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

PROJETO DE ENGENHARIA
DESENVOLVIMENTO DO SOFTWARE
FERRAMENTA PARA GERAÇÃO DA CURVA DE FLUXO FRACIONÁRIO

DISCIPLINA LEP-0144: Projeto de Software Aplicado à Engenharia
Setor de Modelagem Matemática Computacional

Versão 1:
Fabiane da Silva Barros
Prof. André Duarte Bueno

MACAÉ - RJ
Novembro - 2025

Sumário

Lista de Figuras

Lista de Tabelas

Listagens

Capítulo 1

Concepção

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

1.1 Metodologia

A Figura 1.1 apresenta a metodologia a ser utilizada no desenvolvimento do sistema. O desenvolvimento do código-fonte e a estruturação dos documentos

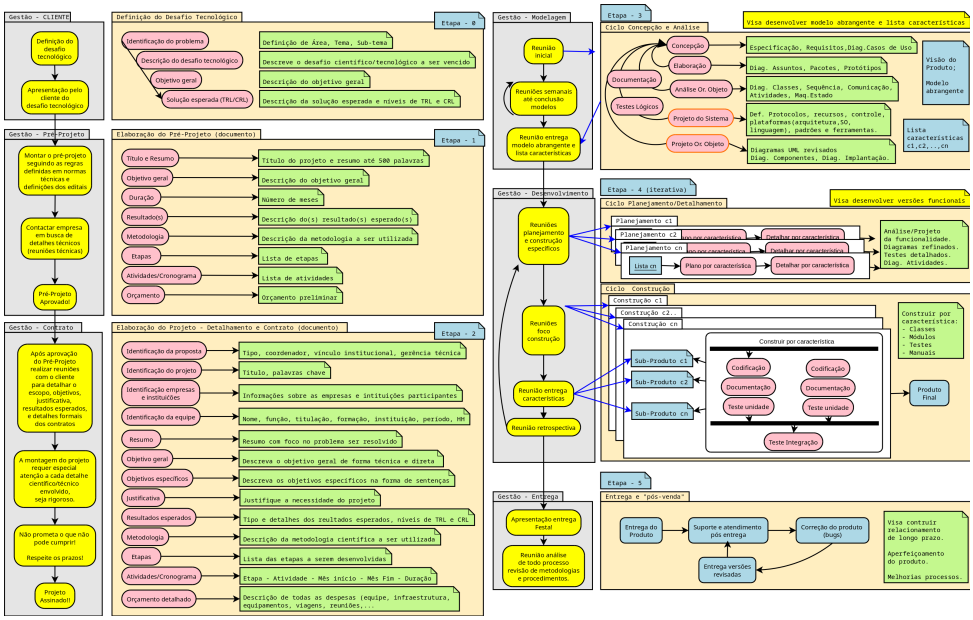

Figura 1.1: Metodologia utilizada no desenvolvimento do sistema.

LaTeX foram auxiliados pelo uso de modelos de linguagem (IA) para otimização de sintaxe e depuração de erros de compilação. Todo o código gerado foi revisado, testado e validado pela autora.

1.2 Nome do Sistema/Produto

Abaixo a Tabela 1.1 com o nome do Sistema/Produto: Ferramenta para Geração da Curva de Fluxo Fracionário.

Tabela 1.1: Nome do Sistema/Produto.

Nome	Ferramenta para Geração da Curva de Fluxo Fracionário
Componentes principais	1. Módulo de Entrada (Leitor de arquivo de texto) 2. Módulo de Dados (Modelos de Permeabilidade) 3. Módulo de Cálculo (Equação de Buckley-Leverett) 4. Módulo de Saída (Gerador de `.csv` e interface com Gnuplot)
Missão	Missão,Calcular e visualizar a curva de fluxo fracionário (f_w vs. S_w) para auxiliar na análise de deslocamento bifásico em meios porosos.

1.3 Especificação

A especificação desta solução de engenharia define uma ferramenta computacional analítica, de caráter didático e profissional, implementada como um executável de linha de comando. O sistema irá abstrair a complexidade do modelo matemático de fluxo fracionário, exigindo do usuário apenas um arquivo de configuração em texto simples. Este arquivo deverá conter os parâmetros reológicos dos fluidos (viscosidades) e os dados de permeabilidade relativa, que são suportados em dois formatos: (a) dados tabulares discretos (S_w, k_{rw}, k_{ro}), que serão processados via interpolação linear; ou (b) parâmetros para o modelo analítico de Corey [Corey, 1954]). O software processará esses dados, calculará a curva f_w vs S_w e gerará dois produtos: um arquivo de dados `.csv` para análise externa e uma visualização gráfica imediata, invocando o Gnuplot para plotar a curva "S" característica.

1.4 Requisitos

Apresenta-se nesta seção os requisitos funcionais e não funcionais.

1.4.1 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

RF-01	O sistema deve ler um arquivo de entrada de texto contendo os parâmetros da simulação (viscosidades e dados de permeabilidade).

RF-02	O sistema deve permitir que os dados de permeabilidade relativa sejam fornecidos de duas formas: (a) uma tabela de pontos (S_w, k_{rw}, k_{ro}) ou (b) parâmetros para um modelo analítico (Correlações de Corey[Corey, 1954]).

RF-03	O sistema deve calcular os valores de fluxo fracionário (f_w) para um intervalo de saturação de água (S_w) de 0 a 1.

RF-04	O sistema deve gerar um arquivo de saída em formato de texto (ex: `.csv`) contendo os pares de dados (S_w, f_w) calculados.

RF-05	O sistema deve invocar um processo externo (Gnuplot) para plotar automaticamente o gráfico de f_w vs. S_w.

RF-06	O sistema deve informar ao usuário em caso de erro na leitura do arquivo de entrada (ex: arquivo não encontrado, formato inválido).

1.4.2 Requisitos não funcionais

RNF-01	Os cálculos de permeabilidade relativa, quando tabulados, devem usar interpolação linear.

RNF-02	O programa deverá ser multi-plataforma, podendo ser executado em *Windows* e GNU/Linux.

RNF-03	O software deve ser desenvolvido na linguagem C++ utilizando o paradigma da Orientação a Objetos.

RNF-04	O software deve ser executável via terminal (linha de comando).

1.5 Casos de Uso

A Tabela 1.2 mostra a descrição de um caso de uso.

Tabela 1.2: Caso de uso 1.

Nome do caso de uso:	Executar Cálculo e Analisar Resultados
Resumo/descrição:	O usuário executa a ferramenta, que lê os dados de entrada, calcula a curva de fluxo fracionário e exibe o gráfico resultante.
Etapas:	1. Usuário executa o programa via terminal, fornecendo o nome do arquivo de entrada. 2. O sistema lê e valida o arquivo de entrada. 3. O sistema calcula a curva de f_w vs. S_w. 4. O sistema salva os resultados em um arquivo `.csv`. 5. O sistema invoca o Gnuplot para exibir o gráfico. 6. O usuário analisa o gráfico e fecha a janela.
Cenários alternativos:	2a. O arquivo de entrada não existe ou está mal formatado: O sistema exibe uma mensagem de erro clara e encerra. 5a. O Gnuplot não está instalado ou não foi encontrado no PATH do sistema: O sistema exibe um aviso, mas informa que o arquivo de dados `.csv` foi gerado com sucesso.

1.5.1 Diagrama de caso de uso geral

O diagrama de caso de uso geral da Figura 1.2 mostra o ator principal, o Usuário (Engenheiro/Estudante), e as interações primárias que ele pode ter com o sistema.

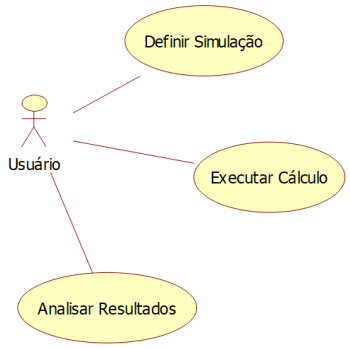

Figura 1.2: Diagrama de caso de uso – Caso de uso geral

1.5.2 Diagrama de caso de uso específico

O caso de uso Executar Cálculo é detalhado na Figura 3.2.2. Ele é composto por várias sub tarefas que são obrigatoriamente incluídas em sua execução.

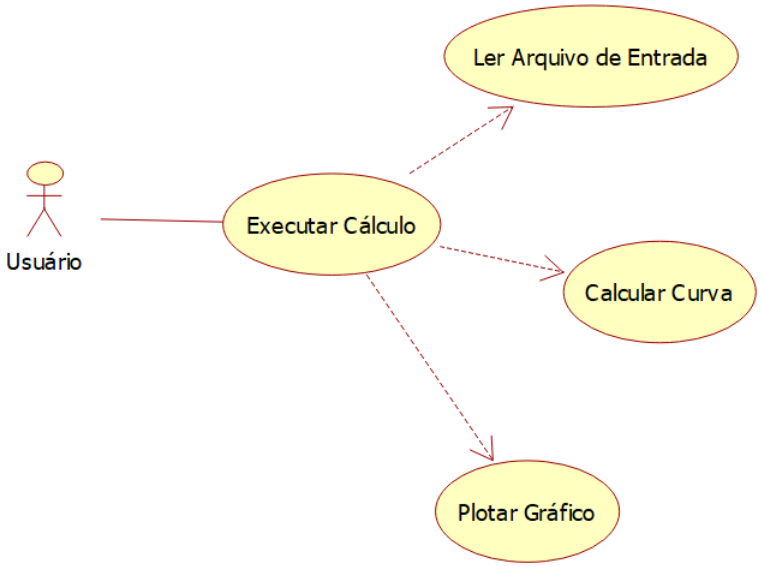

Figura 1.3: Diagrama de caso de uso específico – Título.

Capítulo 2

Elaboração

Este capítulo detalha o domínio do problema e a lógica de engenharia.

2.1 Análise de domínio

O domínio deste projeto situa-se na interseção da Engenharia de Reservatórios com a Modelagem Computacional. A seguir, detalha-se a análise deste domínio e as decisões de arquitetura de software que ela motivou.

- Justificativa da Plataforma Tecnológica: A linguagem C++ (padrão C++ 17) foi selecionada como base tecnológica. A decisão se justifica pelo seu comprovado desempenho computacional, controle de baixo nível de memória e vasta biblioteca padrão (STL). Essa escolha garante que o núcleo de cálculo seja eficiente e, futuramente, possa ser facilmente integrado como uma biblioteca em simuladores de reservatório mais complexos. O desenvolvimento visa a portabilidade entre *Windows* e GNU/Linux, utilizando ferramentas de compilação padrão (g++).

- Arquitetura de Dados: Optou-se por uma arquitetura *serverless* e sem banco de dados. A persistência dos dados de entrada é gerenciada pelo usuário via arquivos de texto `.txt` com formato baseado em palavras-chave. Esta decisão mitiga a complexidade de *deploy* e configuração, alinhando-se à natureza de uma ferramenta de análise rápida. O protocolo de saída (`.csv`) foi escolhido por sua universalidade, permitindo a interoperabilidade com qualquer software de planilha ou *script* de pós-processamento.

- Gerenciamento de Dependências: O projeto terá uma única dependência externa principal: o Gnuplot. Para manter o software leve e evitar a complexidade de linkagem de bibliotecas gráficas, o Gnuplot será invocado como um processo de sistema externo (`system()`). A classe Gnuplot do nosso projeto atuará como uma Facade, encapsulando essa chamada.

- Modelo de Controle: O fluxo de execução do software é puramente sequencial e síncrono (Leitura -> Cálculo -> Plotagem). Dada a baixa complexidade computacional do modelo 1D, não há necessidade de implementação de concorrência ou processamento paralelo, o que simplifica o design.

2.2 Formulação teórica

O núcleo deste projeto é a implementação da equação de fluxo fracionário (f_w) de Buckley-Leverett[Buckley and Leverett, 1942], conforme definido no documento do Desafio Tecnológico :

$$f_w = \frac{1}{1 + \frac{k_{ro}(S_w)}{\mu_o} \cdot \frac{\mu_w}{k_{rw}(S_w)}}$$

A curva, resultado da equação do fluxo fracionário possui uma característica de forma “S”, como está apresentado na Figura 2.1.

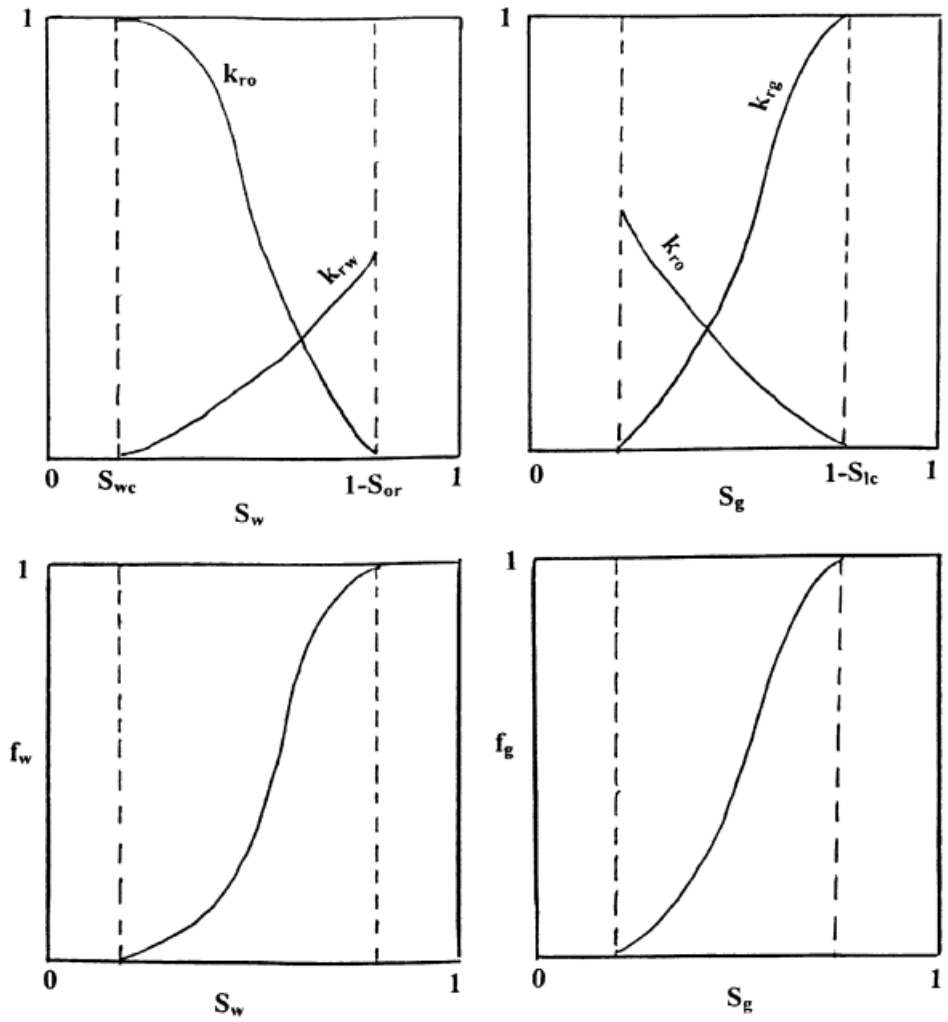

Figura 2.1: Curva de fluxo fracionário (f_w) característica, mostrando a sua forma de "S". Fonte: Adaptado de Ahmed, 2019 [Ahmed, 2010]..

O desafio computacional reside em como obter os valores de $k_{ro}(S_w)$ e $k_{rw}(S_w)$ para qualquer valor de saturação de água S_w, uma vez que os dados de entrada podem vir de duas formas distintas: um modelo tabular ou um modelo analítico.

2.2.1 Implementação do Modelo Tabular (Interpolação Linear)

Quando os dados são fornecidos como uma tabela de pontos (lidos do arquivo de texto), o programa precisa calcular os valores de k_r para saturações que estão entre os pontos da tabela. Para isso, usaremos a Interpolação Linear.

- Algoritmo:

1. O software lerá o arquivo de entrada e armazenará os N pontos de dados em três vetores (ou listas) paralelos: `vec_Sw[]`, `vec_Krw[]`, `vec_Kro[]`.
2. Quando a função `calcularFw(Sw_desejada)` for chamada:
3. O programa irá buscar no `vec_Sw[]` os dois valores, $S_{w,i}$ e $S_{w,i+1}$, que "cercam" o valor de S_w desejado (ou seja, $S_{w,i} \leq S_w \leq S_{w,i+1}$).
4. A interpolação linear é aplicada para encontrar o k_{rw} e k_{ro} correspondentes:
   - $k_{rw} = k_{rw,i} + (S_{w,desejada} - S_{w,i}) \times \frac{k_{rw,i+1} - k_{rw,i}}{S_{w,i+1} - S_{w,i}}$
   - $k_{ro} = k_{ro,i} + (S_{w,desejada} - S_{w,i}) \times \frac{k_{ro,i+1} - k_{ro,i}}{S_{w,i+1} - S_{w,i}}$
5. Estes valores interpolados de k_{rw} e k_{ro} são então usados na equação principal de f_w.
6. Tratamento de Pontas: Se $S_{w,desejada}$ for menor que o primeiro valor da tabela, os valores do primeiro ponto $(k_{rw,0}, k_{ro,0})$ serão usados. Se for maior que o último ponto, os valores do último ponto $(k_{rw,N}, k_{ro,N})$ serão usados (sem extrapolação).

2.2.2 Implementação do Modelo Analítico (Correlações de Corey)

Quando o arquivo de entrada especifica o uso de um modelo analítico como o de Corey [Corey, 1954], o programa não usará uma tabela, mas sim um conjunto de parâmetros para calcular k_r diretamente.

- Algoritmo:

1. O software lerá do arquivo de entrada os parâmetros do modelo de Corey:
   - S_{wir}: Saturação de água irreduzível.
   - S_{orw}: Saturação de óleo residual à injeção de água.
   - $k_{rw,max}$ (ou k_{rwro}): Permeabilidade relativa à água na saturação de óleo residual.
   - $k_{ro,max}$ (ou k_{rocw}): Permeabilidade relativa ao óleo na saturação de água irreduzível.

- n_w: Expoente de Corey para a fase água.
- n_o: Expoente de Corey para a fase óleo.

2. Primeiro, calcula-se a saturação de água normalizada ($S_{w,norm}$) para qualquer S_w de entrada:
   - $S_{w,norm} = \frac{S_w - S_{wir}}{1 - S_{wir} - S_{orw}}$
3. A $S_{w,norm}$ é limitada entre 0 e 1 (se $S_w < S_{wir}$, $S_{w,norm} = 0$; se $S_w > (1 - S_{orw})$, $S_{w,norm} = 1$).
4. As permeabilidades relativas são então calculadas usando $S_{w,norm}$:
   - $k_{rw}(S_w) = k_{rw,max} \times (S_{w,norm})^{n_w}$
   - $k_{ro}(S_w) = k_{ro,max} \times (1 - S_{w,norm})^{n_o}$
5. Estes valores calculados de k_{rw} e k_{ro} são então usados na equação principal de f_w.

A estrutura do software em C++ será projetada de forma que a calculadora principal de f_w não precise saber qual desses métodos está sendo usado, apenas que ela pode pedir os valores de k_r.

Para uma dedução mais detalhada destas equações e uma discussão aprofundada sobre as propriedades dos fluidos e modelos de permeabilidade, consulte as referências clássicas da engenharia de reservatórios, como Ahmed [Ahmed, 2010] e Lake [Lake et al., 2010].

2.3 Identificação de pacotes – assuntos

Identificamos os seguintes pacotes (assuntos) lógicos para a organização do sistema:

- Pacote `Core (Núcleo)`: Contém a classe principal com a lógica de engenharia, ou seja, a `CalculadoraFluxoFracionario`. Este pacote implementa a equação de Buckley-Leverett.
- Pacote `ModelosKr`: Contém a interface `ICurvasPermeabilidade` e suas implementações concretas (`CurvasPermeabilidadeTabelada` e `CurvasPermeabilidadeCorey`). Este pacote é responsável por fornecer os dados de permeabilidade relativa para o pacote Core.
- Pacote `IO (Entrada/Saída)`: Contém classes utilitárias responsáveis pela leitura e *parsing* do arquivo de entrada de texto e pela escrita do arquivo de saída `.csv`.
- Pacote `Visualizacao`: Contém a classe utilitária Gnuplot, responsável por formatar os dados de saída e invocar o processo externo do Gnuplot para plotagem.

- Pacote `Aplicacao`: Contém a classe Simulador (com a função `main()`), que orquestra todo o processo: lê os dados usando `IO`, instancia o `ModelosKr` correto, instancia o `Core` e, por fim, chama a `Visualizacao`.

2.4 Diagrama de pacotes – assuntos

O Diagrama de Pacotes, apresentado na Figura 2.2, ilustra as dependências entre os assuntos lógicos identificados.

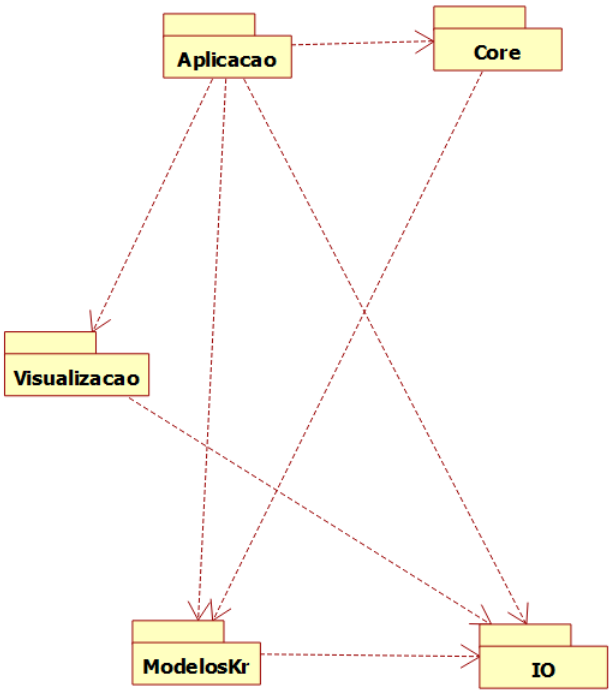

Figura 2.2: Diagrama de Pacotes.

Capítulo 3

AOO – Análise Orientada a Objeto

3.1 Diagramas de classes

O design estático do sistema, apresentado na Figura 3.1, é fundamentado em princípios S.O.L.I.D. de engenharia de software para garantir um código limpo, extensível e de fácil manutenção.

1. Princípio da Inversão de Dependência (D do S.O.L.I.D.): Este é o pilar central da arquitetura. O módulo de alto nível (`CalculadoraFluxoFracionario`) não depende diretamente dos módulos de baixo nível (...Tabelada, ...Corey). Ambos dependem de uma abstração: a interface `ICurvasPermeabilidade`.

2. Padrão de Projeto *Strategy* (Estratégia): A interface `ICurvasPermeabilidade` implementa o padrão *Strategy*. Ela define um "contrato" para uma família de algoritmos (os modelos de k_r) e permite que eles sejam trocados dinamicamente. A Calculadora é configurada com uma dessas estratégias no momento da sua criação, através de Injeção de Dependência em seu construtor. Isso torna a calculadora completamente agnóstica aos detalhes de como o k_r é obtido.

3. Padrão de Projeto Facade (Fachada): A classe Gnuplot é uma Facade estática. Ela fornece uma interface unificada e simples (`plotarCurva()`) para um subsistema complexo (formatação de *scripts*, salvamento de arquivos temporários e chamada de processos do sistema operacional).

3.1.1 Dicionário de classes

- Classe `ICurvasPermeabilidade`: (Interface) Define o contrato de serviço para os modelos de permeabilidade relativa. Garante o polimorfismo e a Inversão de Dependência.

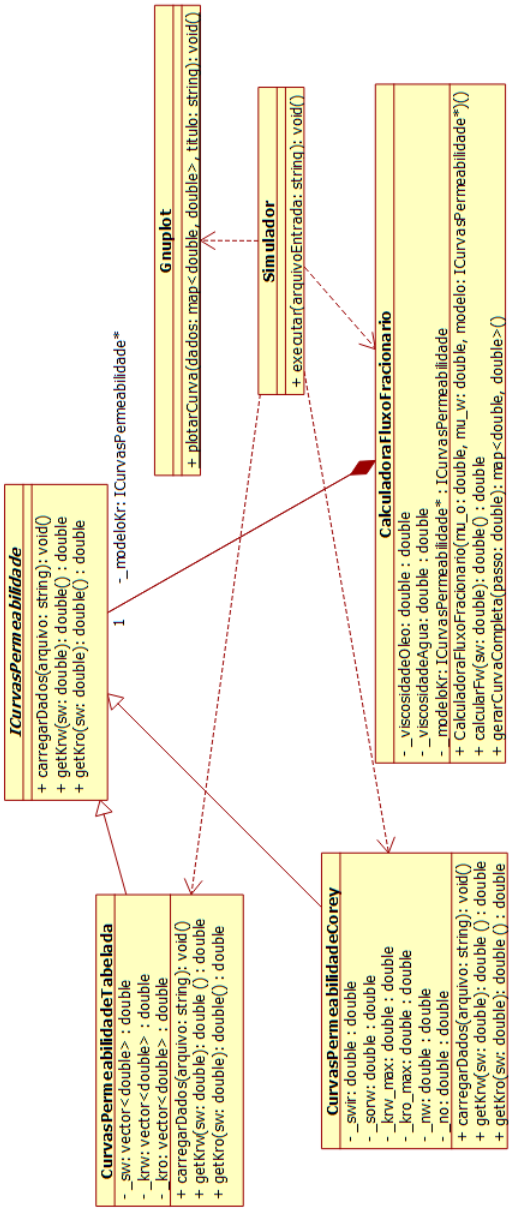

Figura 3.1: Diagrama de classes.

- Classe `CurvasPermeabilidadeTabelada`: (Implementação Concreta) Implementação da *Strategy* de k_r via dados tabulares. Encapsula a lógica de interpolação linear.
- Classe `CurvasPermeabilidadeCorey`: (Implementação Concreta) Implementação da *Strategy* de k_r via correlação analítica de Corey [Corey, 1954]. Encapsula a lógica matemática do modelo.
- Classe `CalculadoraFluxoFracionario`: (Domínio de Negócio) Classe principal que implementa o domínio central de negócio (a equação de Buckley-Leverett). Opera em um alto nível de abstração, dependendo apenas da interface `ICurvasPermeabilidade`.
- Classe `Gnuplot`: (Utilitário / Facade) Classe estática que encapsula toda a lógica de interação com o Gnuplot, simplificando a plotagem para o resto da aplicação.
- Classe `Simulador`: (Orquestrador / Ponto de Entrada) Atua como o orquestrador da aplicação. É responsável por "montar" o sistema (ler arquivos, instanciar os objetos corretos usando *new*) e injetar as dependências (`modeloKr` dentro da Calculadora). Contém o ponto de entrada `main()`.

3.2 Diagrama de sequência – eventos e mensagens

3.2.1 Diagrama de sequência geral

O diagrama de sequência geral, Figura 3.2, ilustra o fluxo principal de execução do software, correspondente ao caso de uso "Executar Cálculo e Analisar Resultados" com sucesso.

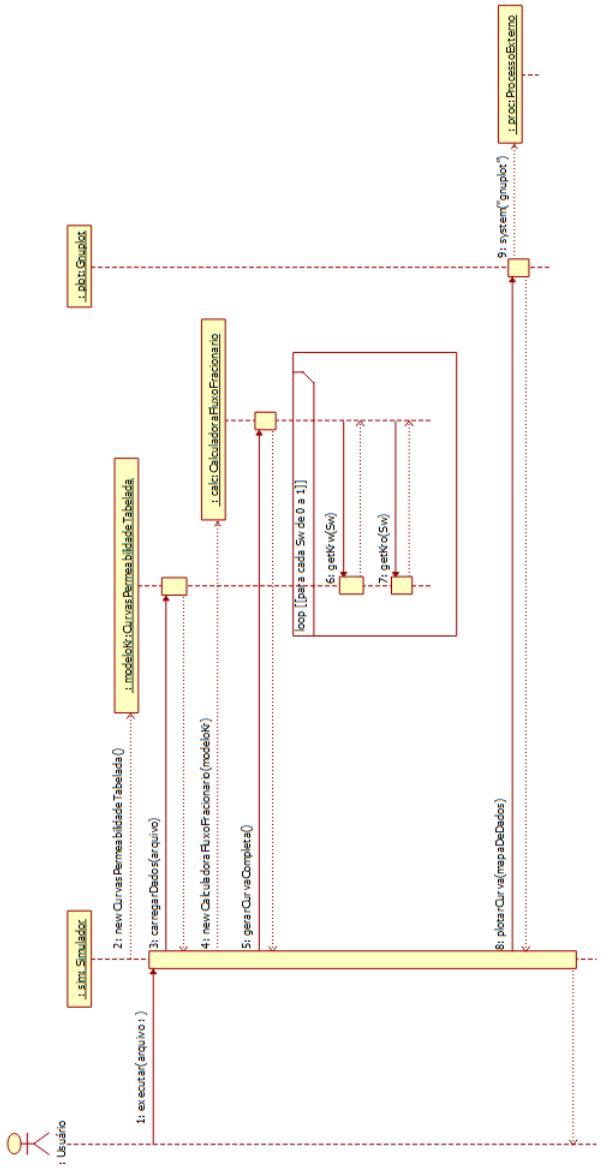

Figura 3.2: Diagrama de sequência.

3.2.2 Diagrama de sequência específico

O diagrama da Figura 3.3 detalha a atividade específica de `getKrw()` na classe `CurvasPermeabilidadeTabelada`, mostrando a lógica de interpolação.

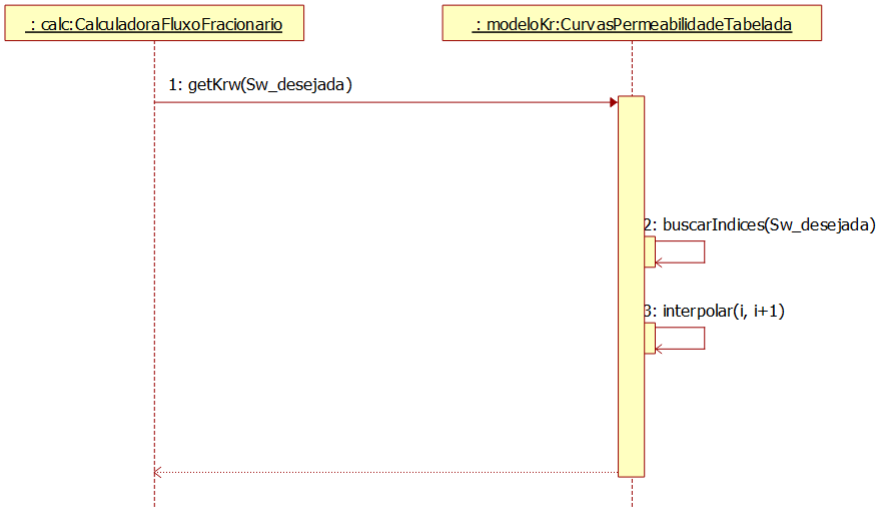

Figura 3.3: Diagrama de sequência.

3.3 Diagrama de comunicação – colaboração

O diagrama de comunicação da Figura 3.3 apresenta a mesma interação do diagrama de sequência geral, mas foca na colaboração e nas mensagens trocadas entre os objetos, em vez da ordem temporal.

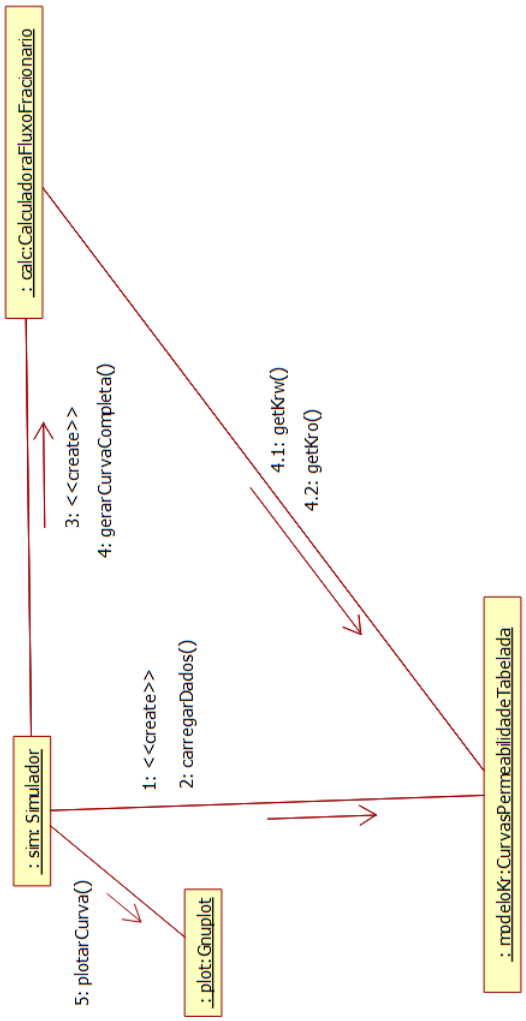

Figura 3.4: Diagrama de comunicação.

3.4 Diagrama de estado

O diagrama da Figura 3.5 modela os estados dinâmicos de um objeto Simulador ao longo de sua vida, desde a criação até a finalização da execução.

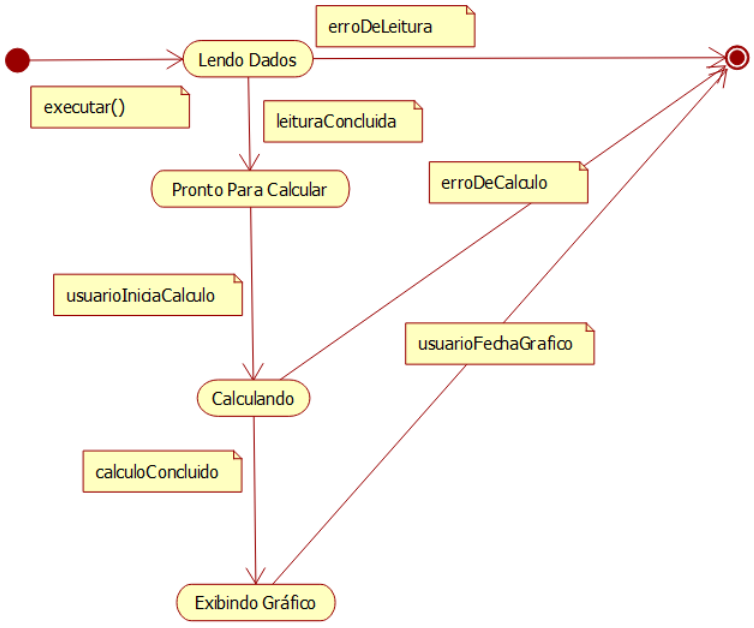

Figura 3.5: Diagrama de máquina de estado.

3.5 Diagrama de atividades

O diagrama de atividades da Figura 3.6 detalha o algoritmo do método `getKrw()` na classe `CurvasPermeabilidadeTabelada`, como descrito na Formulação Teórica. Este é um cálculo específico e crucial.

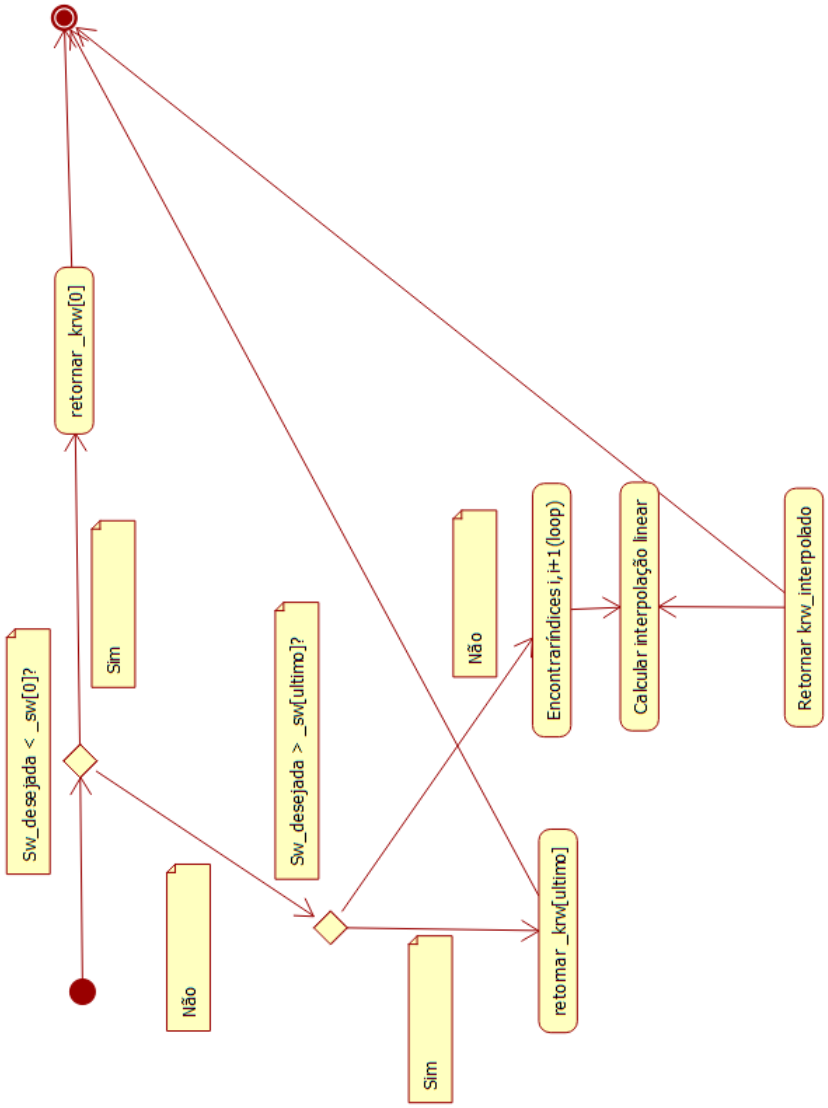

Figura 3.6: Diagrama de atividades.

Capítulo 4

Projeto

4.1 Projeto do sistema

1. Protocolos

   - Comunicação Externa: O protocolo de comunicação com o usuário será via arquivos de texto. O formato do arquivo de entrada (detalhado no Apêndice A) é o protocolo principal . O sistema gerará um arquivo `.csv` (valores separados por vírgula) como protocolo de saída, garantindo fácil interoperabilidade.
   - Comunicação Interna: A comunicação entre os objetos do sistema seguirá um padrão de passagem de mensagens síncronas (chamadas de método diretas).
   - API: A API interna da biblioteca de cálculo será definida pela interface (classe base abstrata) `ICurvasPermeabilidade` e pela classe `CalculadoraFluxoFracionario` .

2. Recursos

   - Banco de Dados: Não será utilizado um banco de dados . Para o escopo da aplicação (uma ferramenta de análise leve), um BD introduziria uma complexidade de implantação e configuração desnecessária.
   - Armazenamento: A persistência dos dados de entrada é gerenciada pelo usuário através de arquivos `.txt`.

3. Controle

   - Implementação: O controle do sistema será sequencial e baseado em procedimentos. A execução seguirá um fluxo linear (Ler -> Calcular -> Plotar).

- Concorrência: Não há necessidade de concorrência ou processamento paralelo (*multithreading*), pois o custo computacional do cálculo do fluxo fracionário 1D é baixo e a execução é instantânea.

4. Plataformas

   - Hardware/SO: O software será desenvolvido em C++ padrão (C++17), garantindo alta portabilidade e compilação nativa em *Windows* e GNU/Linux.
   - Bibliotecas Externas: A única dependência externa será o Gnuplot. Para manter o software leve, o Gnuplot não será linkado ao código; ele será invocado como um processo de sistema externo (`system()`).
   - IDE: O desenvolvimento será feito utilizando um editor de texto (Kate) e compilação via *make* no terminal MSYS2.

5. Padrões de projeto

- Conforme detalhado no Capítulo 3 , o design da arquitetura é centrado no uso dos padrões de projeto *Strategy* (para os modelos de permeabilidade) e Facade (para a classe Gnuplot) .

4.2 Projeto orientado a objeto – POO

- Efeitos do projeto no modelo estrutural: A arquitetura definida na Análise (Cap. 3) é mantida. O uso do padrão *Strategy* garante que o Diagrama de Pacotes seja limpo: o pacote Core depende apenas da abstração em `ModelosKr`, não das implementações concretas .

- Efeitos do projeto no modelo dinâmico: Os diagramas de sequência e estado confirmam o fluxo de controle síncrono e sequencial. A chamada ao Gnuplot (`system()`) será bloqueante, garantindo que o programa só termine após o usuário fechar o gráfico.

- Efeitos do projeto nos atributos: Todos os atributos de classe serão privados (*private*), garantindo encapsulamento. A `CalculadoraFluxoFracionario` armazenará um ponteiro para a interface `ICurvasPermeabilidade`, minimizando o acoplamento.

- Efeitos do projeto nos métodos: Os métodos de cálculo (como `getKrw`, `calcularFw`) serão declarados como `const` sempre que possível, para garantir que não modifiquem o estado do objeto. O método `plotarCurva` da classe `Gnuplot` será `static`, pois não depende de um estado de instância.

- Efeitos do projeto nas heranças: A herança será usada exclusivamente para implementar interfaces (herança de tipo), não para reutilização de código. Isso segue o princípio de "Composição sobre Herança".

- Efeitos do projeto nas associações: A associação principal é a de uso (agregação), onde a Calculadora usa a interface `ICurvasPermeabilidade` que lhe é injetada. O Simulador gerencia o ciclo de vida (criação *new* e destruição delete) dos objetos.
- Efeitos do projeto nas otimizações: Nenhuma otimização de baixo nível (ex: paralelismo) é necessária. A otimização do projeto está no design de O.O., que evita recálculos e promove baixo acoplamento.

4.3 Diagrama de componentes

O Diagrama de Componentes (Figura 4.1) detalha as dependências de compilação e ligação do software. O componente executável final, `fw_calc` , depende da compilação e linkagem de múltiplos componentes de código-fonte (`.cpp`). Cada componente `.cpp` (ex: `Simulador.cpp`, `Calculadora.cpp`) depende do seu respectivo arquivo de cabeçalho (`.h`) e das abstrações das quais depende (ex: `Calculadora.cpp` depende de `ICurvasPermeabilidade.h`). O componente `Gnuplot.cpp` não depende do executável do Gnuplot em tempo de compilação, mas o executável `fw_calc` tem uma dependência em tempo de execução com o Gnuplot.

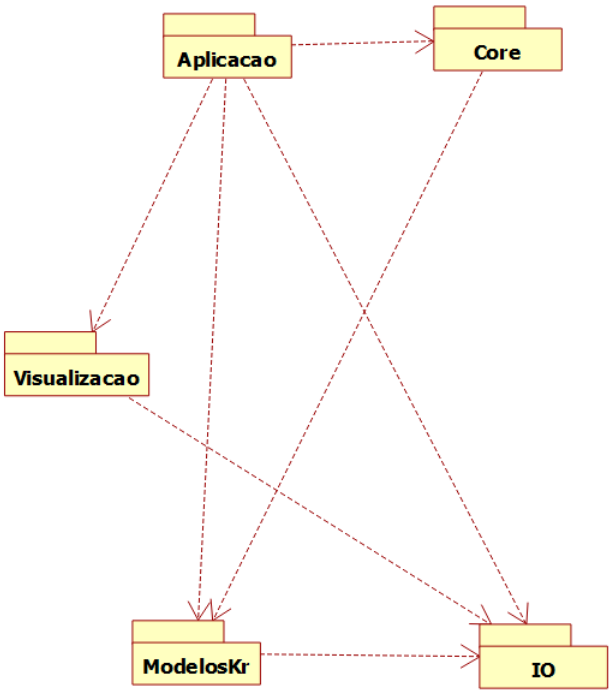

Figura 4.1: Diagrama de componentes.

4.4 Diagrama de implantação

O Diagrama de Implantação (Figura 4.2) para este projeto é simples, pois não se trata de um sistema distribuído. Haverá um único nó principal: `<<device>>` Computador Pessoal. Este nó de hardware hospeda o sistema operacional (ex: *Windows* 10/11 ou GNU/Linux). Dentro deste S.O., residirão os dois artefatos de software necessários para a execução:

1. `[fw_calc]` : O nosso artefato executável compilado.
2. `[gnuplot]`: O artefato executável da dependência externa. O `[fw_calc]` possui uma dependência de execução (do tipo `<<call>>`) com o `[gnuplot]`.

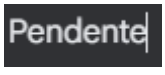

Figura 4.2: Diagrama de implantação.

4.4.1 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
  - Implementar classes Simulador, `CalculadoraFluxoFracionario`.
  - Implementar `ICurvasPermeabilidade` e `CurvasPermeabilidadeTabelada`.
  - Implementar leitor de arquivo de entrada (parser) para o modelo tabelado e viscosidades.
  - Implementar cálculo da curva e salvamento em arquivo `output.csv`.
  - Testes: Validar o `.csv` de saída contra um cálculo manual em planilha.
- v0.2
  - Implementar classe `CurvasPermeabilidadeCorey` [Corey, 1954].
  - Atualizar o parser do arquivo de entrada para ler os parâmetros do modelo Corey .
  - Testes: Validar o `.csv` de saída para o modelo Corey.
- v0.3
  - Implementar classe Gnuplot .
  - Integrar a chamada do Gnuplot no Simulador para plotagem automática.
  - Testes: Validar a exibição do gráfico em *Windows* e Linux.

4.4.2 Tabela classificação sistema

A Tabela 4.1 a seguir é utilizada para classificação do sistema desenvolvido.

Tabela 4.1: Tabela classificação sistema.

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 [] baixo nível
	implementação: [ ] recursivo ou [X] iterativo; [X] determinístico ou [ ] não-determinístico; [ ] exato ou [X] aproximado
	concorrências: [X] serial [] 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 [] serial otimizado [] concorrente [] paralelo [ ] vetorial
	interface do usuário: [ ] kernel numérico [X] 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*
	[X] 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, excessões. 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* [] *threads* [ ] *expressões regulares* [ ] *múltiplos processos*
Bibliotecas externas:	[X] `CGnuplot` [] `QCustomPlot` [ ] `Qt diálogos` [ ] QT Janelas/menus/BT_____
Ferramentas auxiliares:	Montador: [X] make [] cmake [] qmake
IDE:	[X] Editor simples: kate/gedit/emacs [ ] kdevelop [ ] QT-Creator [ ] _____
SCV:	[ ] cvs [ ] svn [X] git
Disciplinas correlacionadas	[ ] estatística [X] cálculo numérico [ ] modelamento numérico [] análise e processamento de imagens

Capítulo 5

Ciclos de Planejamento/Detalhamento

Apresenta-se neste capítulo os ciclos de planejamento/detalhamento para as diferentes versões desenvolvidas. O projeto será dividido em quatro ciclos de desenvolvimento e validação.

5.1 Ciclo 1 (Versão 0.1) - MVP: Núcleo de Cálculo e Modelo Tabular

Este ciclo inicial foca em construir o Produto Mínimo Viável (MVP). O objetivo é validar a arquitetura central e a lógica de engenharia principal (a equação de Buckley-Leverett). Ao final deste ciclo, teremos um software de linha de comando que lê um arquivo `.txt` de um modelo tabelado, calcula a curva de fluxo fracionário e salva um arquivo `.csv` com o resultado.

- Funcionalidades (`features`) implementadas:
  - Implementação da `main()` (no `Simulador.cpp`) para orquestrar o processo.
  - Implementação do parser de arquivo de entrada (no pacote IO) para ler `VISC_OLEO` , `VISC_AGUA` e os dados do `MODELO_KR TABELADO` .
  - Implementação da classe `CalculadoraFluxoFracionario` .
  - Implementação da interface `ICurvasPermeabilidade` .
  - Implementação da classe `CurvasPermeabilidadeTabelada` , incluindo o algoritmo de interpolação linear.
- Teste de Validação do Ciclo 1:
  - O software deve ser executado com o arquivo de entrada `Teste-01.in`. O resultado `output.csv` gerado deve ser comparado, via `Excel`, com

uma planilha de cálculo manual usando os mesmos dados de entrada. Os resultados devem ser idênticos.

5.2 Ciclo 2 (Versão 0.2) - Extensão: Modelo Analítico Corey

Com o núcleo de cálculo validado, este ciclo foca em demonstrar a extensibilidade do software, conforme projetado no Diagrama de Classes (padrão *Strategy*). O objetivo é adicionar uma nova forma de cálculo (Correlação de Corey [Corey, 1954]) sem alterar o núcleo (`CalculadoraFluxoFracionario`).

- Funcionalidades (`features`) implementadas:
  - Implementação da classe `CurvasPermeabilidadeCorey`, herdando de `ICurvasPermeabilidade`.
  - Implementação do algoritmo matemático das equações de Corey. Atualização do parser de IO para reconhecer a palavra-chave: `MODELO_KR COREY` , e ler os seis parâmetros de entrada.
  - Modificação no Simulador para instanciar o objeto `CurvasPermeabilidadeCorey` quando a palavra-chave for encontrada.
- Teste de Validação do Ciclo 2:
  - O software será executado com o arquivo `Teste-02.in`. Como o modelo de Corey é analítico, os resultados no `output.csv` serão validados comparando-os com uma planilha que implementa as mesmas fórmulas de Corey, garantindo a precisão matemática da implementação.

5.3 Ciclo 3 (Versão 0.3) - Validação Científica: Tarek Ahmed

Este ciclo é dedicado exclusivamente à validação científica da ferramenta. O objetivo não é adicionar código novo, mas sim provar que o modelo numérico (tabelado) consegue reproduzir resultados clássicos da literatura de engenharia de petróleo.

- Atividades Realizadas:
  - Levantamento de dados do Exemplo 14-5 do livro *Reservoir Engineering Handbook* de Tarek Ahmed [Ahmed, 2010].
  - Criação do arquivo de entrada específico Teste-Tarek-1cp.in contendo os dados de permeabilidade e viscosidade extraídos do livro.
  - Execução e análise comparativa.

- Teste de Validação do Ciclo 3:
  - O software será executado com o Teste-Tarek-1cp.in. Os resultados no `output.csv` serão validados ponto a ponto contra a tabela de resultados publicada na página 950 do livro de Tarek Ahmed [Ahmed, 2010], como detalhado no Capítulo 7.

5.4 Ciclo 4 (Versão 0.4) - Apresentação: Integração Gnuplot

Este ciclo final foca na camada de apresentação e na usabilidade da ferramenta. O objetivo é implementar a *Facade* Gnuplot para fornecer *feedback* visual imediato ao usuário, completando o requisito RF-05.

- Funcionalidades (`features`) implementadas:
  - Implementação da classe Gnuplot como uma classe estática utilitária.
  - Implementação do método `Gnuplot::plotarCurva()`, que cria os arquivos temporários de dados e *script*, e invoca o `system()` para chamar o Gnuplot.
  - Integração da chamada `Gnuplot::plotarCurva()` no Simulador.
- Teste de Validação do Ciclo 4:
  - O software foi executado com sucesso utilizando os arquivos de teste anteriores, e a janela do Gnuplot com o gráfico foi exibida automaticamente em todos os casos, validando a integração.

Capítulo 6

Ciclos Construção - Implementação

6.1 Código fonte

Apresenta-se a seguir o conjunto de classes (arquivos `.h` e `.cpp`) que compõem a arquitetura do software, conforme o Diagrama de Classes (3.1).

- Interface

Apresenta-se na Listagem 6.1 o arquivo de cabeçalho da interface `ICurvasPermeabilidade`. Esta classe base abstrata define o contrato para todos os modelos de permeabilidade, sendo o pilar do Padrão de Projeto *Strategy* (Nota: Esta classe é puramente virtual, portanto não possui um arquivo `.cpp`).

- Classe `CurvasPermeabilidadeTabelada`

  Apresenta-se na listagem 6.2 o arquivo de cabeçalho da classe `CurvasPermeabilidadeTabelada`, que implementa a interface para dados tabulares.

  Apresenta-se na listagem 6.3 a implementação da classe, incluindo o algoritmo de interpolação linear.

- Classe `CurvasPermeabilidadeCorey`

  Apresenta-se na listagem 6.4 o arquivo de cabeçalho da classe `CurvasPermeabilidadeCorey`, que implementa a interface para o modelo analítico de Corey [Corey, 1954].

  Apresenta-se na listagem 6.5 a implementação da classe, contendo a formulação matemática do modelo.

- Classe `CalculadoraFluxoFracionario`

  Apresenta-se na listagem 6.6 o arquivo de cabeçalho da classe `CalculadoraFluxoFracionario`. Esta é a classe central do domínio de negócio, responsável por implementar a equação de Buckley-Leverett [Buckley and Leverett, 1942].

Listagem 6.1: ICurvasPermeabilidade.h.

```
#ifndef ICURVASPERMEABILIDADE_H
#define ICURVASPERMEABILIDADE_H

#include <string>

/**
 * @class ICurvasPermeabilidade
 * @brief Interface (classe base abstrata) para os modelos de
    permeabilidade relativa.
 *
 * Esta classe define o "contrato" que todos os modelos de Kr (
    Tabelado, Corey, etc.)
 * devem seguir. Ela é a base do Padrão de Projeto Strategy,
    permitindo que a
 * Calculadora opere em um alto nível de abstração (Inversão de
     Dependência).
 */
class ICurvasPermeabilidade {
public:
    /**
     * @brief Destrutor virtual padrão, essencial para classes
        base polimórficas.
     */
    virtual ~ICurvasPermeabilidade() {}

    /**
     * @brief Método virtual puro para carregar dados de um
        arquivo.
     *
     * Cada classe filha deve implementar este método para ler
        seus
     * parâmetros específicos do arquivo de entrada (ex: ler a
        tabela ou ler os parâmetros de Corey).
     * @param arquivo O caminho (path) para o arquivo de
        configuração .txt.
     */
    virtual void carregarDados(const std::string& arquivo) = 0;

    /**
     * @brief Obtém a permeabilidade relativa da água (Krw).
     * @param sw A saturação de água (Sw) para a qual o Krw
        será calculado.
     * @return O valor de Krw (entre 0 e 1).
     */
    virtual double getKrw(double sw) const = 0;

    /**
     * @brief Obtém a permeabilidade relativa do óleo (Kro).
     * @param sw A saturação de água (Sw) para a qual o Kro
        será calculado.
     * @return O valor de Kro (entre 0 e 1).
     */
    virtual double getKro(double sw) const = 0;
};

#endif
```

Listagem 6.2: CurvasPermeabilidadeTabelada.h.

```
#ifndef CURVASPERMEABILIDADETABELADA_H
#define CURVASPERMEABILIDADETABELADA_H

#include "ICurvasPermeabilidade.h"
#include <vector>
#include <string> // Incluído para std::string

/**
 * @class CurvasPermeabilidadeTabelada
 * @brief Implementação concreta da interface
     ICurvasPermeabilidade para dados tabulados.
 *
 * Esta classe lê uma tabela de Sw, Krw e Kro de um arquivo de
     entrada e usa
 * interpolação linear para calcular valores intermediários.
 */
class CurvasPermeabilidadeTabelada : public
    ICurvasPermeabilidade {
private:
    /// Vetor com os valores de Saturação de Água da tabela.
    std::vector<double> _sw;

    /// Vetor com os valores de Krw da tabela.
    std::vector<double> _krw;

    /// Vetor com os valores de Kro da tabela.
    std::vector<double> _kro;

    /**
     * @brief Algoritmo de Interpolação Linear (e extrapolação
         de ponta).
     * Este é o algoritmo detalhado no Diagrama de Atividades.
     * @param x_desejado A saturação (Sw) que queremos.
     * @param vec_x O vetor de Saturações da tabela (_sw).
     * @param vec_y O vetor de Krw ou Kro correspondente.
     * @return O valor de y (Kr) interpolado.
     */
    double interpolar(double x_desejado, const std::vector<
        double>& vec_x, const std::vector<double>& vec_y) const;

public:
    /**
     * @brief Carrega os dados da tabela (bloco DADOS_KR_INICIO
         ) do arquivo.
     * @param arquivo O caminho para o arquivo de configuração
         .txt.
     */
    void carregarDados(const std::string& arquivo) override;

    /**
     * @brief Obtém o Krw, usando interpolação se necessário.
     * @param sw A saturação de água.
     * @return O valor de Krw interpolado.
     */
    double getKrw(double sw) const override;

    /**
     * @brief Obtém o Kro, usando interpolação se necessário.
     * @param sw A saturação de água.
```

Listagem 6.3: CurvasPermeabilidadeTabelada.cpp.

```
#include "CurvasPermeabilidadeTabelada.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <stdexcept>

/**
 * @brief Carrega os dados tabelados (Sw, Krw, Kro) do arquivo
     de entrada.
 * @param arquivo O caminho para o arquivo de configuração .txt
     .
 */
void CurvasPermeabilidadeTabelada::carregarDados(const std::
    string& arquivo) {
    std::ifstream arq(arquivo);
    if (!arq.is_open()) {
        throw std::runtime_error("Erro␣(Tabelado):␣Nao␣foi␣
            possivel␣abrir␣o␣arquivo:␣" + arquivo);
    }

    std::string linha;
    bool lendoDados = false;

    while (std::getline(arq, linha)) {
        if (linha.empty() || linha[0] == '#') continue;

        std::stringstream ss(linha);
        std::string palavraChave;
        ss >> palavraChave; // Lê a primeira "palavra"

        if (palavraChave == "DADOS_KR_INICIO") {
            lendoDados = true;
            continue;
        }

        if (palavraChave == "FIM_DADOS") {
            lendoDados = false;
            break;
        }

        if (lendoDados) {
            // Se estamos lendo, a "palavraChave" é na verdade
                o valor do Sw
            std::stringstream ss_linha(linha);
            double sw, krw, kro;
            ss_linha >> sw >> krw >> kro;

            // Adiciona os valores aos vetores da classe
            _sw.push_back(sw);
            _krw.push_back(krw);
            _kro.push_back(kro);
        }
    }

    if (_sw.empty()) {
        throw std::runtime_error("Erro:␣Nenhum␣dado␣de␣
            permeabilidade␣encontrado␣(bloco␣DADOS_KR_INICIO...
            FIM_DADOS)␣no␣arquivo.");
    }
```

Listagem 6.4: CurvasPermeabilidadeCorey.h.

```
#ifndef CURVASPERMEABILIDADECOREY_H
#define CURVASPERMEABILIDADECOREY_H

#include "ICurvasPermeabilidade.h"
#include <string> // Incluído para std::string

/**
 * @class CurvasPermeabilidadeCorey
 * @brief Implementação concreta da interface
     ICurvasPermeabilidade para o modelo de Corey.
 *
 * Esta classe lê os 6 parâmetros do modelo de Corey do arquivo
      de entrada
 * e calcula Krw/Kro analiticamente usando as fórmulas de Corey
     .
 */
class CurvasPermeabilidadeCorey : public ICurvasPermeabilidade
    {
private:
    /// Saturação de água irreduzível (Swir)
    double _swir;

    /// Saturação de óleo residual (Sorw)
    double _sorw;

    /// Permeabilidade relativa máxima da água (no Sorw)
    double _krw_max;

    /// Permeabilidade relativa máxima do óleo (no Swir)
    double _kro_max;

    /// Expoente de Corey para a água (nw)
    double _nw;

    /// Expoente de Corey para o óleo (no)
    double _no;

public:
    /**
     * @brief Carrega os 6 parâmetros do modelo Corey do
         arquivo.
     * @param arquivo O caminho para o arquivo de configuração
         .txt.
     */
    void carregarDados(const std::string& arquivo) override;

    /**
     * @brief Calcula o Krw usando a fórmula de Corey.
     * @param sw A saturação de água.
     * @return O valor de Krw analítico.
     */
    double getKrw(double sw) const override;

    /**
     * @brief Calcula o Kro usando a fórmula de Corey.
     * @param sw A saturação de água.
     * @return O valor de Kro analítico.
     */
    double getKro(double sw) const override;
```

Listagem 6.5: CurvasPermeabilidadeCorey.cpp.

```
#include "CurvasPermeabilidadeCorey.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <stdexcept>
#include <cmath>      // Para std::pow
#include <algorithm> // Para std::max e std::min

/**
 * @brief Carrega os parâmetros do modelo de Corey do arquivo
     de entrada.
 * @param arquivo O caminho para o arquivo de configuração .txt
     .
 */
void CurvasPermeabilidadeCorey::carregarDados(const std::string
    & arquivo) {
    // Seta valores padrão inválidos para checagem
    _swir = _sorw = _krw_max = _kro_max = _nw = _no = -1.0;

    std::ifstream arq(arquivo);
    if (!arq.is_open()) {
        throw std::runtime_error("Erro (Corey): Nao foi
            possivel abrir o arquivo: " + arquivo);
    }

    std::string linha;
    std::string palavraChave;

    while (std::getline(arq, linha)) {
        if (linha.empty() || linha[0] == '#') continue;

        std::stringstream ss(linha);
        ss >> palavraChave;

        // Lê e armazena cada parâmetro
        if (palavraChave == "COREY_SWIR")      ss >> _swir;
        else if (palavraChave == "COREY_SORW")     ss >> _sorw;
        else if (palavraChave == "COREY_KRW_MAX") ss >>
            _krw_max;
        else if (palavraChave == "COREY_KRO_MAX") ss >>
            _kro_max;
        else if (palavraChave == "COREY_NW")       ss >> _nw;
        else if (palavraChave == "COREY_NO")       ss >> _no;
    }

    // Validação simples
    if (_swir < 0 || _sorw < 0 || _krw_max < 0 || _kro_max < 0
        || _nw < 0 || _no < 0) {
        throw std::runtime_error("Erro: Um ou mais parametros
            do modelo Corey nao foram carregados corretamente.")
            ;
    }
    std::cout << "DEBUG: Parametros de Corey carregados com
        sucesso.\n";
}

/**
 * @brief Calcula a Saturação Normalizada (Sw_norm).
 * Privado, apenas para uso interno desta classe.
```

Listagem 6.6: CalculadoraFluxoFracionario.h

```
#ifndef CALCULADORAFLUXOFRACIONARIO_H
#define CALCULADORAFLUXOFRACIONARIO_H

#include "ICurvasPermeabilidade.h"
#include <map>
#include <string> // Incluído para std::string

/**
 * @class CalculadoraFluxoFracionario
 * @brief Classe principal que implementa o "domínio de negócio" (a lógica de engenharia).
 *
 * Esta classe é responsável por implementar a equação de Buckley-Leverett.
 * Ela depende de uma abstração (ICurvasPermeabilidade) e não de uma
 * implementação concreta, seguindo o Princípio da Inversão de Dependência.
 */
class CalculadoraFluxoFracionario {
private:
    /// Viscosidade do Óleo (cPoise)
    double _viscosidadeOleo;

    /// Viscosidade da Água (cPoise)
    double _viscosidadeAgua;

    /// Ponteiro para o modelo de Kr (Strategy Pattern)
    ICurvasPermeabilidade* _modeloKr;

public:
    /**
     * @brief Construtor da Calculadora.
     * Recebe as viscosidades e o modelo de Kr via Injeção de Dependência.
     * @param mu_o Viscosidade do Óleo (cPoise).
     * @param mu_w Viscosidade da Água (cPoise).
     * @param modelo Um ponteiro para um objeto que implementa ICurvasPermeabilidade.
     */
    CalculadoraFluxoFracionario(double mu_o, double mu_w, ICurvasPermeabilidade* modelo);

    /**
     * @brief Calcula um único ponto da curva de fluxo fracionário.
     * @param sw A saturação de água para a qual o Fw será calculado.
     * @return O valor do fluxo fracionário (fw).
     */
    double calcularFw(double sw) const;

    /**
     * @brief Gera a curva completa de Fw vs Sw, iterando sobre a saturação.
     * @param passo O incremento de Saturação (ex: 0.01 para 1%).
     * @return Um mapa (std::map) contendo os pares (Sw, Fw) da
```

Apresenta-se na listagem 6.7 a implementação da classe. Note que ela opera apenas contra a interface `ICurvasPermeabilidade`, garantindo baixo acoplamento.

- Classe Gnuplot

  Apresenta-se na listagem 6.8 o arquivo de cabeçalho da classe utilitária Gnuplot, que atua como uma Facade para a plotagem.

  Apresenta-se na listagem 6.9 a implementação da classe, incluindo a chamada `system()` para o processo externo do Gnuplot.

- Classe Simulador

  Apresenta-se na listagem 6.10 o arquivo de cabeçalho da classe Simulador, que atua como o orquestrador da aplicação.

  Apresenta-se na listagem 6.11 a implementação da classe, responsável por ler o arquivo de entrada, instanciar os objetos corretos e injetar as dependências.

- Programa Principal `main.cpp`

  Apresenta-se na listagem 6.12 o arquivo `main.cpp`, que contém o ponto de entrada do programa. Sua única responsabilidade é instanciar e executar o Simulador.

Listagem 6.7: CalculadoraFluxoFracionario.cpp.

```
#include "CalculadoraFluxoFracionario.h"
#include <cmath>      // Para std::pow
#include <stdexcept> // Para std::runtime_error
#include <limits>     // Para checagem de divisão por zero

/**
 * @brief Construtor da Calculadora.
 * @param mu_o Viscosidade do Óleo.
 * @param mu_w Viscosidade da Água.
 * @param modelo Ponteiro para o modelo de Kr (injetado).
 */
CalculadoraFluxoFracionario::CalculadoraFluxoFracionario(double
    mu_o, double mu_w, ICurvasPermeabilidade* modelo)
: _viscosidadeOleo(mu_o), _viscosidadeAgua(mu_w), _modeloKr(
    modelo) {

    // Validação de segurança
    if (modelo == nullptr) {
        throw std::runtime_error("Erro: Calculadora recebeu um
            modelo de permeabilidade nulo.");
    }
    if (mu_o <= 0 || mu_w <= 0) {
        throw std::runtime_error("Erro: Viscosidades devem ser
            positivas.");
    }
}

/**
 * @brief Calcula o valor do fluxo fracionário (fw).
 * @param sw Saturação de água.
 * @return O valor de fw.
 */
double CalculadoraFluxoFracionario::calcularFw(double sw) const
    {

    // 1. Pedir os valores de Kr para o modelo (Strategy
        Pattern)
    double krw = _modeloKr->getKrw(sw);
    double kro = _modeloKr->getKro(sw);

    // 2. Implementar a Equação de Buckley-Leverett

    // Razão de mobilidade da água: Lambda_w = krw / mu_w
    double lambda_w = krw / _viscosidadeAgua;

    // Razão de mobilidade do óleo: Lambda_o = kro / mu_o
    double lambda_o = kro / _viscosidadeOleo;

    // Mobilidade Total: Lambda_t = Lambda_w + Lambda_o
    double lambda_t = lambda_w + lambda_o;

    // Fórmula do Fluxo Fracionário: fw = Lambda_w / Lambda_t

    // Checagem de segurança para divisão por zero
    if (lambda_t < std::numeric_limits<double>::epsilon()) {
        // Se a mobilidade total é zero (ambos krw e kro são 0)
            ,
        // o fluxo fracionário também é zero.
        return 0.0;
```

Listagem 6.8: Gnuplot.h.

```
#ifndef GNUPLOT_H
#define GNUPLOT_H

#include <map>
#include <string>

/**
 * @class Gnuplot
 * @brief Classe utilitária que implementa o Padrão de Projeto
     Facade.
 *
 * Esta classe esconde toda a complexidade de lidar com o
     Gnuplot
 * (criar arquivos temporários, formatar scripts, chamar o
     system())
 * por trás de um único método estático.
 */
class Gnuplot {
public:
    /**
     * @brief Plota um mapa de dados (x, y) usando o Gnuplot.
     * Este é um método estático, pois não precisa de um estado
         de instância.
     * @param dados Um std::map<double, double> onde a chave é
         o eixo X (Sw) e o valor é o eixo Y (Fw).
     * @param titulo O título que aparecerá no topo do gráfico.
     */
    static void plotarCurva(const std::map<double, double>&
        dados, const std::string& titulo);
};

#endif
```

Listagem 6.9: Gnuplot.cpp.

```
#include "Gnuplot.h"
#include <iostream>
#include <fstream>
#include <cstdlib> // Para system()

// TODO: Documentar com JAVADOC/Doxygen
void Gnuplot::plotarCurva(const std::map<double, double>& dados
    , const std::string& titulo) {
    std::cout << "DEBUG: Chamando Gnuplot...\n";

    // Nomes dos arquivos temporários
    std::string tempDados = "temp_data.csv";
    std::string tempScript = "temp_script.gp";

    // --- 1. Salvar dados no arquivo .csv ---
    std::ofstream arqDados(tempDados);
    if (!arqDados.is_open()) {
        std::cerr << "Erro: Nao foi possivel criar arquivo de
            dados temporario.\n";
        return;
    }
    arqDados << "# Sw, Fw\n";
    for (const auto& par : dados) {
        arqDados << par.first << ", " << par.second << "\n";
    }
    arqDados.close();

    // --- 2. Criar script do Gnuplot ---
    std::ofstream arqScript(tempScript);
    if (!arqScript.is_open()) {
        std::cerr << "Erro: Nao foi possivel criar script
            Gnuplot temporario.\n";
        return;
    }
    arqScript << "set title '" << titulo << "'\n";
    arqScript << "set xlabel 'Saturacao de Agua (Sw)'\n";
    arqScript << "set ylabel 'Fluxo Fracionario de Agua (Fw)'\n
        ";
    arqScript << "set grid\n";
    arqScript << "set key top left\n";
    arqScript << "set datafile separator ','\n";
    arqScript << "plot '" << tempDados << "' with lines title '
        Curva Fw'\n";
    arqScript << "pause -1 'Pressione Enter para fechar'\n"; //
         Mantém a janela aberta
    arqScript.close();

    // --- 3. Executar Gnuplot ---
    // Este comando supõe que 'gnuplot' está no PATH do sistema
    std::string comando = "gnuplot " + tempScript;
    std::system(comando.c_str());

    // --- 4. (Opcional) Limpar arquivos temporários ---
    // std::remove(tempDados.c_str());
    // std::remove(tempScript.c_str());
}
```

Listagem 6.10: Simulador.h.

```
#ifndef SIMULADOR_H
#define SIMULADOR_H

#include <string>

/**
 * @class Simulador
 * @brief O orquestrador da aplicação.
 *
 * Esta classe é responsável por controlar o fluxo de execução
    do programa:
 * ler o arquivo de entrada, instanciar os objetos corretos
 * (fazendo o 'new' dos modelos) e coordenar as chamadas
 * para a calculadora e o plotter.
 */
class Simulador {
public:
    /**
     * @brief Ponto de entrada principal da lógica do simulador
        .
     * @param arquivoEntrada O caminho (path) para o arquivo de
         configuração .txt.
     */
    void executar(const std::string& arquivoEntrada);
};

#endif
```

Listagem 6.11: Simulador.cpp

```
#include "Simulador.h"
#include "CalculadoraFluxoFracionario.h"
#include "CurvasPermeabilidadeTabelada.h"
#include "CurvasPermeabilidadeCorey.h"
#include "Gnuplot.h"

#include <iostream>
#include <fstream>   // Para ler arquivos (ifstream)
#include <sstream>   // Para processar strings (stringstream)
#include <stdexcept> // Para lançar erros (runtime_error)
#include <map>

/**
 * @brief Executa a simulação completa.
 * * Este método orquestra todo o processo:
 * 1. Lê o arquivo de entrada para definir os parâmetros.
 * 2. Instancia o modelo de permeabilidade correto (Tabelado ou
     Corey).
 * 3. Delega o carregamento de dados detalhados para o modelo.
 * 4. Instancia a calculadora.
 * 5. Gera a curva de fluxo fracionário.
 * 6. Chama o Gnuplot para exibir o resultado.
 * * @param arquivoEntrada O caminho para o arquivo de
    configuração .txt.
 */
void Simulador::executar(const std::string& arquivoEntrada) {
    std::cout << "Iniciando simulador...\n";

    // --- 1. Leitura e Parsing do Arquivo de Entrada ---

    double mu_o = -1.0;
    double mu_w = -1.0;
    std::string tipoModelo;
    ICurvasPermeabilidade* modelo = nullptr;

    std::ifstream arq(arquivoEntrada);
    if (!arq.is_open()) {
        throw std::runtime_error("Erro fatal: Nao foi possivel
            abrir o arquivo: " + arquivoEntrada);
    }

    std::cout << "Lendo arquivo de configuracao: " <<
        arquivoEntrada << "\n";
    std::string linha;
    std::string palavraChave;

    while (std::getline(arq, linha)) {
        // Ignora linhas vazias ou comentários
        if (linha.empty() || linha[0] == '#') {
            continue;
        }

        std::stringstream ss(linha);
        ss >> palavraChave;

        if (palavraChave == "VISC_OLEO") {
            ss >> mu_o;
        } else if (palavraChave == "VISC_AGUA") {
            ss >> mu_w;
```

Listagem 6.12: main.cpp.

```
#include "Simulador.h"
#include <iostream>

int main(int argc, char* argv[]) {
    // Verifica se o usuário passou o nome do arquivo de
        entrada
    if (argc < 2) {
        std::cerr << "Erro: Por favor, forneça o nome do
            arquivo de entrada.\n";
        std::cerr << "Uso: ./fw_calc <
            caminho_para_arquivo_de_entrada>\n";
        return 1; // Retorna um código de erro
    }

    std::string arquivoEntrada = argv[1];

    try {
        Simulador sim;
        sim.executar(arquivoEntrada);
    } catch (const std::exception& e) {
        std::cerr << "Uma exceção ocorreu: " << e.what() << '\n
            ';
        return 1;
    }

    return 0; // Sucesso
}
```

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: Validando o Ciclo 1 - Modelo Tabelado

Descrição do Teste:

- O que está sendo testado? Este teste valida o funcionamento básico do núcleo numérico e o algoritmo de interpolação linear. O objetivo é garantir que o software consiga ler uma tabela de permeabilidade relativa genérica e calcular valores intermediários corretamente.
- Como será realizado? O software é executado com o arquivo de entrada `test/Teste-01.in`, que contém uma tabela simplificada com 7 pontos de k_{rw} e k_{ro}.
- Como será validado? A validação é feita pela análise da curva gerada no arquivo `output.csv`. Verifica-se se a curva é contínua e se os pontos calculados correspondem a uma interpolação linear suave entre os pontos de entrada, sem saltos ou descontinuidades numéricas.

7.1.1 Arquivo de entrada de dados

Apresenta-se a seguir o arquivo de entrada `Teste-01.in`, Listagem 7.1, que utiliza o modelo tabelado.

Listagem 7.1: Teste-01.in.

```
# Exemplo de arquivo de entrada para modelo tabelado
VISC_OLEO 1.5
VISC_AGUA 0.8
MODELO_KR TABELADO #SW/KRW/KRO
DADOS_KR_INICIO
0.20  0.00  0.90
0.30  0.05  0.75
0.40  0.12  0.50
0.50  0.20  0.30
0.60  0.30  0.15
0.70  0.40  0.05
0.80  0.50  0.00
FIM_DADOS
```

7.1.2 Arquivo de saída de dados (resultados da tela)

O software é de linha de comando e não gera saída na tela, exceto por mensagens de status ou erro. O resultado principal é o arquivo `output.csv`, que conterá duas colunas: S_w e f_w.

7.1.3 Figuras

A Figura 7.1 mostra a execução do programa no terminal. A Figura 7.2 mostra o gráfico gerado a partir do `output.csv`, comprovando a geração da curva "S" característica.

A Figura 7.2 apresenta o gráfico gerado. A curva suavizada confirma que a interpolação linear entre os pontos fornecidos está funcionando corretamente, sem descontinuidades.

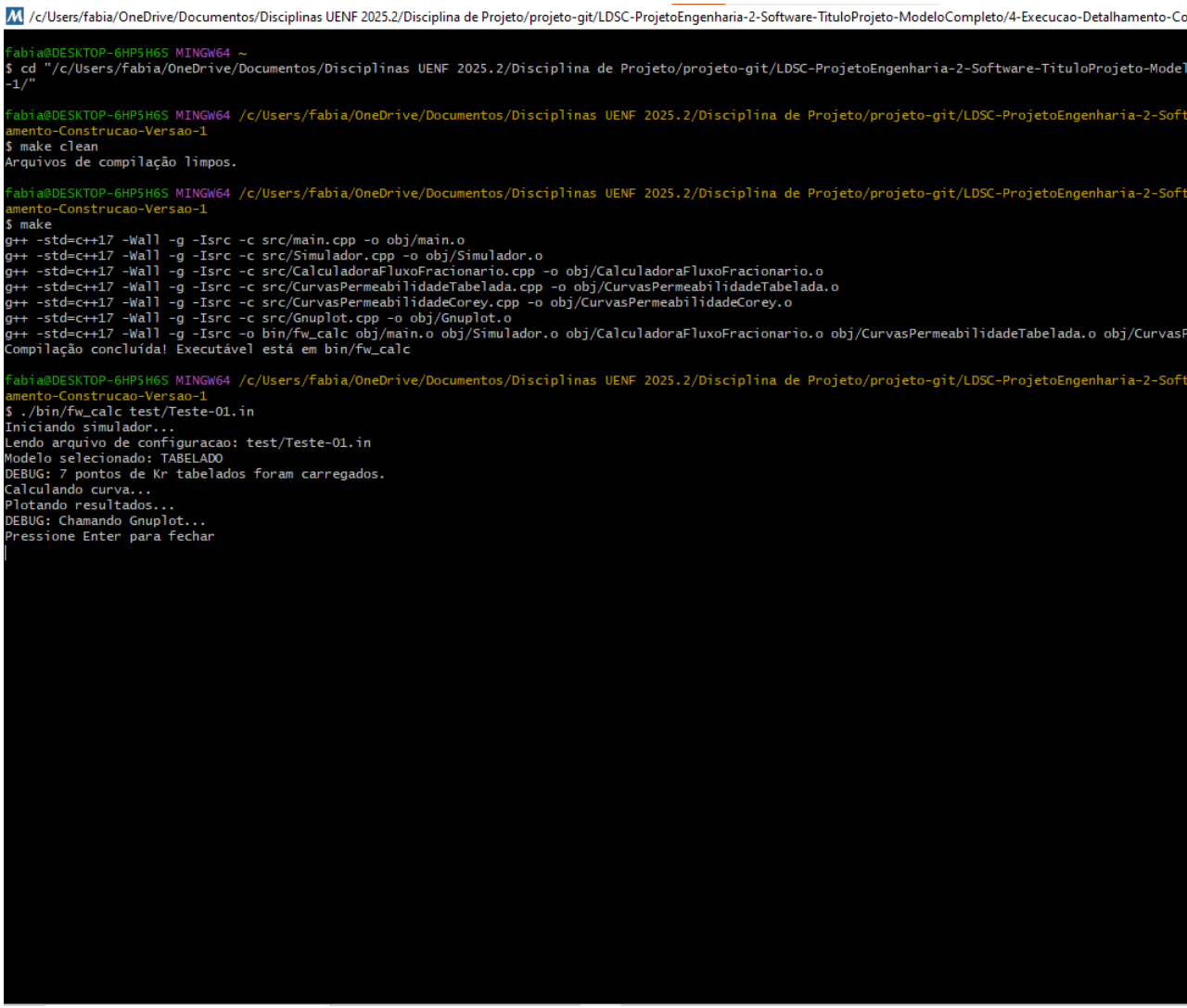

Figura 7.1: Tela do programa mostrando terminal executando.

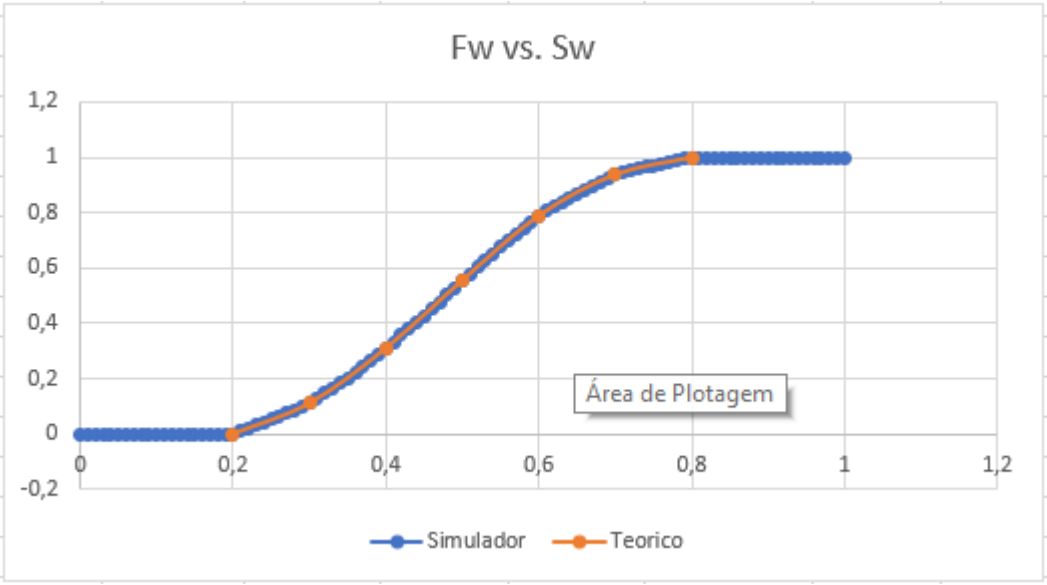

Figura 7.2: Tela do programa mostrando gráfico (*Excel*) gerado a partir do `output.csv`.

7.2 Teste 2: Validando o Ciclo 2 - Modelo Corey

Descrição do Teste:

- O que está sendo testado? Este teste valida a arquitetura de software (Padrão *Strategy*) e a implementação matemática do modelo analítico de Corey. O objetivo é provar que o sistema consegue trocar de modelo de cálculo dinamicamente apenas alterando o arquivo de entrada.
- Como será realizado? O software é executado com o arquivo `test/Teste-02.in`, que utiliza a palavra-chave `MODELO_KR COREY` e define os expoentes $n_w = 2.0$ e $n_o = 2.5$.
- Como será validado? A validação é visual e analítica. A curva resultante deve apresentar o perfil exponencial característico das equações de Corey, diferindo das linhas retas do modelo tabelado. Além disso, os pontos finais (saturação irreduzível de água e óleo residual) devem coincidir exatamente com os parâmetros S_{wir} e S_{orw} fornecidos no arquivo.

7.2.1 Arquivo de entrada de dados

Apresenta-se a seguir o arquivo de entrada `Teste-02.in`, Listagem 7.2.

Listagem 7.2: Teste-02.in.

```
# Exemplo de arquivo de entrada para modelo Corey
VISC_OLEO 2.0
VISC_AGUA 1.0
MODELO_KR COREY
COREY_SWIR      0.15
COREY_SORW      0.20
COREY_KRW_MAX   0.5
COREY_KRO_MAX   0.9
COREY_NW        2.0
COREY_NO        2.5
```

7.2.2 Arquivo de saída de dados (resultados da tela)

Similar ao Teste 1, o resultado principal é o arquivo `output.csv`.

7.2.3 Figuras

A Figura 7.3 mostra o resultado gráfico. A curva apresenta o comportamento exponencial característico do modelo de Corey, confirmando a correta implementação das equações de potência.

A curva gerada apresenta a concavidade característica das funções de potência, confirmando que os expoentes $n_w = 2.0$ e $n_o = 2.5$ foram aplicados corretamente na fórmula analítica.

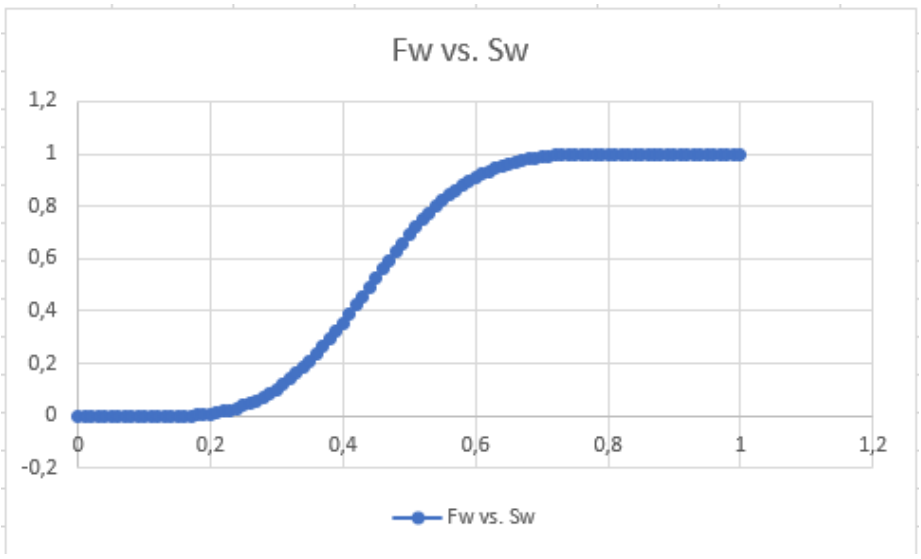

Figura 7.3: Tela do programa mostrando curva do modelo de Corey.

7.3 Teste 3: Validação com Literatura (Tarek Ahmed)

Descrição do Teste

- O que está sendo testado? Este é o teste de validação científica do projeto. O objetivo é comprovar a precisão física dos resultados comparando-os com um problema clássico da literatura de engenharia de reservatórios.
- Como será realizado? O software é executado com o arquivo `test/Teste-Tarek-1cp.in`. Este arquivo reproduz os parâmetros exatos do Exemplo 14-5 do livro *Reservoir Engineering Handbook* de Tarek Ahmed [Ahmed, 2010], considerando a viscosidade do óleo de 1.0 cp.
- Como será validado?
    1. Numérica: Os valores do arquivo de saída `output.csv` são comparados com a coluna "$\mu_o = 1.0$" da tabela de resultados na página 950 do livro 1. O erro absoluto deve ser desprezível.
    2. Gráfica: A curva gerada (Figura 7.4) é comparada visualmente com a Figura 7.5 do livro, buscando a reprodução fiel da curva "S".

7.3.1 Arquivo de entrada de dados

Apresenta-se a seguir o conteúdo do arquivo `Teste-Tarek-1cp.in` utilizado para esta simulação, Listagem 7.3.

7.3.2 Figuras

A Figura 7.4 apresenta o gráfico gerado automaticamente pelo software após a execução do teste.

A análise comparativa com a Figura 7.5 demonstra que a curva obtida reproduz com exatidão o comportamento esperado descrito por Ahmed [Ahmed, 2010]. Os pontos de saturação inicial de água ($S_{wi} \approx 0.24$) e saturação de óleo residual ($S_{or} \approx 0.22$, ou seja, $S_w \approx 0.78$) coincidem perfeitamente com os limites da tabela de entrada, e a curvatura "S" reflete corretamente a variação da razão de mobilidade ao longo do domínio de saturação. Isso confirma que o núcleo numérico da ferramenta é robusto e confiável para aplicações de engenharia.

Listagem 7.3: Teste-Tarek-1cp.in.

```
# Teste de Validação - Exemplo 14-5 do Tarek Ahmed
# Viscosidade do Oleo = 1.0 cp
# Viscosidade da Agua = 0.5 cp

VISC_OLEO 1.0
VISC_AGUA 0.5

MODELO_KR TABELADO

# Dados lidos da Figura 14-15 (Tabela pg. 950)
DADOS_KR_INICIO
0.24 0.00 0.95
0.30 0.01 0.89
0.40 0.04 0.74
0.50 0.09 0.45
0.60 0.17 0.19
0.65 0.28 0.12
0.70 0.22 0.06
0.75 0.36 0.03
0.78 0.41 0.00
FIM_DADOS
```

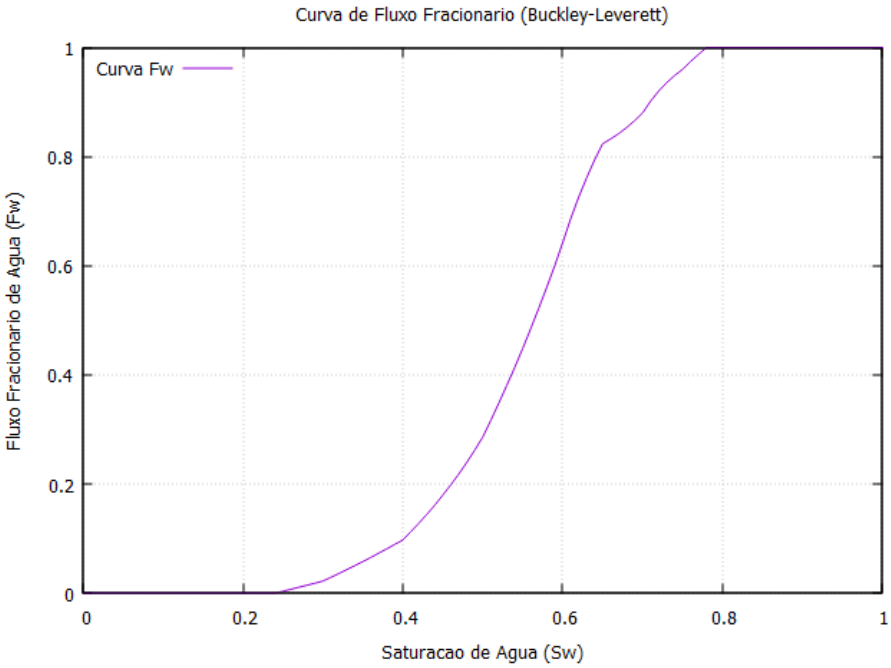

Figura 7.4: Tela do programa mostrando curva do Exemplo 14-5 do livro *Reservoir Engineering Handbook* de Tarek Ahmed [Ahmed, 2010] (com $\mu_o = 1.0$).

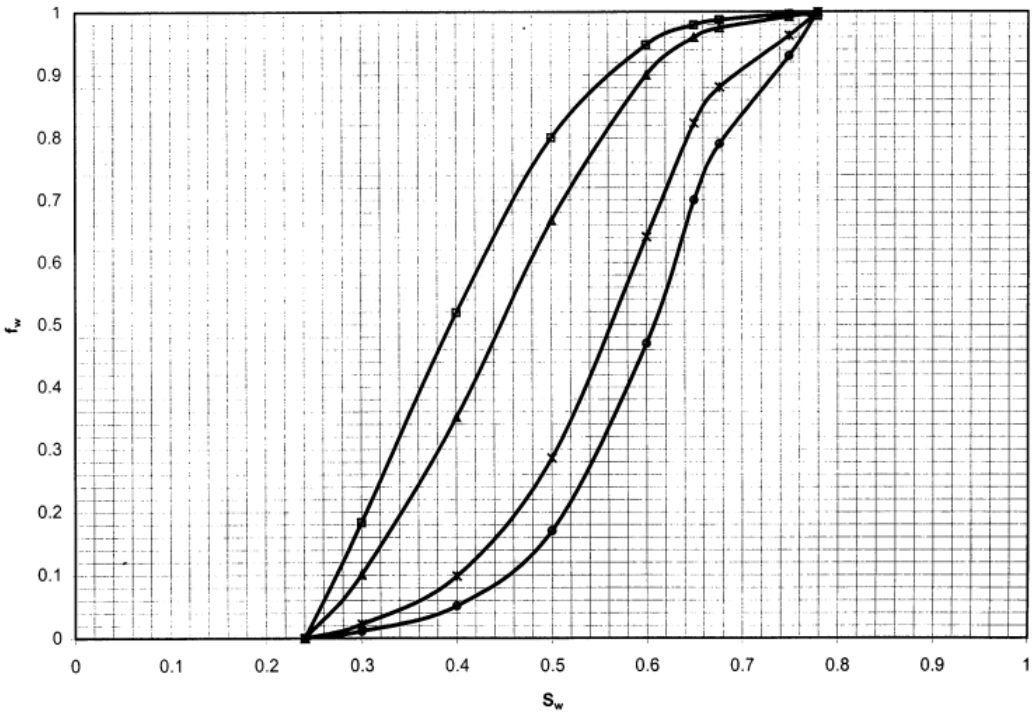

Figure 14-16. Effect of μ_o on f_w.

Figura 7.5: Curva do Exemplo 14-5 do livro *Reservoir Engineering Handbook* de Tarek Ahmed [Ahmed, 2010] (com $\mu_o = 0.5$,$\mu_o = 1.0$,$\mu_o = 5$e $\mu_o = 10$).

Observar que, a Figura 7.4representa a curva com viscosidade do óleo igual a $10(\mu_o = 10)$, para comparação com a devida curva da imagem 7.5.

7.4 Teste 4: Validação da Camada de Apresentação (Gnuplot)

Descrição do Teste:

- O que está sendo testado? Este teste valida o requisito não funcional de usabilidade e a integração com a ferramenta externa Gnuplot (Padrão Facade).
- Como será realizado? O teste consiste na execução bem-sucedida de qualquer um dos cenários anteriores em um ambiente *Desktop* (*Windows* ou

Linux).

- Como será validado? O teste é considerado aprovado se, após o cálculo, uma janela gráfica do Gnuplot for aberta automaticamente, exibindo a curva correta com títulos, eixos nomeados e grade (*grid*) ativada, sem a necessidade de intervenção manual do usuário para plotar os dados. A persistência da janela (ela não deve fechar sozinha imediatamente) também é um critério de sucesso.

7.4.1 Arquivo de entrada de dados

Este teste reutiliza o arquivo Teste-01.in. Além da saída padrão (`output.csv`), o resultado esperado é uma ação do sistema: a abertura de uma janela de gráfico.

7.4.2 Figuras

A Figura 7.6 mostra uma captura de tela a janela do Gnuplot que foi aberta automaticamente pelo software, exibindo o resultado gráfico.

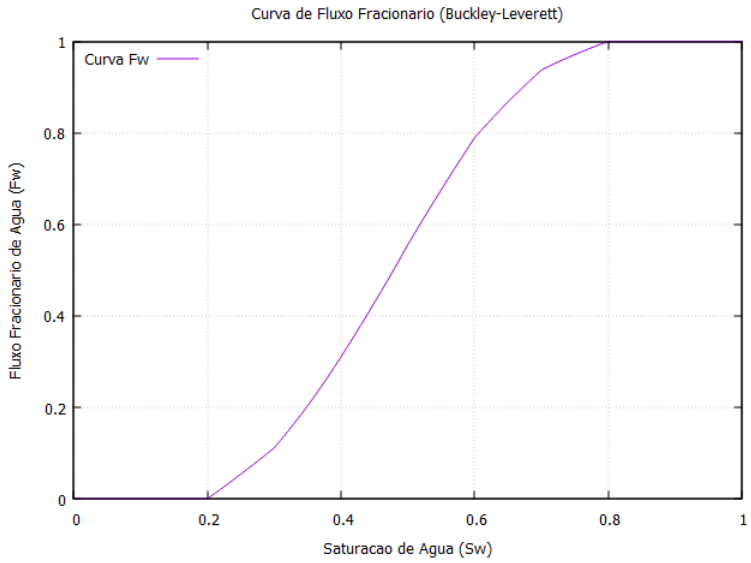

Figura 7.6: Tela do programa mostrando captura de tela a janela do Gnuplot.

Capítulo 8

Documentação para o Desenvolvedor

8.1 Dependências para compilar o software

Para compilar e executar o software "Ferramenta de Fluxo Fracionário", é necessário configurar um ambiente de desenvolvimento C++ compatível. O projeto foi desenvolvido e testado no ambiente MSYS2 (*MinGW* 64-bit) no *Windows*.

As seguintes dependências são obrigatórias:

1. Compilador C++ (g++): Deve suportar o padrão C++17.
2. GNU *Make*: Ferramenta de automação de compilação utilizada pelo `Makefile` do projeto.
3. Gnuplot: Ferramenta de plotagem externa, necessária para a visualização dos gráficos (requisito de execução).
4. Doxygen (Opcional): Para geração da documentação automática.

- Instalação das Dependências (via MSYS2):

No terminal `MINGW64`, execute os seguintes comandos, Listagem 8.1, para instalar todas as ferramentas necessárias:

Listagem 8.1: Comando para instalar Compilador, Make e Gnuplot no MSYS2..

```
pacman -S mingw-w64-x86_64-toolchain mingw-w64-x86_64-
    gnuplot
```

Listagem 8.2: Instalação do Doxygen.

```
pacman -S mingw-w64-x86_64-doxygen
```

Listagem 8.3: Gerando a documentação HTML.

```
doxygen
```

8.2 Como gerar a documentação usando doxygen

A documentação do código-fonte foi escrita utilizando o padrão Javadoc. Isso permite que a ferramenta Doxygen analise os arquivos `.h` e gere um site HTML navegável com a descrição de todas as classes e métodos.

- Passo a Passo para Gerar a Documentação:
- Instale o Doxygen: Caso não tenha instalado, use o comando: `gen}`
- Navegue até a pasta do código: Abra o terminal na raiz do projeto e entre na pasta `src/`
- Execute o Doxygen, Listagem 8.2: O arquivo de configuração Doxyfile já está configurado no projeto. Basta rodar o comando:
- Visualize o Resultado, Listagem 8.3: A documentação será gerada na pasta `html/`. Para visualizar, abra o arquivo `index.html` no seu navegador:

A Figura 8.1 ilustra a página inicial da documentação gerada, onde é possível navegar pela hierarquia de classes - Figura 8.2 - e ver os diagramas de colaboração. Na figura 8.3 a página para `CalculadoraFluxoFracionario` *Class Reference* e na Figura 8.4 o arquivo `CalculadoraFluxoFracionario.h`.

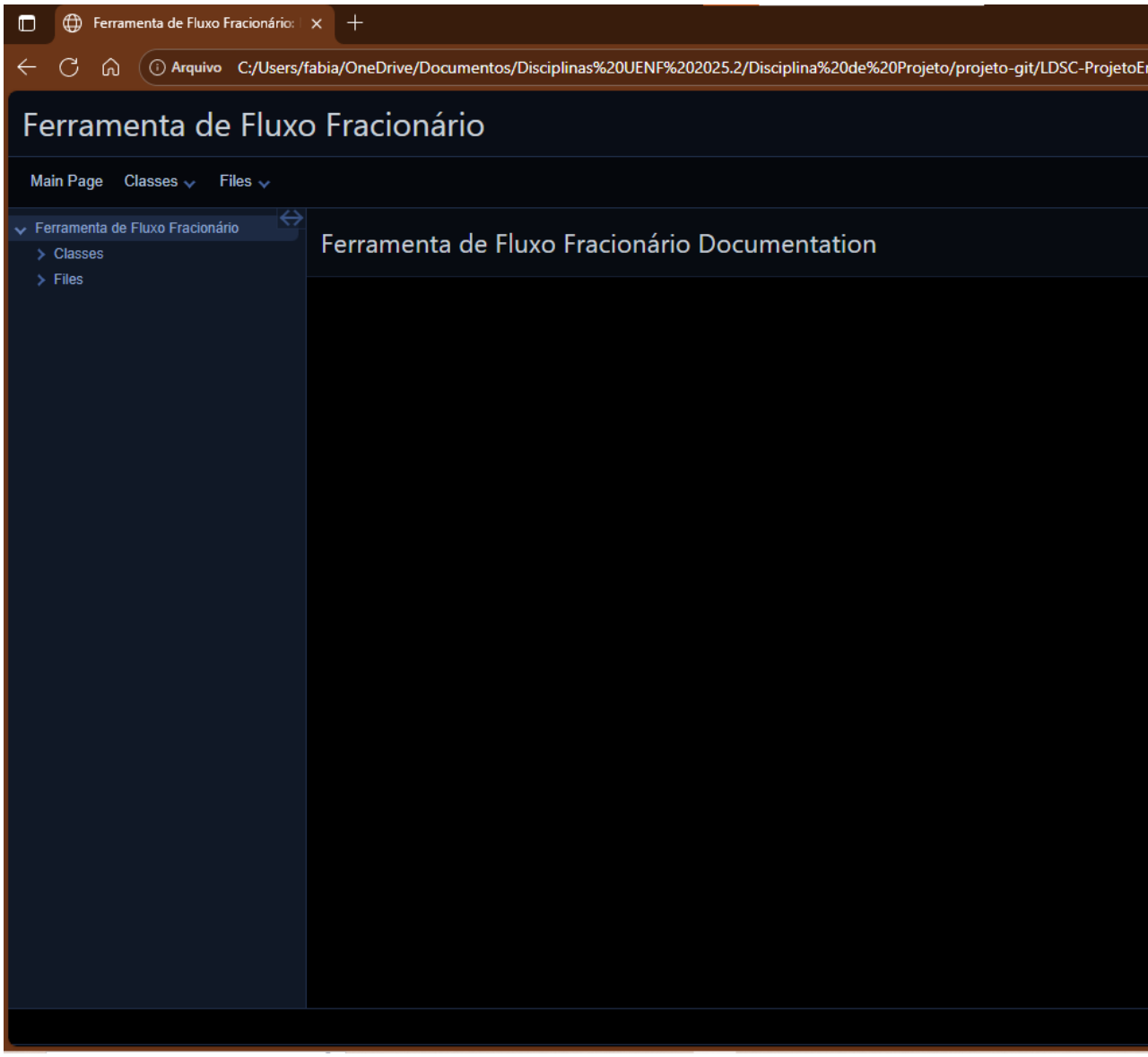

Figura 8.1: Página inicial.

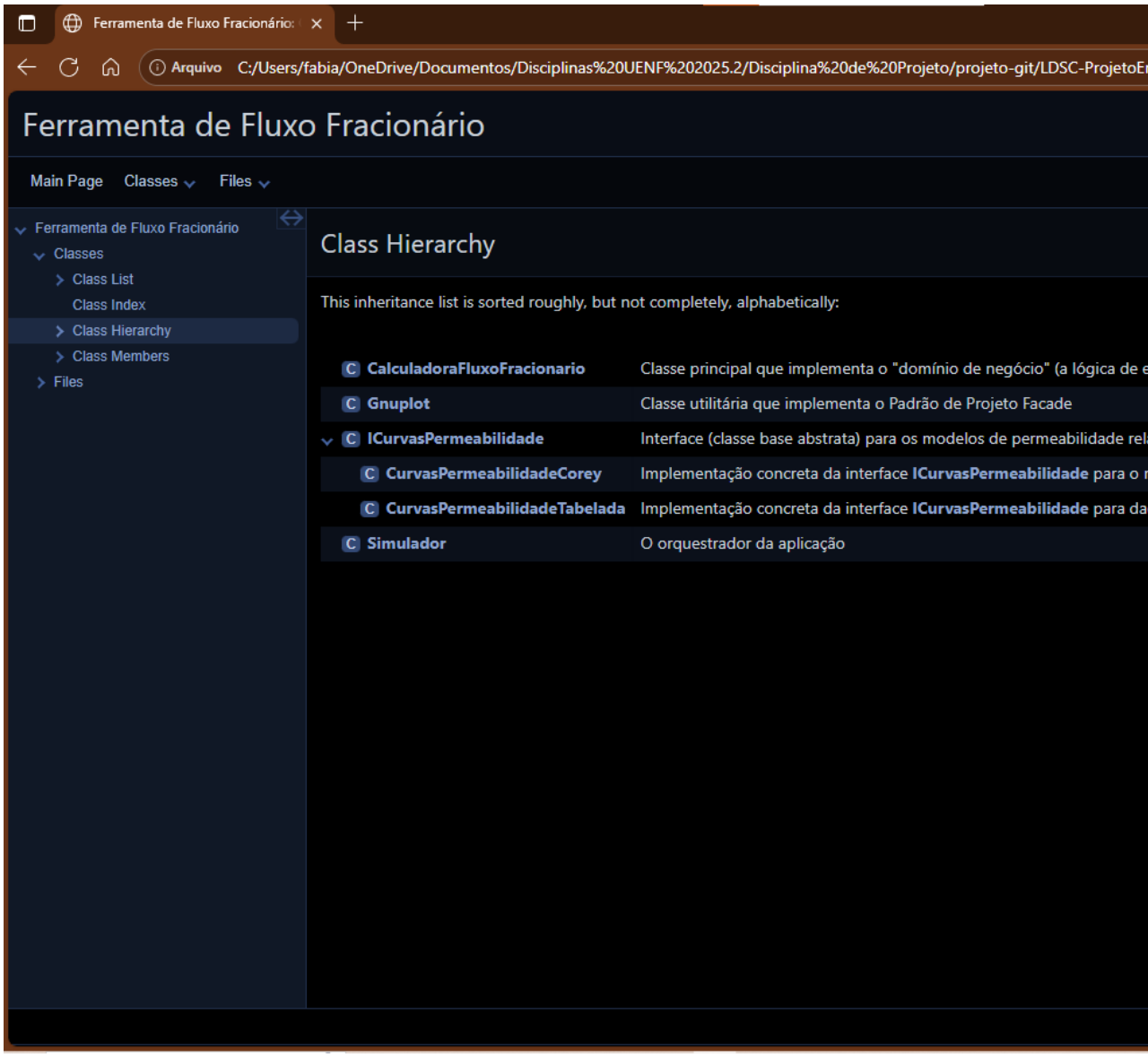

Figura 8.2: Hierarquia de classes.

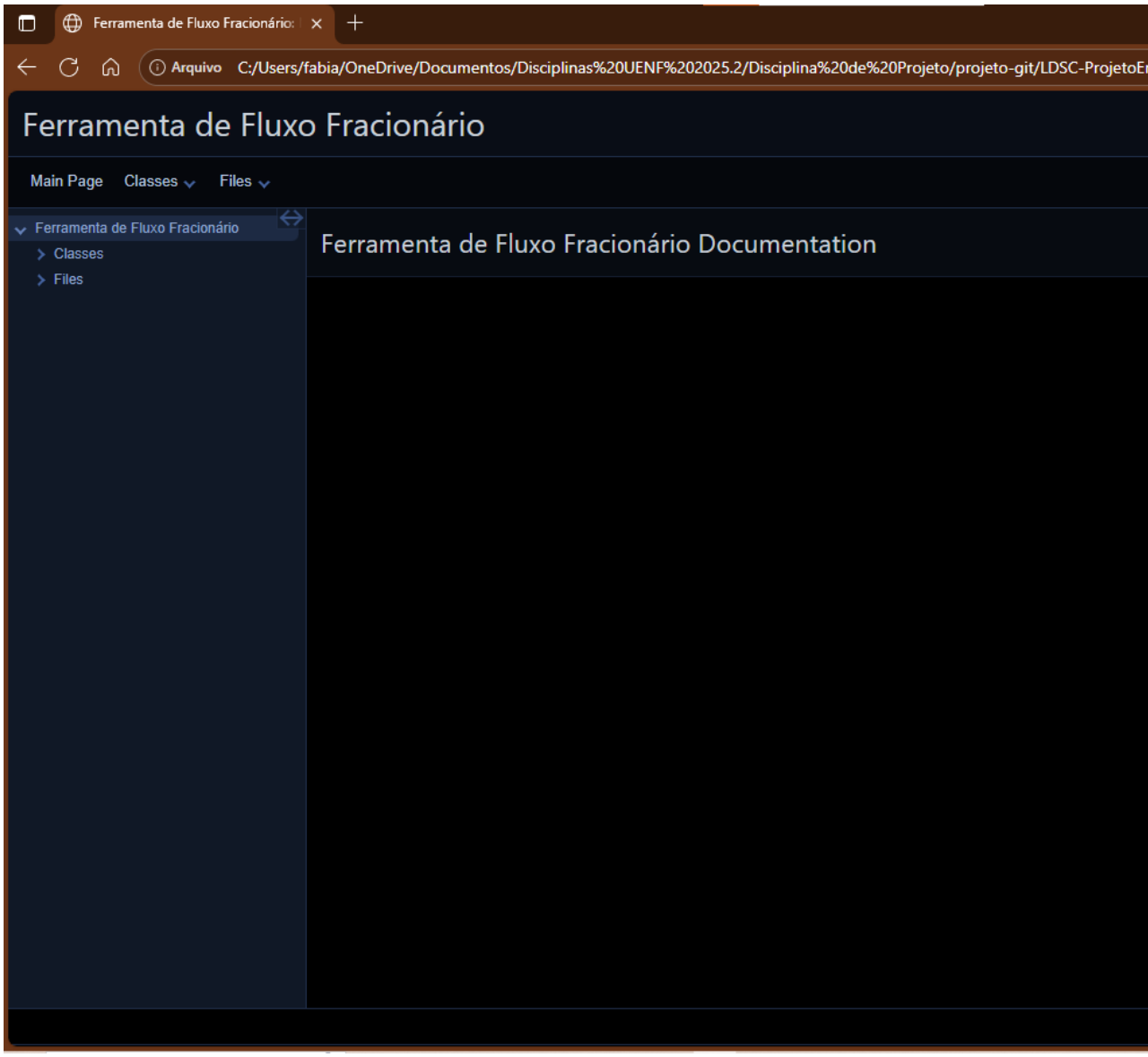

Figura 8.3: Página para `CalculadoraFluxoFracionario` *Class Reference*.

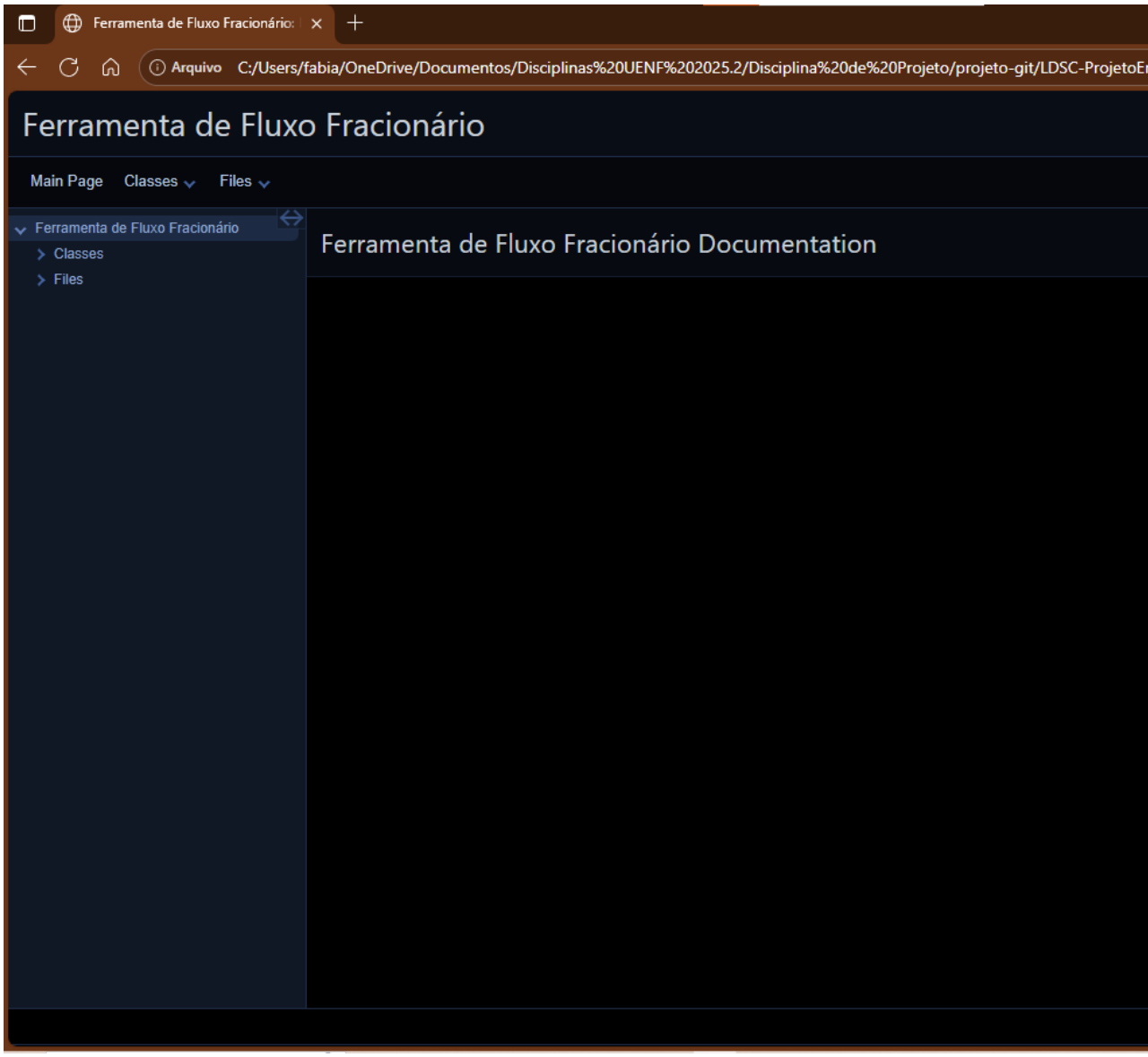

Figura 8.4: Arquivo `CalculadoraFluxoFracionario.h`.

Referências Bibliográficas

[Ahmed, 2010] Ahmed, T. (2010). *Reservoir Engineering Handbook*. Gulf Publishing Company, 4th edition.

[Buckley and Leverett, 1942] Buckley, S. E. and Leverett, M. C. (1942). Mechanism of Fluid Displacement in Sands. *Transactions of the AIME*, 146:107–116.

[Corey, 1954] Corey, A. T. (1954). The Interrelation Between Gas and Oil Relative Permeabilities. *Producers Monthly*, 19(1):38–41.

[Lake et al., 2010] Lake, L. W. et al. (2010). *Fundamentals of Enhanced Oil Recovery*. Society of Petroleum Engineers.

Apêndice A

Formato do Arquivo de Entrada

O software utiliza um arquivo de entrada de texto `.txt` baseado em palavras--chave. O parser identifica a palavra-chave (ex: `VISC_OLEO`) e lê o valor na mesma linha.

A palavra-chave `MODELO_KR` define qual modelo de permeabilidade será usado: `TABELADO` ou `COREY` .

A.1 Exemplo de Arquivo de Entrada (Modelo Tabelado)

A seguir a Listagem A.1, com um exemplo de arquivo de entrada para modelo tabelado.

A.2 Exemplo de Arquivo de Entrada (Modelo Corey)

A seguir a Listagem A.2, com um exemplo de arquivo de entrada para modelo Corey.

Listagem A.1: Exemplo de arquivo de entrada para modelo tabelado.

```
# Exemplo de arquivo de entrada para modelo tabelado
# Comentários são ignorados

VISC_OLEO 1.5
VISC_AGUA 0.8

MODELO_KR TABELADO

# Definição da tabela Sw, Krw, Kro
# Os dados são lidos até a palavra-chave FIM_DADOS
DADOS_KR_INICIO
0.20 0.00 0.90
0.30 0.05 0.75
0.40 0.12 0.50
0.50 0.20 0.30
0.60 0.30 0.15
0.70 0.40 0.05
0.80 0.50 0.00
FIM_DADOS
```

Listagem A.2: Exemplo de arquivo de entrada para modelo Corey.

```
# Exemplo de arquivo de entrada para modelo Corey
# Comentários são ignorados

VISC_OLEO 2.0
VISC_AGUA 1.0

MODELO_KR COREY

# Parâmetros da correlação de Corey
COREY_SWIR      0.15
COREY_SORW      0.20
COREY_KRW_MAX   0.5
COREY_KRO_MAX   0.9
COREY_NW        2.0
COREY_NO        2.5
```

Apêndice B

Guia de Compilação e Execução

Este apêndice descreve os passos exatos para compilar e executar o software "Ferramenta de Fluxo Fracionário" em um ambiente *Windows*, utilizando o terminal MSYS2 MINGW64.

B.1 Pré-requisitos

Certifique-se de que o ambiente MSYS2 está instalado com os pacotes `mingw-w64-x86_64-toolchain` (que inclui g++ e *make*) e `mingw-w64-x86_64-gnuplot`.

B.2 Compilação do Projeto

1. Abra o terminal MSYS2 MINGW64.
2. Navegue até a pasta raiz do projeto (onde está o arquivo `Makefile`). Exemplo, Listagem B.1:

   Listagem B.1: Comando para navegar até a pasta do projeto.

   ```
   cd "/c/Users/SeuUsuario/Documentos/ProjetoFluxoFracionario/"
   ```

3. Execute o comando de limpeza para garantir uma compilação limpa, Listagem B.2:

Listagem B.2: Limpando arquivos antigos.

```
make clean
```

Listagem B.3: Comando de compilação.

```
make
```

Listagem B.4: Executando o Teste 1.

```
./bin/fw_calc test/Teste-01.in
```

4. Compile o projeto usando o comando `make`. O `Makefile` irá gerenciar a compilação de todos os arquivos `.cpp` e gerar o executável na pasta `bin/`, Listagem B.3.

B.3 Execução e Teste

Após a compilação bem-sucedida (mensagem "Compilação concluída!"), o programa pode ser executado.

1. Para rodar o Teste 1 (Modelo Tabelado), execute, Listagem B.4:
2. Para rodar o Teste de Validação (Dados de Tarek Ahmed,[Ahmed, 2010]), execute, Listagem B.5:

Se o ambiente estiver configurado corretamente, uma janela do Gnuplot abrirá automaticamente exibindo o gráfico da curva de fluxo fracionário.

Listagem B.5: Executando a Validação com Literatura.

```
./bin/fw_calc test/Teste-Tarek-1cp.in
```

Índice Remissivo