.getIndiceProdutividade();
     double pi = reservatorio.PressaoInicial();
     double pb = reservatorio.PressaoBolha();

     // Inicializa a classe CIPRVariacaoPwf
     iprVariacaoPwf(pi);

     if (pi <= pb) {
         // Loop para calcular a vazao para cada valor de Pwf
         for (double pwf : valoresPwf) {
             vazao.push_back(j * (pi - pb) + ((j * pb) / 1.8) * (1.0 -
                0.2 * (pwf / pb) - 0.8 * ((pwf / pb) * (pwf / pb))));

             std::cout << "Pwf:␣" << pwf << ",␣Vazao:␣" << j * (pi - pwf)
                 << std::endl;
         }
     } else {
         // Mostrar mensagem de erro e encerrar o codigo
         for (double pwf : valoresPwf) {
             vazao.push_back(j * (pi - pwf));

             std::cout << "Pwf:␣" << pwf << ",␣Vazao:␣" << j * (pi - pwf)
                 << std::endl;
         }
     }

}
```

Apresenta-se na listagem **??**o arquivo com código da classe `CMetodo`.

Listing 6.26: Arquivo de cabeçalho da classe CReservatorio

```
#ifndef CReservatorio_h
#define CReservatorio_h

// Inclusao dos arquivos de cabecalho

#include "CRocha.h"
#include "CGeometria.h"
#include "CLinear.h"
#include "CRadial.h"
#include "CTipoGeometria.h"
#include "CTipoFluido.h"
#include "CFluido.h"
#include "COleo.h"
#include "CGas.h"
#include "CPoco.h"
```

```
#include <iostream>

// Classe CReservatorio que representa o reservatorio

class CReservatorio: public CRocha {

        protected:

                double espessura; // Espessura do reservatorio
                double EulerCte = 1.781; // Constante de Euler
                double fatorPelicula; // Fator de pelicula do
                   reservatorio
                double pressaoBolha; // Pressao de bolha do reservatorio
                double pressaoInicial; // Pressao media inicial do
                   reservatorio
                double temperatura; // Temperatura
                double coeficienteDietz; // Coeficiente de Dietz
                double indiceProdutividade; // Indice de Produtividade
                   do reservatorio
                CTipoGeometria enumeracaoGeometria; // Enumeracao para
                   escolha do tipo de geometria
                CTipoFluido enumeracaoFluido; // Enumeracao para escolha
                    do tipo de fluido
                CPoco poco; // Objeto de CPoco associado a reservatorio

        public:

                CReservatorio () {} // Construtor default

                ~CReservatorio () {} // Destrutor

                double Espessura () const {return espessura;} // Metodo
                   get para a espessura
                void Espessura (double _espessura) {espessura =
                   _espessura;} // Metodo set para a espessura

                double FatorPelicula() const {return fatorPelicula;} //
                   Metodo get para a fatorPelicula
                void FatorPelicula (double _fatorPelicula) {
                   fatorPelicula = _fatorPelicula;} // Metodo set para a
                    fatorPelicula

                double PressaoBolha() const {return pressaoBolha;} //
                   Metodo get para a pressaoBolha
                void PressaoBolha (double _pressaoBolha) {pressaoBolha =
                    _pressaoBolha;} // Metodo set para a pressaoBolha

                double PressaoInicial() const {return pressaoInicial;}
```

```
                // Metodo get para a pressaoInicial
            void PressaoInicial (double _pressaoInicial) {
                pressaoInicial = _pressaoInicial;} // Metodo set para
                 a pressaoInicial

            double Temperatura() const {return temperatura;} //
                Metodo get para a temperatura
            void Temperatura (double _temperatura) {temperatura =
                _temperatura;} // Metodo set para a temperatura

            void setGeometria(CTipoGeometria _enumeracaoGeometria);
                // Metodo set para o tipo de geometria

            void setFluido(CTipoFluido _enumeracaoFluido); // Metodo
                 set para o tipo de fluido

            void setFatorVolumeFormacao(double _fatorVolumeFormacao)
                ; // Metodo set para o fator volume-formacao

            void CoeficienteDietz(CTipoGeometria
                _enumeracaoGeometria); // Metodo para calculo do
                Coeficiente de Dietz a ser utilizado
    double getCoeficienteDietz() { return coeficienteDietz; }

            void IndiceProdutividade(CFluido* _fluido, CGeometria*
                _geometria, CPoco& _poco); // Metodo para calculo do
                Indice de Produtividade
    double getIndiceProdutividade() {return indiceProdutividade; }

            double VazaoMaxima() {return ((indiceProdutividade *
                pressaoInicial)/1.8);} // Metodo para calculo da
                maxima vazao obtida pelo reservatorio (AOF)

};

#endif
```

Apresenta-se na listagem 6.27o arquivo com código da classe `CSimuladorCurvasIPR`.

Listing 6.27: Arquivo de cabeçalho da classe CSimuladorCurvasIPR

```
#ifndef CSimuladorCurvasIPR_h
#define CSimuladorCurvasIPR_h

#include <iostream>
#include <string>
#include <vector>

#include "CPoco.h"
#include "CFluido.h"
```

```
#include "COleo.h"
#include "CGas.h"
#include "CGeometria.h"
#include "CLinear.h"
#include "CRadial.h"
#include "CReservatorio.h"
#include "CRocha.h"
#include "CIPRLinear.h"
#include "CIPRFetkovich.h"
#include "CIPRVogel.h"
#include "CIPRGeneralizada.h"
#include "CIPR.h"
#include "CGnuplot.h"
#include "CMetodo.h"
#include "CTipoGeometria.h"

class CSimuladorCurvasIPR {

        private:

        CReservatorio reservatorio; // Criando objeto reservatorio
        CPoco poco; // Criando objeto poco
        CPoco pocoDois; // Criando segundo objeto poco para IPR do tipo
           Fetkovich
        CIPR* IPR; // Ponteiro para apontar para o tipo de metodo
        CGeometria* forma; // Ponteiro para apontar a geometria do
           reservatorio
        CFluido* fluido; // Ponteiro para apontar o tipo de fluido: gas
           ou oleo
        Gnuplot plot; // Criando objeto plot
        CMetodo metodo; // Criando objeto para enumeracao do metodo

        std::vector<double> vazoes; // Criando vetor de vazoes
        std::vector<double> pressoes; // Criando vetor de pressoes de
           fundo

        public:

                CSimuladorCurvasIPR(){}; // Construtor
                ~CSimuladorCurvasIPR(); // Destrutor

                void EntradaDados(); // Metodo para entrada de dados do
                   usuario

        void Calculos(); // Metodo para calculos da simulacao

                void Plotar(); // Metodo para plotar as curvas
```

```
                void SalvarGrafico(); // Metodo para salvar o grafico

                void Executar(); // Metodo para execucao do programa

};

#endif
```

Apresenta-se na listagem 6.28 o arquivo de implementação da classe `CSimuladorCurvasIPR`.

Listing 6.28: Arquivo de implementação da classe CSimuladorCurvasIPR

```
#include "CSimuladorCurvasIPR.h"

#include <iostream>

using namespace std;

// Sobrecarga de operador >> para o metodos

istream &operator>>(istream &is, CMetodo &metodo) {

   unsigned int a;
   is >> a;
   metodo = static_cast<CMetodo>(a);

   return is;

}

// Sobrecarga de operador >> para o tipo de fluido

istream &operator>>(istream &is, CTipoFluido &fluido) {

   unsigned int a;
   is >> a;
   fluido = static_cast<CTipoFluido>(a);

   return is;

}

// Sobrecarga de operador >> para geometria do reservatorio
istream &operator>>(istream &is, CTipoGeometria &geometria) {

   unsigned int a;
   is >> a;
   geometria = static_cast<CTipoGeometria>(a);

   return is;
```

```
}

CSimuladorCurvasIPR::~CSimuladorCurvasIPR() {

    delete forma;
    forma = nullptr;

}

void CSimuladorCurvasIPR::EntradaDados() {

    cout << endl;
    cout << "---------------␣SIMULADOR␣DE␣CURVAS␣IPR␣PARA␣POCOS␣
       VERTICAIS␣---------------" << endl;
    cout << "
       ------------------------------------------------------------------------
       " << endl;
    cout << "------------␣Atila␣Junior,␣Giovanna␣Massardi␣e␣Marcelo␣
       Bernardo␣------------" << endl;
    cout << "--------------------------------␣UENF/LENEP␣
       --------------------------------" << endl;
    cout << endl;

    cout << "Aperte␣Enter␣para␣prosseguir..." << endl;

    cin.get();


    cout << "Qual␣o␣metodo␣sera␣utilizado?" << endl;
    cout << "1␣-␣IPR␣Linear" << endl;
    cout << "2␣-␣IPR␣Generalizada" << endl;
    cout << "3␣-␣IPR␣de␣Fetkovich␣(Ideal␣para␣reservatorios␣de␣gas)" <<
       endl;
    cout << "4␣-␣IPR␣de␣Vogel" << endl;
    cout << "
       ------------------------------------------------------------------------
       " << endl;

    cin >> metodo; // Recebe o tipo de metodo que o usuario entrar

    cin.get();

        // Switch para o tipo de metodo de calculo
    switch (metodo) {

        case CMetodo::IPR_LINEAR:
```

```
    {

        CTipoGeometria geometria = CTipoGeometria::linear;
cin.get();

forma = new CLinear;
CLinear* cast = dynamic_cast<CLinear*>(forma);

reservatorio.CoeficienteDietz(geometria);

cout << "Entre␣com␣o␣comprimento␣do␣reservatorio␣(FT)" <<
   endl;

double esp;

cin >> esp;

cout << "Entre␣com␣a␣largura␣do␣reservatorio␣(FT)" << endl;

double lar;

cin >> lar;

cast-> Length(esp);
cast-> Width(lar);
cast-> Area();

cout << "Entre␣com␣o␣raio␣do␣poco␣(FT):" << endl;

double raio;

cin >> raio;

poco.setRaioPoco(raio);

cout << "Entre␣com␣a␣pressao␣inicial␣do␣reservatorio␣(PSIA):
   ␣" << endl;

double pi;

cin >> pi;

reservatorio.PressaoInicial(pi);

cout << "Entre␣com␣a␣pressao␣de␣bolha␣do␣reservatorio␣(PSIA)
   :␣" << endl;

double pb;
```

```
cin >> pb;

reservatorio.PressaoBolha(pb);

poco.CalcArea();

cout << "Qual␣tipo␣de␣fluido␣presente␣no␣reservatorio?" <<
   std::endl;
cout << "1␣-␣Oleo" << endl;
cout << "2␣-␣Gas" << endl;
cout << "
   ------------------------------------------------------------------
   " << endl;

CTipoFluido tipoFluido;

cin >> tipoFluido;
cin.get();

bool chs = true;

while(chs)

            {

                    switch (tipoFluido) // Switch para
                       definicao do tipo de fluido

                    {

                            case CTipoFluido::oleo: // Caso
                               oleo, cria um novo objeto da
                               classe COleo

                                    {

            fluido = new COleo;
            cout << "Entre␣com␣fator␣volume␣de␣formacao␣do␣
               oleo:␣" << endl;
            double Bo;
            cin >> Bo;
            fluido->FatorVolumeFormacao(Bo);
            cout << "Entre␣com␣a␣viscosidade␣do␣fluido␣(CP):
               " << endl;
            double viscosidade;
            cin >> viscosidade;
            fluido->Viscosidade(viscosidade);
```

```
                chs = false;
                break;

                                    }

                                case CTipoFluido::gas: // Caso
                                    gas, cria um novo objeto da
                                    classe CGas

                                    {

                fluido = new CGas;
                cout << "Entre com fator volume de formacao do
                    gas:" << endl;
                double Bg;
                cin >> Bg;
                fluido->FatorVolumeFormacao(Bg);
                cout << "Entre com a viscosidade do fluido (CP):
                    " << endl;
                double viscosidade;
                cin >> viscosidade;
                fluido->Viscosidade(viscosidade);

                chs = false;
                break;

                                    }

                                default:
                cout << "Entrada invalida!" << endl;
                cout << "Qual tipo de fluido presente no
                    reservatorio? " << endl;
                cin >> tipoFluido;
                cin.get();

                        }

        }

        cout << "Entre com a espessura do reservatorio (FT):" <<
            endl;

        double espreservatorio;

        cin >> espreservatorio;

        reservatorio.Espessura(espreservatorio);
```

```
cout << "Entre␣com␣a␣permeabilidade␣da␣rocha␣reservatorio␣(
   MD):" << endl;

double perm;

cin >> perm;

reservatorio.Permeabilidade(perm);

cout << "Entre␣com␣fator␣de␣pelicula␣do␣reservatorio:" <<
   endl;

double s;

cin >> s;

reservatorio.FatorPelicula(s);

reservatorio.IndiceProdutividade(fluido, forma, poco);

cout << "IP␣=␣" << reservatorio.getIndiceProdutividade() <<
   endl << endl;

IPR = new CIPRLinear();

break;

    }

    case CMetodo::IPR_GENERALIZADA:

    {

            CTipoGeometria geometria = CTipoGeometria::
               radial; // Caso geometria radial cria objeto
               do tipo CRadial
cin.get();

forma = new CRadial;
CRadial* cast = dynamic_cast<CRadial*>(forma);

reservatorio.CoeficienteDietz(geometria);

cout << "Entre␣com␣o␣raio␣externo␣do␣reservatorio␣(FT)" <<
   endl;

double Re;
```

```
cin >> Re;

cast->RaioExterno(Re);
cast->Area();

cout << "Entre␣com␣a␣raio␣do␣poco␣(FT)" << endl;

double raio;

cin >> raio;

poco.setRaioPoco(raio);

cout << "Entre␣com␣a␣pressao␣inicial␣(PSIA):" << endl;

double pi;

cin >> pi;

reservatorio.PressaoInicial(pi);

cout << "Entre␣com␣a␣pressao␣de␣bolha␣(PSIA):␣" << endl;

double pb;

cin >> pb;

reservatorio.PressaoBolha(pb);

poco.CalcArea();

cout << "Qual␣tipo␣de␣fluido␣presente␣no␣reservatorio?" <<
   endl;
cout << "1␣-␣Oleo" << endl;
cout << "2␣-␣Gas" << endl;
cout << "
   ------------------------------------------------------------------
   " << endl;

CTipoFluido tipoFluido;

cin >> tipoFluido;
cin.get();

bool chs = true;

while(chs)
```

```
{

        switch (tipoFluido)

        {

                case CTipoFluido::oleo:

                        {

fluido = new COleo;
cout << "Entre␣com␣fator␣volume␣de␣formacao␣do␣
   oleo:" << endl;
double Bo;
cin >> Bo;
fluido->FatorVolumeFormacao(Bo);
cout << "Entre␣com␣a␣viscosidade␣do␣fluido␣(CP):
   " << endl;
double viscosidade;
cin >> viscosidade;
fluido->Viscosidade(viscosidade);

chs = false;
break;

                        }

                case CTipoFluido::gas:

                        {

fluido = new CGas;
cout << "Entre␣com␣fator␣volume␣de␣formacao␣do␣
   gas:" << endl;
double Bg;
cin >> Bg;
fluido->FatorVolumeFormacao(Bg);
cout << "Entre␣com␣a␣viscosidade␣do␣fluido␣(CP):
   " << endl;
double viscosidade;
cin >> viscosidade;
fluido->Viscosidade(viscosidade);

chs = false;
break;

                        }
```

```
                                default:
                    cout << "Entrada␣invalida!" << endl;
                    cout << "Qual␣tipo␣de␣fluido␣presente␣no␣
                       reservatorio?" << endl;
                    cin >> tipoFluido;
                    cin.get();

                            }

        }

        cout << "Entre␣com␣a␣espessura␣do␣reservatorio␣(FT):" <<
           endl;

        double espreservatorio;

        cin >> espreservatorio;

        reservatorio.Espessura(espreservatorio);

        cout << "Entre␣com␣a␣permeabilidade␣da␣rocha␣(MD):" << endl;

        double perm;

        cin >> perm;

        reservatorio.Permeabilidade(perm);

        cout << "Entre␣com␣fator␣de␣pelicula␣do␣reservatorio:" <<
           endl;

        double s;

        cin >> s;

        reservatorio.FatorPelicula(s);

        reservatorio.IndiceProdutividade(fluido, forma, poco);

        IPR = new CIPRGeneralizada();

        break;

            }

            case CMetodo::IPR_FETKOVICH:
```

```
            {

            CTipoGeometria geometria = CTipoGeometria::
               radial;
cin.get();

forma = new CRadial;
CRadial* cast = dynamic_cast<CRadial*>(forma);

reservatorio.CoeficienteDietz(geometria);

cout << "Entre␣com␣o␣raio␣externo␣do␣reservatorio␣(FT)" <<
   endl;

double Re;

cin >> Re;

cast->RaioExterno(Re);
cast->Area();

cout << "Entre␣com␣a␣raio␣do␣poco␣." << endl;

double raio;

cin >> raio;

poco.setRaioPoco(raio);

cout << "Entre␣com␣a␣pressao␣media␣do␣reservatorio␣(PSIA):␣"
    << endl;

double pi;

cin >> pi;

reservatorio.PressaoInicial(pi);

cout << "Entre␣com␣a␣pressao␣de␣bolha␣(PSIA):␣" << endl;

double pb;

cin >> pb;

reservatorio.PressaoBolha(pb);


cout << "Entre␣com␣a␣pressao␣de␣fundo␣no␣poco␣A␣(PSIA):" <<
```

```
    endl;

double pwf1;

cin >> pwf1;

poco.setPressao(pwf1);

cout << "Entre␣com␣a␣vazao␣no␣poco␣A␣(STB/d):" << endl;

double q1;

cin >> q1;

poco.setVazaoProducao(q1);

cout << "Entre␣com␣a␣pressao␣de␣fundo␣no␣poco␣B␣(PSIA):" <<
    endl;

double pwf2;

cin >> pwf2;

pocoDois.setPressao(pwf2);

cout << "Entre␣com␣a␣vazao␣no␣poco␣B␣(STB/d):" << endl;

double q2;

cin >> q2;

pocoDois.setVazaoProducao(q2);

poco.CalcArea();

cout << "Qual␣tipo␣de␣fluido␣presente␣no␣reservatorio?" <<
    endl;
cout << "1␣-␣Oleo" << endl;
cout << "2␣-␣Gas" << endl;
cout << "
    ------------------------------------------------------------------
    " << endl;

CTipoFluido tipoFluido;

cin >> tipoFluido;
cin.get();
```

```
bool chs = true;

while(chs)

            {

                    switch (tipoFluido)

                    {
        case CTipoFluido::oleo:

                                    {

            fluido = new COleo;
            cout << "Entre␣com␣fator␣volume␣de␣formacao␣do␣
               oleo:" << endl;
            double Bo;
            cin >> Bo;
            fluido->FatorVolumeFormacao(Bo);
            cout << "Entre␣com␣a␣viscosidade␣do␣fluido␣(CP):
               " << endl;
            double viscosidade;
            cin >> viscosidade;
            fluido->Viscosidade(viscosidade);

            chs = false;
            break;

                                    }

                        case CTipoFluido::gas:

                                    {

            fluido = new CGas;
            cout << "Entre␣com␣fator␣volume␣de␣formacao␣do␣
               gas:" << endl;
            double Bg;
            cin >> Bg;
            fluido->FatorVolumeFormacao(Bg);
            cout << "Entre␣com␣a␣viscosidade␣do␣fluido␣(CP):
               " << endl;
            double viscosidade;
            cin >> viscosidade;
            fluido->Viscosidade(viscosidade);

            chs = false;
            break;
```

```
                                        }

                                default:
                    cout << "Entrada␣invalida!" << endl;
                    cout << "Qual␣tipo␣de␣fluido␣presente␣no␣
                       reservatorio?" << endl;
                    cin >> tipoFluido;
                    cin.get();

                            }

}

cout << "Entre␣com␣a␣espessura␣do␣reservatorio␣(FT):" <<
   endl;

double espreservatorio;

cin >> espreservatorio;

reservatorio.Espessura(espreservatorio);

cout << "Entre␣com␣a␣permeabilidade␣da␣rocha␣reservatorio␣(
   MD):" << endl;

double perm;

cin >> perm;

reservatorio.Permeabilidade(perm);

cout << "Entre␣com␣fator␣de␣pelicula␣do␣reservatorio:" <<
   endl;

double s;

cin >> s;

reservatorio.FatorPelicula(s);

reservatorio.IndiceProdutividade(fluido, forma, poco);

IPR = new CIPRFetkovich();

break;

    }
```

```
    case CMetodo::IPR_VOGEL:

            {

            CTipoGeometria geometria = CTipoGeometria::
               radial;
cin.get();

forma = new CRadial;
CRadial* cast = dynamic_cast<CRadial*>(forma);

reservatorio.CoeficienteDietz(geometria);

cout << "Entre␣com␣o␣raio␣externo␣do␣reservatorio␣(FT)" <<
   endl;

double Re;

cin >> Re;

cast->RaioExterno(Re);
cast->Area();

cout << "Entre␣com␣a␣raio␣do␣poco␣(FT):" << endl;

double raio;

cin >> raio;

poco.setRaioPoco(raio);

cout << "Entre␣com␣a␣pressao␣inicial␣do␣reservatorio␣(PSIA):
   " << endl;

double pi;

cin >> pi;

reservatorio.PressaoInicial(pi);

cout << "Entre␣com␣a␣pressao␣de␣bolha␣(PSIA):␣" << endl;

double pb;

cin >> pb;

reservatorio.PressaoBolha(pb);
```

```
poco.CalcArea();

cout << "Qual␣tipo␣de␣fluido␣presente␣no␣reservatorio?" <<
   endl;
cout << "1␣-␣Oleo" << endl;
cout << "2␣-␣Gas" << endl;
cout << "
   ------------------------------------------------------------------
   " << endl;

CTipoFluido tipoFluido;

cin >> tipoFluido;
cin.get();

bool chs = true;

while(chs)

            {

                    switch (tipoFluido)

                    {

                            case CTipoFluido::oleo:

                                    {

            fluido = new COleo;
            cout << "Entre␣com␣fator␣volume␣de␣formacao␣do␣
               oleo:" << endl;
            double Bo;
            cin >> Bo;
            fluido->FatorVolumeFormacao(Bo);
            cout << "Entre␣com␣a␣viscosidade␣do␣fluido␣(CP):
               " << endl;
            double viscosidade;
            cin >> viscosidade;
            fluido->Viscosidade(viscosidade);

            chs = false;
            break;

                                    }

                            case CTipoFluido::gas:
```

```
                                        {

                    fluido = new CGas;
                    cout << "Entre␣com␣fator␣volume␣de␣formacao␣do␣
                       gas:" << endl;
                    double Bg;
                    cin >> Bg;
                    fluido->FatorVolumeFormacao(Bg);
                    cout << "Entre␣com␣a␣viscosidade␣do␣fluido␣(CP):
                       " << endl;
                    double viscosidade;
                    cin >> viscosidade;
                    fluido->Viscosidade(viscosidade);

                    chs = false;
                    break;

                                        }

                                default:
                    cout << "Entrada␣invalida!" << endl;
                    cout << "Qual␣tipo␣de␣fluido␣presente␣no␣
                       reservatorio?␣" << endl;
                    cin >> tipoFluido;
                    cin.get();

                            }

        }

        cout << "Entre␣com␣a␣espessura␣do␣reservatorio␣(FT):" <<
           endl;

        double espreservatorio;

        cin >> espreservatorio;

        reservatorio.Espessura(espreservatorio);

        cout << "Entre␣com␣a␣permeabilidade␣da␣rocha␣(MD):" << endl;

        double perm;

        cin >> perm;

        reservatorio.Permeabilidade(perm);
```

```
            cout << "Entre␣com␣fator␣de␣pelicula␣do␣reservatorio:" <<
               endl;

            double s;

            cin >> s;

            reservatorio.FatorPelicula(s);

            reservatorio.IndiceProdutividade(fluido, forma, poco);

            IPR = new CIPRVogel();

            break;

                }

                default:
            break;

    }

}

void CSimuladorCurvasIPR::Calculos() {

    cout << endl << endl;
    cout << "Calculando␣IPR..." << endl;
    cout << endl << endl;

    cout << "Deseja␣realizar␣o␣calculo␣de␣pressoes␣em␣quantas␣partes?"
       << endl; // Realizara os calculos com base na quantidade de
       pontos solicitados ate que atinja a pressao igual a zero
    cout << "Digite␣um␣numero␣inteiro."  << endl;

    int partes; // Variavel que recebe a entrada do usuario

    cin >> partes;

    IPR->setPartes(partes);

    if(metodo == CMetodo::IPR_FETKOVICH)
        IPR->CalcIPR(fluido, reservatorio, poco, pocoDois); // Para
           Fetkovich eh necessario dois pocos para calculo das
           constantes
    else
        IPR->CalcIPR(fluido, reservatorio, poco);
```

```
    vazoes = IPR->getVazao();
    pressoes = IPR->getPWF();

}

// Metodo para plotar o grafico

void CSimuladorCurvasIPR::Plotar() {

    cout << "Deseja␣plotar␣o␣grafico?" << std::endl;
    cout << "␣1␣-␣Sim" << endl;
    cout << "␣2␣-␣Nao␣" << endl;
    cout << "
       ------------------------------------------------------------------------
       " << endl;

    char ans; // Variavel para receber resposta do usuario

    cin >> ans;
    cin.get();

    bool boleano = true;

    while(boleano)

        {

        switch(ans)

                {

                        case '1':

                        {

                plot.XLabel("Vazao␣(STB/d)");
                plot.YLabel("Pressao␣de␣Fundo␣(PSIA)");
                plot.Title("Curva␣IPR");
                plot.Style("lines");
                plot.plot_xy(vazoes, pressoes);
                cin.get();
                boleano = false;
                break;

                        }

                        case '2':
                boleano = false;
```

```
                break;
            default:
                cout << "Opcao␣invalida!" << endl;

                }

    }

}

// Metodo para salvar o grafico
void CSimuladorCurvasIPR::SalvarGrafico() {

    plot.savetops("GraficoIPR");

}

// Metodo para executar o software
void CSimuladorCurvasIPR::Executar() {

    bool run = true;

    cout << endl;
    cout << "---------------␣SIMULADOR␣DE␣CURVAS␣IPR␣PARA␣POCOS␣
       VERTICAIS␣---------------" << endl;
    cout << "
       ------------------------------------------------------------------------------
       " << endl;
    cout << "------------␣Atila␣Junior,␣Giovanna␣Massardi␣e␣Marcelo␣
       Bernardo␣------------" << endl;
    cout << "--------------------------------␣UENF/LENEP␣
       --------------------------------" << endl;
    cout << endl;


    cout << "␣Deseja␣executar␣o␣programa␣?" << endl;
    cout << "␣1␣-␣Sim" << endl;
    cout << "␣2␣-␣Nao␣" << endl;
    cout << "
       ------------------------------------------------------------------------------
       " << endl;

    char ans; // Variavel para receber a resposta do usuario

    cin >> ans;
    cin.get();

// Caso usuario entre com uma opcao diferente das apresentadas retorna
```

```
    mensagem de erro

while (run)

{

        while ( ans!= '1' && ans != '2')

        {

                cout << "Opcao invalida" << endl;
        cout << " Deseja executar o programa ?" << endl;
        cout << " 1 - Sim" << endl;
        cout << " 2 - Nao " << endl;
        cout << "
           ------------------------------------------------------------------------
           " << endl;

        cin >> ans;
        cin.get();

    }

    switch(ans)

        {

                case '1': // Caso resposta positiva do usuario, executa
                    o programa

                {

                        EntradaDados();
            Calculos();
            Plotar();

            cout << "Deseja salvar o grafico?" << endl;
            cout << " 1 - Sim" << endl;
                cout << " 2 - Nao " << endl;
                cout << "
                   ------------------------------------------------------------------
                   " << endl;

            bool test = true;

            char chse; // Variavel para receber a resposta do usuario

            cin >> chse;
```

```
cin.get();

while(test)

            {

                    switch(chse)

                    {

                            case '1':

                            {

                                    SalvarGrafico();
                test = false;
                break;

                            }

                            case '2':

                            {

                                    test = false;
                break;

                            }

            // Caso usuario entre com uma opcao diferente das
               apresentadas retorna mensagem de erro

            default:

                            {

                                    cout << "Opcao Invalida!
                                       " << endl;
                cout << "Deseja salvar o grafico?" << endl;
                        cout << " 1 - Sim" << endl;
                            cout << " 2 - Nao " << endl;
                            cout << "
                               ------------------------------------
                               " << endl;
                cin >> chse;
                break;

                            }
```

```
                        }

            }

            cout << "␣Deseja␣executar␣o␣programa?" << endl;
            cout << "␣1␣-␣Sim" << endl;
                cout << "␣2␣-␣Nao␣" << endl;
                cout << "
                   ------------------------------------------------------------
                   " << endl;
            cin >> ans;
            cin.get();

            while ( ans!= '1' && ans != '2')

                        {

                                cout << "Opcao␣invalida" << endl;
                cout << "␣Deseja␣executar␣o␣programa␣?" << endl;
                cout << "␣1␣-␣Sim" << endl;
                        cout << "␣2␣-␣Nao␣" << endl;
                        cout << "
                           ------------------------------------------------------
                           " << endl;

                cin >> ans;
                cin.get();

            }

                        break;

                }

                case '2':
            run = false;
            cout << "Obrigado␣por␣utilizar␣o␣software!" << endl;
            break;
        default:
            break;

    }

}

}
```

Apresenta-se na listagem 6.29 a implementação da função `main`.

Listing 6.29: Arquivo de implementação da função `main()`

```
#include "CSimuladorCurvasIPR.h"


int main ()
{

    CSimuladorCurvasIPR simulador;

    simulador.Executar();

    return 0;
}
```

Capítulo 7

Teste

Todo projeto de engenharia passa por uma etapa de testes. Neste capítulo apresentamos alguns testes do software desenvolvido. Estes testes devem dar resposta aos diagramas de caso de uso inicialmente apresentados (diagramas de caso de uso geral e específicos).

7.1 Teste 1: Entrada de dados

O software inicia com a entrada de dados referentes ao problema como: tipo de fluido, raio de poço, viscosidade, fator volume formação do fluido e etc. Os dados são inseridos via teclado pelo usuário 7.1. É importante notar que os tipos de dados inseridos está condicionado ao método de IPR que será utilizado nos cálculos.

```
--------------- SIMULADOR DE CURVAS IPR PARA POCOS VERTICAIS ---------------
--------------------------------------------------------------------------------
------------ Atila Junior, Giovanna Massardi e Marcelo Bernardo ------------
------------------------------- UENF/LENEP -------------------------------

 Deseja executar o programa ?
 1 - Sim
 2 - Nao
--------------------------------------------------------------------------------
1

--------------- SIMULADOR DE CURVAS IPR PARA POCOS VERTICAIS ---------------
--------------------------------------------------------------------------------
------------ Atila Junior, Giovanna Massardi e Marcelo Bernardo ------------
------------------------------- UENF/LENEP -------------------------------

Aperte Enter para prosseguir...

Qual o metodo sera utilizado?
1 - IPR Linear
2 - IPR Generalizada
3 - IPR de Fetkovich (Ideal para reservatorios de gas)
4 - IPR de Vogel
--------------------------------------------------------------------------------
2

Entre com o raio externo do reservatorio (FT)
2980
Entre com a raio do poco (FT)
0.328
Entre com a pressao inicial (PSIA):
5651
Entre com a pressao de bolha (PSIA):
5651
Qual tipo de fluido presente no reservatorio?
1 - Oleo
2 - Gas
--------------------------------------------------------------------------------
1
Entre com fator volume de formacao do oleo:
1.1
Entre com a viscosidade do fluido (CP):
1.7
Entre com a espessura do reservatorio (FT):
53
Entre com a permeabilidade da rocha (MD):
```

Figura 7.1: Software - Entrada de dados.

7.2 Teste 2: Cálculos

Após a entrada de dados é realizado o cálculo de vazões com base no método selecionado. A saída dos resultados é gerada com o valor de vazão para a pressão de fundo em questão7.2.

```
Deseja realizar o calculo de pressoes em quantas partes?
Digite um numero inteiro.
20
Pwf: 5651, Vazao: 0
Pwf: 5368.45, Vazao: 53.5463
Pwf: 5085.9, Vazao: 107.093
Pwf: 4803.35, Vazao: 160.639
Pwf: 4520.8, Vazao: 214.185
Pwf: 4238.25, Vazao: 267.731
Pwf: 3955.7, Vazao: 321.278
Pwf: 3673.15, Vazao: 374.824
Pwf: 3390.6, Vazao: 428.37
Pwf: 3108.05, Vazao: 481.916
Pwf: 2825.5, Vazao: 535.463
Pwf: 2542.95, Vazao: 589.009
Pwf: 2260.4, Vazao: 642.555
Pwf: 1977.85, Vazao: 696.102
Pwf: 1695.3, Vazao: 749.648
Pwf: 1412.75, Vazao: 803.194
Pwf: 1130.2, Vazao: 856.74
Pwf: 847.65, Vazao: 910.287
Pwf: 565.1, Vazao: 963.833
Pwf: 282.55, Vazao: 1017.38
Pwf: 0, Vazao: 1070.93
Deseja plotar o grafico?
 1 - Sim
 2 - Nao
--------------------------------------------------------------------------------
```

Figura 7.2: Cálculo de vazão.

7.3 Teste 3: Plotar gráficos

Após a entrada de dados e cálculo dos resultados o usuário pode optar pelo plot das curvas. Os resultados quando comparados à literatura foram satisfatórios 7.3.

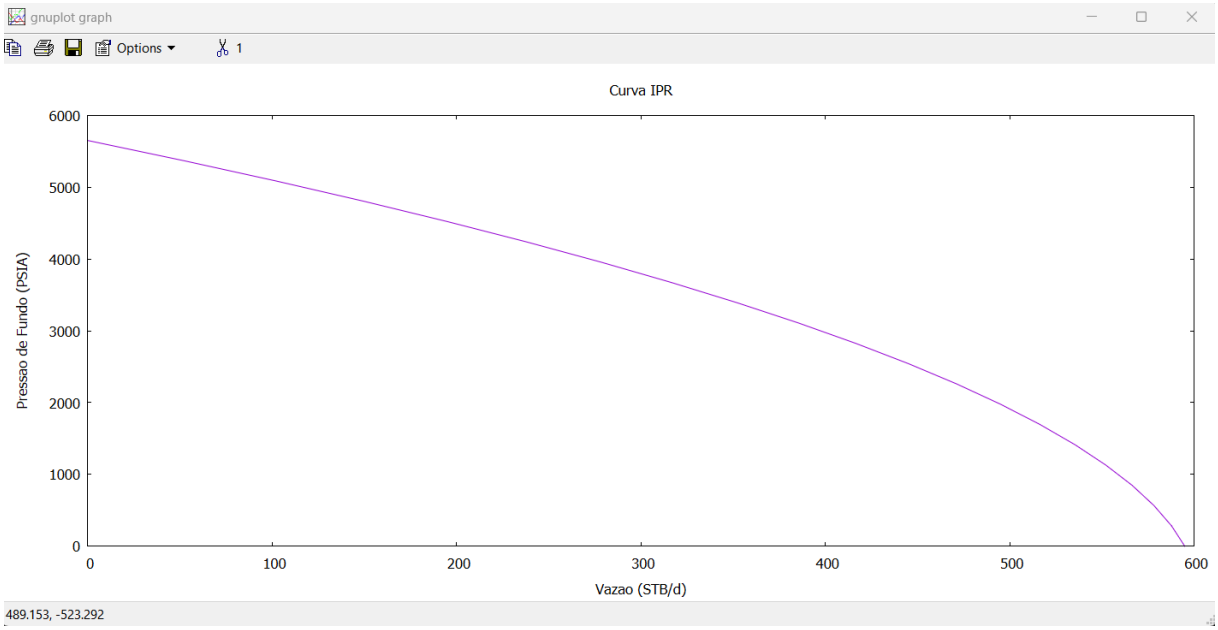

Figura 7.3: Gráfico IPR generalizada

7.4 Teste 4: Salvar gráficos

Após o gráfico ser gerado o usuário possui a autonomia de salvar o gráfico e executar novamente o programa para utilizar outros métodos de cálculo de IPR 7.4.

```
Deseja realizar o calculo de pressoes em quantas partes?
Digite um numero inteiro.
20
Pwf: 5651, Vazao: 0
Pwf: 5368.45, Vazao: 53.5463
Pwf: 5085.9, Vazao: 107.093
Pwf: 4803.35, Vazao: 160.639
Pwf: 4520.8, Vazao: 214.185
Pwf: 4238.25, Vazao: 267.731
Pwf: 3955.7, Vazao: 321.278
Pwf: 3673.15, Vazao: 374.824
Pwf: 3390.6, Vazao: 428.37
Pwf: 3108.05, Vazao: 481.916
Pwf: 2825.5, Vazao: 535.463
Pwf: 2542.95, Vazao: 589.009
Pwf: 2260.4, Vazao: 642.555
Pwf: 1977.85, Vazao: 696.102
Pwf: 1695.3, Vazao: 749.648
Pwf: 1412.75, Vazao: 803.194
Pwf: 1130.2, Vazao: 856.74
Pwf: 847.65, Vazao: 910.287
Pwf: 565.1, Vazao: 963.833
Pwf: 282.55, Vazao: 1017.38
Pwf: 0, Vazao: 1070.93
Deseja plotar o grafico?
 1 - Sim
 2 - Nao
--------------------------------------------------------------------------------
1

Deseja salvar o grafico?
 1 - Sim
 2 - Nao
--------------------------------------------------------------------------------
1
 Deseja executar o programa?
 1 - Sim
 2 - Nao
--------------------------------------------------------------------------------
```

Figura 7.4: Gráfico IPR generalizada

Capítulo 8

Documentação para o Desenvolvedor

Todo projeto de engenharia precisa ser bem documentado. Neste sentido, apresenta-se neste capítulo documentações extras para o desenvolvedor. Ou seja, instruções para pessoas que venham a dar continuidade a este projeto de engenharia.

Nota: O manual do usuário é apresentado em um documento separado. Veja diretório "doc/ManualDoUsuario".

8.1 Dependências para compilar o software

Para compilar o software é necessário atender as seguintes dependências:

- Instalar o compilador g++ da GNU disponível em http://gcc.gnu.org. Para instalar no GNU/Linux use o comando `yum install gcc`.
- Biblioteca CGnuplot; os arquivos para acesso a biblioteca CGnuplot devem estar no diretório com os códigos do software;
- O software `gnuplot`, disponível no endereço http://www.gnuplot.info/, deve estar instalado. É possível que haja necessidade de setar o caminho para execução do `gnuplot`.
- .
- .

8.2 Como gerar a documentação usando doxygen

A documentação do código do software deve ser feita usando o padrão JAVADOC, conforme apresentada no Capítulo - Documentação, do livro texto da disciplina. Depois de documentar o código, use o software `doxygen` para gerar a documentação do desenvolvedor no formato html. O software `doxygen` lê os arquivos com os códigos (*.h e *.cpp) e gera uma documentação muito útil e de fácil navegação no formato html.

- Veja informações sobre uso do formato JAVADOC em:
  - http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html
- Veja informações sobre o software `doxygen` em
  - http://www.stack.nl/~dimitri/doxygen/

Passos para gerar a documentação usando o `doxygen`.

- Documente o código usando o formato JAVADOC. Um bom exemplo de código documentado é apresentado nos arquivos da biblioteca CGnuplot, abra os arquivos `CGnuplot.h` e `CGnuplot.cpp` no editor de texto e veja como o código foi documentado.
- Abra um terminal.
- Vá para o diretório onde esta o código.

  ```
  cd /caminho/para/seu/codigo
  ```

- Peça para o `doxygen` gerar o arquivo de definições (arquivo que diz para o doxygem como deve ser a documentação).

  ```
  dogygen -g
  ```

- Peça para o `doxygen` gerar a documentação.

  ```
  doxygen
  ```

- Verifique a documentação gerada abrindo o arquivo html/index.html.

  ```
  firefox html/index.html
  ```

  ou

  ```
  chrome html/index.html
  ```

Apresenta-se a seguir algumas imagens com as telas das saídas geradas pelo software `doxygen`.

Nota:

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

Capítulo 9

Sugestões para Trabalhos Futuros

Com a finalidade de melhorar o projeto, sugere-se que os tópicos abaixo sejam incorporados de modo que o código e formulação teórica sejam ampliados a outros casos.

- Formulação Teórica:
  - Adicionar a IPR que abrange reservatórios estratificados, ou seja, reservatórios com mais de uma camada e possibilitar o cálculo utilizando os modelos empíricos.
  - Adicionar a IPR Futura, método utilizado para prever produtividade.
  - Implementar outros regimes de escoamento, como o permanente e transiente.
  - Considerar os cálculos para poços horizontais ou direcionais.
- Formulação do Código:
  - Permitir que o usuário entre com os dados na forma de arquivo de disco.
  - Permitir que o usuário salve os resultados de pressão e vazão calculados pelo método escolhido.

Referências Bibliográficas

[1] Tarek Ahmed. *Reservoir Engineering Book*. ELSEVIER, Oxford, 2006.

[2] James P. Brill. *Multiphase Flow in Wells*. SPE, Pennsylvania, 1999.

[3] AndrÃ© Duarte Bueno. *ProgramaÃ§Ã£o Orientada a Objeto com C++ - Aprenda a Programar em Ambiente Multiplataforma com Software Livre*. Novatec, SÃ£o Paulo, 2003.

[4] Boyun Guo and Ali Ghalambor. *Well Productivity Handbook*. GPC, 2008.

[5] Boyun Guo, William C. Lyons, and Ali Ghalambor. *Petroleum Production Engineering*. ELSEVIER, Oxford, 2007.

[6] Adalberto JosÃ© Rosa and Renato de Souza Carvalho. *Engenharia de Reservatorio de Petroleo*. InterciÃªncia, 2006.

[7] Moshood Sanni. *Petroleum Engineering*. Wiley, 2018.