

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

PROJETO ENGENHARIA
SIMULAÇÃO IMPES DE PROCESSOS DE DRENAGEM E
EMBEBIÇÃO DE TESTEMUNHOS

THIAGO COUTO DE ALMEIDA CHAVES

MACAÉ - RJ
DEZEMBRO - 2019

Capítulo 1

Especificação

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

1.1 Especificação do software - descrição dos requisitos

Será desenvolvido um software em modo terminal para a simulação de processos de embebição e drenagem em um testemunho curto de rocha, onde a partir de dados geométricos como diâmetro e comprimento do testemunho; de dados petrofísicos como porosidade, permeabilidade; de dados dos fluidos utilizados no experimento como a viscosidade do óleo e da água; e de dados do experimento em si como a pressão atmosférica, a vazão de injeção e o tempo de injeção; pode-se obter os campos de saturação e de pressão no testemunho durante e ao fim do experimento.

Para a solução dos campos de saturação e de pressão, será utilizado o método numérico IMPES que é adequado para a simulação de fluxo bifásico e incompressível no meio poroso - que é o tipo de fluxo em questão. Este software realizará a simulação utilizando diferentes formulações das equações de fluxo além de dar a opção de realizar a simulação utilizando somente um núcleo de processamento - simulação serial - ou vários núcleos - simulação paralela. Ao final da simulação, o software gerará quatro saídas gráficas que correspondem aos campos de saturação e pressão dos experimentos de embebição e drenagem.

1.2 Especificação do software - requisitos

1.2.1 Nome do sistema/produto

Nome	Petro-Explorer
Componentes principais	Simulador dos ensaios de drenagem e embebição
Missão	Resolver os campos de saturação e pressão utilizando o método IMPES

1.2.2 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

RF-01	O usuário deverá ter liberdade em escolher quais as maneiras em que dar-se-á a entrada de dados.
RF-02	O usuário deverá ter liberdade em escolher quais as maneiras em que dar-se-á a entrada de dados.
RF-03	Deve permitir a geração de gráficos com o resultado da simulação.
RF-04	Deve mostrar os resultados na tela.
RF-05	Deve salvar os resultados em um arquivo de disco em um diretório de escolha do usuário.

1.2.3 Requisitos não funcionais

RNF-01	O software é livre;
RNF-02	O software é escrito em C++ para a plataforma Windows;
RNF-03	Utiliza processamento paralelo para maior rapidez na simulação.

1.3 Casos de uso do software

Esta seção contém uma tabela 1.1 que descreve um caso de uso do sistema, assim como os diagramas de caso de uso.

Tabela 1.1: Exemplo de caso de uso.

Nome do caso de uso:	Simulação do processo de drenagem
Resumo/descrição:	Solução dos campos de pressão e saturação do testemunho através do método IMPES em um ensaio de drenagem.
Etapas:	<ol style="list-style-type: none"> 1. Entrada dos dados geométricos, petrofísicos e experimentais; 2. Seleção do padrão de unidades utilizado; 3. Seleção do tipo de simulação a ser utilizada; 4. Aplicação do método numérico IMPES. 5. Analisar/validar resultados; 6. Exportar/salvar resultados.
Cenários alternativos:	Valores de porosidade menores que 0 e maiores 1 serão rejeitados pelo software que em seguida, dará ao usuário controle para alterar a entrada de dados fornecida.

1.3.1 Diagrama de caso de uso geral

O diagrama de caso de uso geral da Figura 1.1 mostra as diferentes operações que o usuário pode realizar com o Petro-Explorer.

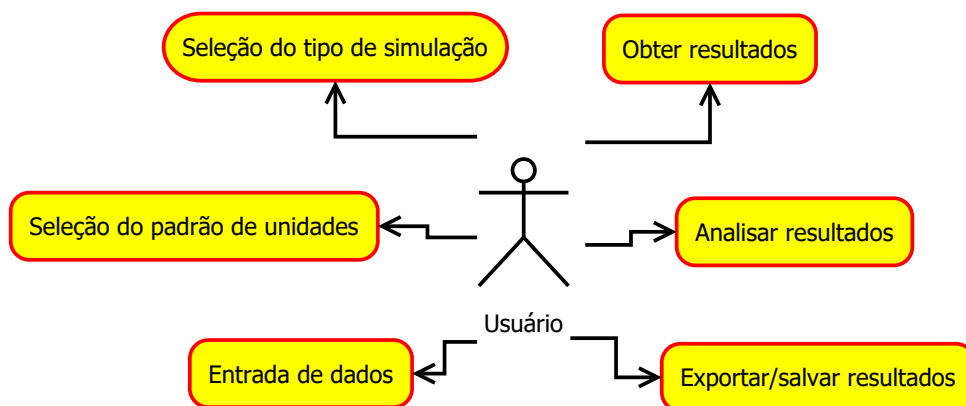


Figura 1.1: Diagrama de caso de uso – Caso de uso geral

Capítulo 2

Elaboração

Esta etapa envolve o estudo dos conceitos relacionados ao simulador deste projeto, análise de domínio e a identificação dos pacotes para a análise dos requisitos, ajustando os requisitos iniciais de forma a desenvolver um sistema útil, que atenda às necessidades do usuário e, na medida do possível, permita seu reuso e futura extensão.

2.1 Análise de domínio

Tendo como base a experiência de projetos realizados no passado, do estudo da formulação teórica, da avaliação do escopo e da especificação do projeto, resume-se a seguir, os domínios abordados pelo programa:

- Engenharia de Reservatórios: domínio principal deste software. O software utiliza conceitos como pressão capilar, permeabilidade relativa, embebição e drenagem. Além de formulações matemáticas das equações dos campos de pressão e saturação discretizadas conforme o método IMPES.
- Engenharia de Software: responsável por prover o ambiente computacional pelo qual as análises serão realizadas. Por meio do uso da linguagem *C++* e seus recursos como estruturas de controle, estruturas condicionais, contêineres de dados, classes, etc.; bibliotecas auxiliares como a *armadillo*; e programação paralela utilizando múltiplas *threads* e processos; o usuário tem a sua disposição um software fluido e competente que dará suporte à realização de suas simulações.

2.2 Formulação teórica

2.2.1 Modelos de permeabilidade relativa e de pressão capilar

Para as curvas de pressão capilar e de permeabilidade relativa, tomou-se por base os modelos empíricos do livro Engenharia de Reservatórios de Petróleo (Rosa et al., 2011), de forma que:

A expressão da permeabilidade relativa à água (Eq. 2.183 - Rosa et al., 2011):

$$K_{rw}(S_w) = \frac{Krw_{s=1-Sor}(S_w - S_{wi})^A}{(1 - Sor - S_{wi})^A}$$

A expressão da permeabilidade relativa ao óleo (Eq. 2.184 - Rosa et al., 2011):

$$K_{ro}(S_w) = \frac{Kro_{s=S_{wi}}(1 - S_w - Sor)^B}{(1 - Sor - S_{wi})^B}$$

A expressão da pressão capilar (Eq. 2.185 - Rosa et al., 2011):

$$P_c(S_w) = \frac{Pc_{S_w=S_{wi}}(1 - S_w - Sor)^C}{(1 - Sor - S_{wi})^C}$$

2.2.2 Dedução da Equação da Pressão

Premissas:

- Fluxo incompressível, horizontal e bifásico
- Permeabilidade absoluta constante
- Viscosidade dos fluidos constante
- Ausência de termos fonte
- Fluxo 1D – direção X

Dedução:

$$\nabla \cdot \vec{u} = q$$

- Fluxo 1D e sem termos fonte:

$$\frac{\partial u_x}{\partial x} = 0$$

- Decompondo a velocidade em termos das fases água e óleo:

$$u_x = u_{wx} + u_{ox}$$

$$u_x = \frac{-KK_{rw}}{\mu_w} \frac{\partial p_w}{\partial x} - \frac{KK_{ro}}{\mu_o} \frac{\partial p_o}{\partial x}$$

- Aplicando a definição de pressão capilar para retirar a dependência com a pressão da fase água:

$$\begin{aligned}
u_x &= \frac{-K K_{rw}}{\mu_w} \frac{\partial p_o}{\partial x} + \frac{K K_{rw}}{\mu_w} \frac{\partial p_c}{\partial x} - \frac{K K_{ro}}{\mu_o} \frac{\partial p_o}{\partial x} \\
u_x &= -K \left(\frac{K_{rw}}{\mu_w} + \frac{K_{ro}}{\mu_o} \right) \frac{\partial p_o}{\partial x} + \frac{K K_{rw}}{\mu_w} \frac{\partial p_c}{\partial x} \\
u_x &= -K \lambda_T \frac{\partial p}{\partial x} + K \lambda_w \frac{\partial p_c}{\partial x} \\
\frac{\partial}{\partial x} \left[-K \lambda_T \frac{\partial p}{\partial x} + K \lambda_w \frac{\partial p_c}{\partial x} \right] &= 0 \\
\frac{\partial}{\partial x} \left[K \lambda_w \frac{\partial p_c}{\partial x} \right] &= \frac{\partial}{\partial x} \left[K \lambda_T \frac{\partial p}{\partial x} \right]
\end{aligned}$$

- Tem-se finalmente:

$$\frac{\partial}{\partial x} \left[\lambda_T \frac{\partial p}{\partial x} \right] = \frac{\partial}{\partial x} \left[\lambda_w \frac{\partial p_c}{\partial x} \right] \quad (\text{Equação da Pressão})$$

2.2.3 Dedução da Equação da Saturação

Premissas:

- Fluxo incompressível, horizontal e bifásico
- Permeabilidade absoluta constante
- Viscosidade dos fluidos constante
- Ausência de termos fonte
- Fluxo 1D – direção X

$$\phi \frac{\partial S}{\partial t} + \nabla \cdot \left\{ \vec{K} f_{w(S_w)} \lambda_{o(S_w)} \left[\frac{\partial p_c}{\partial S_w} \nabla S - (\rho_o - \rho_w) g \nabla z \right] + f_{w(S_w)} \vec{U} \right\} = \tilde{q}_{w(p,s)}$$

Abordagens:

1. $U_x = K \lambda_T \frac{\partial p}{\partial x} + K \lambda_w \frac{\partial p_c}{\partial x}$
2. $U_x = \frac{q_o(t) + q_w(t)}{A}$

Abordagem 1

$$\begin{aligned}
\phi \frac{\partial S}{\partial t} + \frac{\partial}{\partial x} \left[K f_{w(S_w)} \lambda_o \frac{\partial p_c}{\partial x} + K f_{w(S_w)} \lambda_w \frac{\partial p_c}{\partial x} - K f_w \lambda_T \frac{\partial p_c}{\partial x} \right] &= 0 \\
\frac{\phi}{K} \frac{\partial S}{\partial t} + \frac{\partial}{\partial x} \left[f_{w(S_w)} \lambda_T \frac{\partial p_c}{\partial x} - f_{w(S_w)} \lambda_T \frac{\partial p}{\partial x} \right] &= 0
\end{aligned}$$

- $f_w = \frac{\lambda_w}{\lambda_T}$

$$\frac{\phi}{K} \frac{\partial S}{\partial t} + \frac{\partial}{\partial x} \left[\lambda_w \frac{\partial p_c}{\partial x} - \lambda_w \frac{\partial p}{\partial x} \right] = 0$$

- Utilizando a Equação da Pressão, obtemos a seguinte equação:

$$\frac{\phi}{K} \frac{\partial S}{\partial t} + \frac{\partial}{\partial x} \left[\lambda_T \frac{\partial p}{\partial x} - \lambda_w \frac{\partial p}{\partial x} \right] = 0$$

$$\frac{\phi}{K} \frac{\partial S}{\partial t} + \frac{\partial}{\partial x} \left[\lambda_o \frac{\partial p}{\partial x} + \lambda_w \frac{\partial p}{\partial x} - \lambda_w \frac{\partial p}{\partial x} \right] = 0$$

$$\frac{\phi}{K} \frac{\partial S}{\partial t} + \frac{\partial}{\partial x} \left[\lambda_o \frac{\partial p}{\partial x} \right] = 0 \quad (Eq. da Saturação A)$$

Abordagem 2

$$\phi \frac{\partial S}{\partial t} + \frac{\partial}{\partial x} \left[K f_w \lambda_o \frac{\partial p_c}{\partial x} + f_w U_x \right] = 0$$

$$\phi \frac{\partial S}{\partial t} + \frac{\partial (f_w U_x)}{\partial x} = - \frac{\partial}{\partial x} \left[K f_w \lambda_o \frac{\partial p_c}{\partial x} \right]$$

$$\phi \frac{\partial S}{\partial t} + \frac{U_x \partial (f_w)}{\partial x} = - \frac{\partial}{\partial x} \left[K f_w \lambda_o \frac{\partial p_c}{\partial x} \right]$$

$$\phi \frac{\partial S}{\partial t} + \left(\frac{q_o(t) + q_w(t)}{A} \right) \frac{\partial (f_w)}{\partial x} = - \frac{\partial}{\partial x} \left[K f_w \lambda_o \frac{\partial p_c}{\partial x} \right] \quad (Eq. da Saturação B)$$

2.2.4 Discretização da Equação da Pressão

- Para $i \neq 0$ e $i \neq I$:

$$\lambda_{T_{i+1/2}} p_{i+1} - \left(\lambda_{T_{i+1/2}} + \lambda_{T_{i-1/2}} \right) p_i + \lambda_{T_{i-1/2}} p_{i-1} = \lambda_{W_{i+1/2}} p_{c_{i+1}} - \left(\lambda_{W_{i+1/2}} + \lambda_{W_{i-1/2}} \right) p_{c_i} + \lambda_{W_{i-1/2}} p_{c_{i-1}}$$

- $i = 0$:

$$\lambda_{T_{1/2}} p_1 - \left(\lambda_{T_{1/2}} + \lambda_{T_{-1/2}} \right) p_0 + \lambda_{T_{-1/2}} p_{-1} = \lambda_{W_{1/2}} p_{c1} - \left(\lambda_{W_{1/2}} + \lambda_{W_{-1/2}} \right) p_{c0} + \lambda_{W_{-1/2}} p_{c-1}$$

$$\frac{q_o + q_w}{A} = -K \lambda_T \frac{\partial p}{\partial x} + K \lambda_w \frac{\partial p_c}{\partial x}$$

$$\frac{(q_o + q_w)h}{AK} = -\lambda_{T_{-\frac{1}{2}}}(p_0 - p_{-1}) + \lambda_{W_{-\frac{1}{2}}}(p_{c_0} - p_{c_{-1}})$$

$$\frac{(q_o + q_w)h}{AK} = -\lambda_{T_{-\frac{1}{2}}}(p_0 - p_{-1}) + \lambda_{W_{-\frac{1}{2}}}(p_{c_0} - p_{c_{-1}})$$

$$\lambda_{T_{\frac{1}{2}}}(p_1 - p_0) = \lambda_{W_{\frac{1}{2}}}(p_{c_1} - p_{c_0}) - \frac{(q_o + q_w)h}{AK}$$

- Para $i = I$, tem-se duas abordagens:

Abordagem A: Realizar o mesmo raciocínio de $i=0$

Pode-se demonstrar que:

$$\frac{U_x h}{x}|_{x=L} = -\lambda_{T_L}(p_{I+1} - p_I) + \lambda_{W_L}(p_{c_{I+1}} - p_{c_I})$$

E finalmente:

$$\lambda_{T_{I-\frac{1}{2}}}(p_I - p_{I-1}) = \lambda_{W_{I-1/2}}(p_{c_I} - p_{c_{I-1}}) - \frac{U_x \cdot h}{x}$$

Abordagem B: Uso de meia discretização

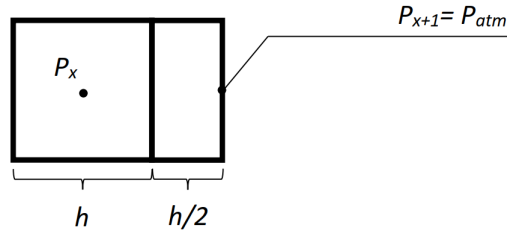


Figura 2.1: Meia-discretização

$$\frac{\partial}{\partial x} \left[\lambda_T \frac{\partial p}{\partial x} \right] = \frac{\partial}{\partial x} \left[\lambda_w \frac{\partial p_c}{\partial x} \right]$$

Discretizando a equação acima em diferenças finitas:

$$\frac{\lambda_{T_{I+\frac{1}{2}}} \frac{(p_{I+1} - p_I)}{h/2} - \lambda_{T_{I-\frac{1}{2}}} \frac{(p_I - p_{I-1})}{h}}{\frac{h/2 + h}{2}} - \frac{\lambda_{W_{I+\frac{1}{2}}} \frac{(p_{c_{I+1}} - p_{c_I})}{h/2} - \lambda_{W_{I-\frac{1}{2}}} \frac{(p_{c_I} - p_{c_{I-1}})}{h}}{\frac{h/2 + h}{2}} = 0$$

$$2\lambda_{T_{I+\frac{1}{2}}}(p_{atm} - p_I) - \lambda_{T_{I-\frac{1}{2}}}(p_I - p_{I-1}) =$$

$$\begin{aligned}
& 2 \lambda_{W_{I+\frac{1}{2}}} (p_{cI+1} - p_{cI}) - \lambda_{W_{I-\frac{1}{2}}} (p_{cI} - p_{cI-1}) \\
& 2 \lambda_{T_{I+\frac{1}{2}}} p_{atm} - 2 \lambda_{T_{I+\frac{1}{2}}} p_I - \lambda_{T_{I-\frac{1}{2}}} p_I + \lambda_{T_{I-\frac{1}{2}}} p_{I-1} = \\
& 2 \lambda_{W_{I+\frac{1}{2}}} (p_{cI+1} - p_{cI}) - \lambda_{W_{I-\frac{1}{2}}} (p_{cI} - p_{cI-1}) \\
& - \left(2 \lambda_{T_{I+\frac{1}{2}}} + \lambda_{T_{I-\frac{1}{2}}} p_I + \lambda_{T_{I-\frac{1}{2}}} \right) p_I + \lambda_{T_{I-\frac{1}{2}}} p_{I-1} = \\
& 2 \lambda_{W_{I+\frac{1}{2}}} (p_{cI+1} - p_{cI}) - \lambda_{W_{I-\frac{1}{2}}} (p_{cI} - p_{cI-1}) - 2 \lambda_{T_{I+\frac{1}{2}}} p_{atm}
\end{aligned}$$

2.2.5 Discretização da Equação da Saturação

Equação da saturação A

$$\begin{aligned}
& \frac{\phi}{K} \frac{\partial S}{\partial t} = - \frac{\partial}{\partial x} \left[\lambda_o \frac{\partial p}{\partial x} \right] \\
& - \frac{\phi}{K} \frac{\partial S}{\partial t} = \frac{\lambda_{0_{i+\frac{1}{2}}} \frac{(p_{i+1} - p_i)}{h} - \lambda_{0_{i-\frac{1}{2}}} \frac{(p_i - p_{i-1})}{h}}{h} \\
& - \frac{\phi}{K} \frac{\partial S}{\partial t} = \frac{1}{h^2} \left[\lambda_{0_{i+\frac{1}{2}}} (p_{i+1} - p_i) - \lambda_{0_{i-\frac{1}{2}}} (p_i - p_{i-1}) \right]
\end{aligned}$$

$$\bullet \text{ i} = 0 \text{ (Drenagem)} \iff \lambda_{W_{-\frac{1}{2}}} = 0$$

$$\frac{\phi}{K} \frac{\partial S}{\partial t} = - \frac{1}{h^2} \left[\lambda_{W_{\frac{1}{2}}} (p_1 - p_0) + \frac{U_x \cdot h}{k} + \lambda_{W_{-\frac{1}{2}}} \frac{(p_0 - p_{-1})}{h} - \lambda_{W_{-\frac{1}{2}}} \frac{(p_{c0} - p_{c-1})}{h} \right]$$

$$\frac{\partial S}{\partial t} = - \frac{k}{\phi h^2} \left[\lambda_{W_{\frac{1}{2}}} (p_1 - p_0) + \frac{U_x \cdot h}{k} \right]$$

$$\bullet \text{ i} = 0 \text{ (Embebição)} \iff \lambda_{0_{-\frac{1}{2}}} = 0$$

$$\frac{\partial S}{\partial t} = - \frac{k}{\phi h^2} \left[\lambda_{0_{i+\frac{1}{2}}} (p_{i+1} - p_i) \right]$$

$$\bullet \text{ i} = I \text{ (Uso de meia-discretização)}$$

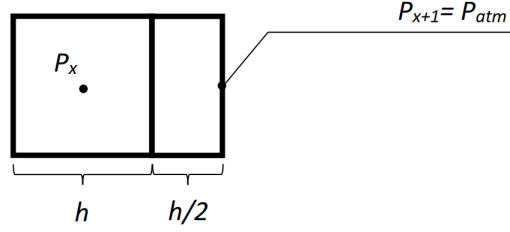


Figura 2.2: Meia-discretização

$$-\frac{\phi}{K} \frac{\partial S}{\partial t} = \frac{\lambda_{0_{I+\frac{1}{2}}} \frac{(p_{I+1} - p_I)}{h/2} - \lambda_{0_{I-\frac{1}{2}}} \frac{(p_I - p_{I-1})}{h}}{\frac{3h}{4}}$$

$$\frac{\phi}{K} \frac{\partial S}{\partial t} = -\frac{4}{3h} \left[2 \lambda_{0_{I+\frac{1}{2}}} \frac{(p_{atm} - p_I)}{h} - \lambda_{0_{I-\frac{1}{2}}} \frac{(p_I - p_{I-1})}{h} \right]$$

$$\frac{\partial S}{\partial t} = -\frac{4k}{3\phi h^2} \left[2 \lambda_{0_{I+\frac{1}{2}}} (p_{atm} - p_I) - \lambda_{0_{I-\frac{1}{2}}} (p_I - p_{I-1}) \right]$$

Equação da saturação B

$$\phi \frac{\partial S}{\partial t} + \left(\frac{q_o(t) + q_w(t)}{A} \right) \frac{\partial (f_w)}{\partial x} = -\frac{\partial}{\partial x} \left[K f_w \lambda_o \frac{\partial p_c}{\partial x} \right]$$

$$\phi \frac{\partial S}{\partial t} + u \frac{\partial (f_w)}{\partial x} = -\frac{\partial}{\partial x} \left[K f_w \lambda_o \frac{\partial p_c}{\partial x} \right]$$

$$\phi \frac{\partial S}{\partial t} = -\frac{\partial}{\partial x} \left[K f_w \lambda_o \frac{\partial p_c}{\partial x} \right] - u \frac{\partial (f_w)}{\partial x}$$

$$\phi \frac{\partial S}{\partial t} = -\frac{K}{h} \left[(f_w \lambda_o)|_{i+1/2} \frac{(p_{c_{i+1}} - p_{c_i})}{h} - (f_w \lambda_o)|_{i-1/2} \frac{(p_{c_i} - p_{c_{i-1}})}{h} \right] - u \frac{(f_{w_{i+1/2}} - f_{w_{i-1/2}})}{h}$$

$$\frac{\partial S}{\partial t} = -\frac{K}{\phi h^2} \left[(f_w \lambda_o)|_{i+1/2} (p_{c_{i+1}} - p_{c_i}) - (f_w \lambda_o)|_{i-1/2} (p_{c_i} - p_{c_{i-1}}) \right] - u \frac{(f_{w_{i+1/2}} - f_{w_{i-1/2}})}{\phi h}$$

• **i = 0 (Drenagem)** $\iff \lambda_{W_{-\frac{1}{2}}} = 0$

$$\frac{\partial S}{\partial t} = -\frac{K}{\phi h^2} \left[(f_w \lambda_o)|_{1/2} (p_{c_1} - p_{c_0}) - (f_w \lambda_o)|_{-1/2} (p_{c_i} - p_{c_{i-1}}) \right] - u \frac{(f_{w_{1/2}} - f_{w_{-1/2}})}{\phi h}$$

$$\frac{\partial S}{\partial t} = -\frac{K}{\phi h^2} \left[(f_w \lambda_o)|_{1/2} (p_{c_1} - p_{c_0}) \right] - u \frac{(f_{w_{1/2}})}{\phi h}$$

• **i = 0 (Embebição)** $\iff \lambda_{0_{-\frac{1}{2}}} = 0$

$$\frac{\partial S}{\partial t} = -\frac{K}{\phi h^2} [(f_w \lambda_o)|_{1/2}(p_{c1} - p_{c0}) - (f_w \lambda_o)|_{-1/2}(p_{ci} - p_{ci-1})] - u \frac{(f_{w1/2} - f_{w-1/2})}{\phi h}$$

$$\frac{\partial S}{\partial t} = -\frac{K}{\phi h^2} [(f_w \lambda_o)|_{1/2}(p_{c1} - p_{c0})] - u \frac{(f_{w1/2} - 1)}{\phi h}$$

- **i = I (Uso de meia-discretização)**

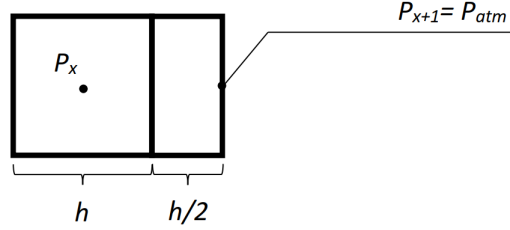


Figura 2.3: Meia-discretização

$$\frac{\partial S}{\partial t} = -\frac{4K}{\phi 3h^2} [2(f_w \lambda_o)|_{I+1/2}(p_{cI+1} - p_{cI}) - (f_w \lambda_o)|_{I-1/2}(p_{cI} - p_{cI-1})] - 2u \frac{(f_{wI+1/2} - f_{wI-1/2})}{\phi h}$$

2.2.6 Outros parâmetros de entrada e curvas de pressão capilar e permeabilidade relativa

Para a obtenção dos passos de tempos através da equação:

$$\Delta t = \frac{\Delta S_{max}}{|Gi^n|}$$

Foi adotado ΔS_{max} igual a 0.001.

O tratamento das transmissibilidades foi feito conforme a média upstream de um ponto (single-point upstream weighting). De tal forma que:

$$\lambda_{w_{i-1/2}} = \begin{cases} \lambda_{w_{i-1}} & \Delta \Phi < 0 \\ \lambda_{w_{i+1}} & \Delta \Phi > 0 \end{cases}$$

Porém, $\Delta \Phi < 0$ é sempre verdade para este experimento já que o fluxo é do bloco i-1 para o bloco i. Então:

$$\lambda_{w_{i-1/2}} = \lambda_{w_{i-1}}$$

$$\lambda_{w_{i+1/2}} = \lambda_{w_i}$$

2.3 Identificação de pacotes – assuntos

Por meio da análise de domínio e da formulação teórica deste projeto, temos os seguintes pacotes:

- Testemunho: Esse pacote recebe os dados do usuário e constrói uma representação do testemunho do experimento, baseado nas características geométricas e petrofísicas informadas. Além disso, esse banco de dados serve como base para os cálculos da simulação.
- Simulador: Neste pacote se encontram os algoritmos necessários para a simulação numérica-computacional dos experimentos de embebição e drenagem propostos por este software. Isto é, aqui se encontram as classes que implementam o método IMPES, tanto na forma serial, quanto na forma paralelizada.
- Gráfico: Aqui se encontra a biblioteca do *gnuplot*, necessária para a geração dos gráficos dos resultados da simulação para uma melhor análise.

2.4 Diagrama de pacotes – assuntos

O diagrama de pacotes da figura 2.4 mostra as relações existentes entre os pacotes deste software. Foram identificados, conforme seção 2.3, os seguintes pacotes: Testemunho, Simulador e Gráfico.

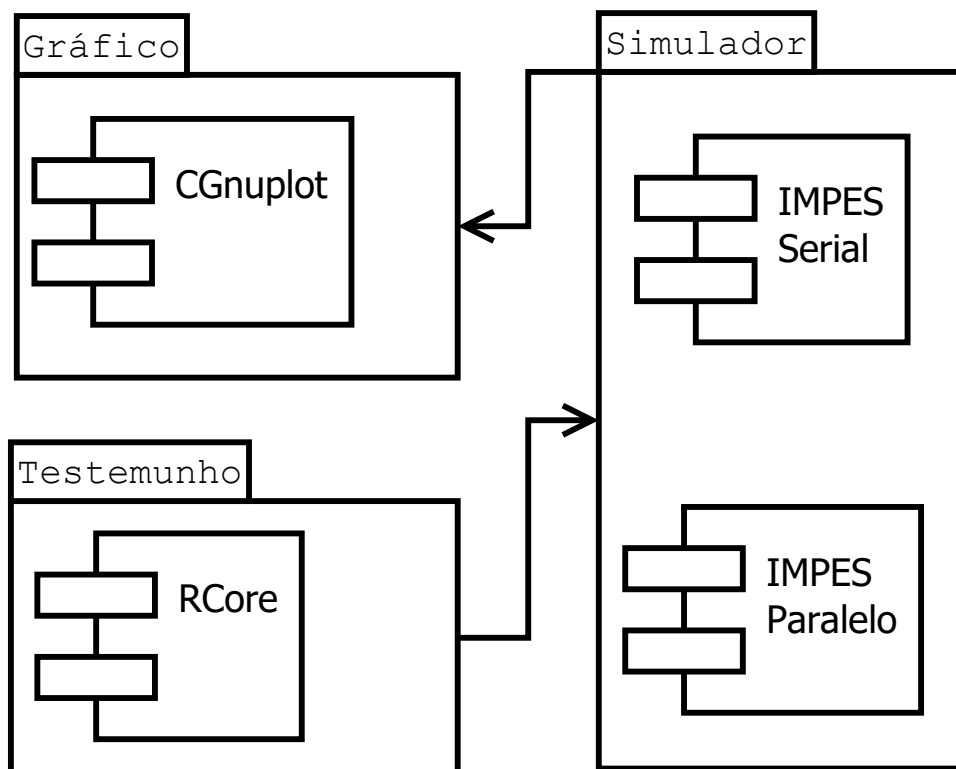


Figura 2.4: Diagrama de Pacotes

Capítulo 3

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 relações entre os pacotes, as classes, os atributos, os métodos, as heranças, as associações, as agregações, as composições e as dependências.

O modelo de análise deve ser conciso, simplificado e deve mostrar o que deve ser feito, não se preocupando como isso será realizado. Serão vistos neste capítulo, os diagramas de classe, de sequência, de máquina de estado, e de atividades; o diagrama de comunicação não foi modelado por falta de aplicação neste software em específico.

3.1 Diagramas de classes

O diagrama de classes é essencial para a montagem da versão inicial do código do software. Este diagrama é constituído pelas classes, seus métodos e atributos, além das diversas relações entre as classes. São apresentados aqui o diagrama de classes para este software na figura 3.1 onde pode-se ver as quatro principais classes deste software responsáveis por implementar o método IMPES de maneiras diferentes.



Figura 3.1: Diagrama de classes

3.1.1 Dicionário de classes

- Classe Statistics: classe responsável por realizar os seguintes cálculos de estatística básica: média, mediana, moda, desvio padrão, máximo, mínimo, amplitude, intervalo inter-quartil, percentil, e o coeficiente de Pearson.
- Classe RCore: classe responsável por modelar um plug de rocha com atributos como comprimento, diâmetro, permeabilidade, porosidade; além de aspectos dinâmicos como permeabilidade relativa e pressão capilar.
- Classe IMPES: classe responsável por implementar o método numérico IMPES de maneira serial, isto é, nesta classe não há códigos paralelizados.
- Classe IMPES_Parallel: herdeira de IMPES, implementa o método IMPES utilizando de dois tipos de paralelização, uma paralelização para o cálculo do campo de saturação e outra para o cálculo do campo de pressão.
- Struct IMPESArgs: reúne argumentos para alimentar classe IMPES.
- Struct coreArgs: reúne argumentos para alimentar classe RCore.

3.2 Diagrama de seqüência – eventos e mensagens

O diagrama de seqüências representa, no tempo, a troca de eventos e mensagens entre os objetos do sistema. Ele é parte do modelo dinâmico da análise orientada a objeto.

3.2.1 Diagrama de seqüência geral

Na figura 3.2, encontra-se o diagrama de seqüência de um caso geral de análise deste software. Foi tomado para a confecção deste diagrama a seqüência de uso que o usuário tem com o software para realizar a simulação.

Cenário: Simulação de embebição e drenagem utilizando versão paralela do método IMPES

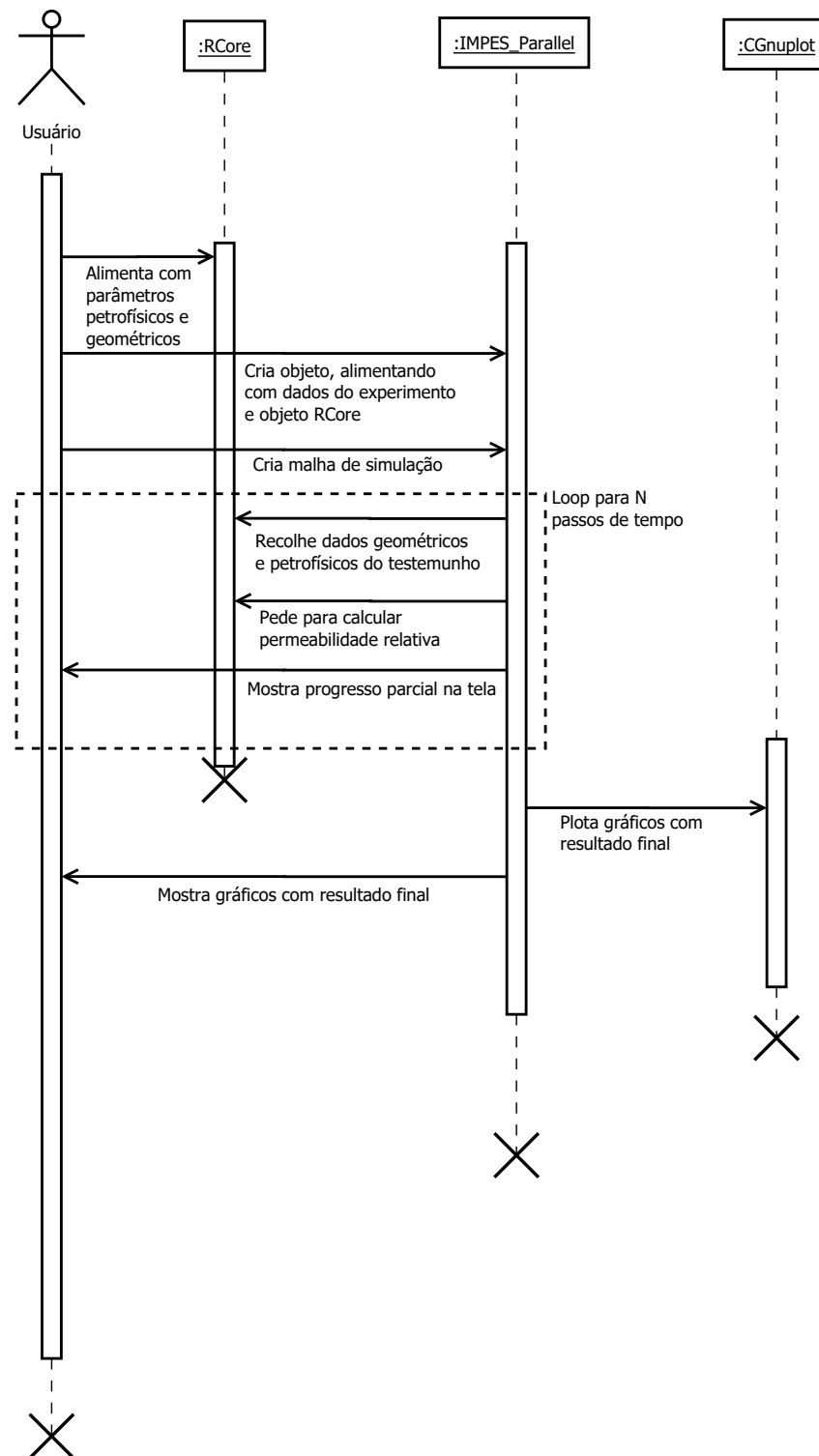


Figura 3.2: Diagrama de sequência

3.3 Diagrama de máquina de estado

Um diagrama de máquina de estado representa os diversos estados que o objeto assume e os eventos que ocorrem ao longo de sua vida ou mesmo ao longo de um processo (histórico

do objeto). É usado para modelar aspectos dinâmicos do objeto. Veja na Figura 3.3 o diagrama de máquina de estado para um objeto da classe IMPES.

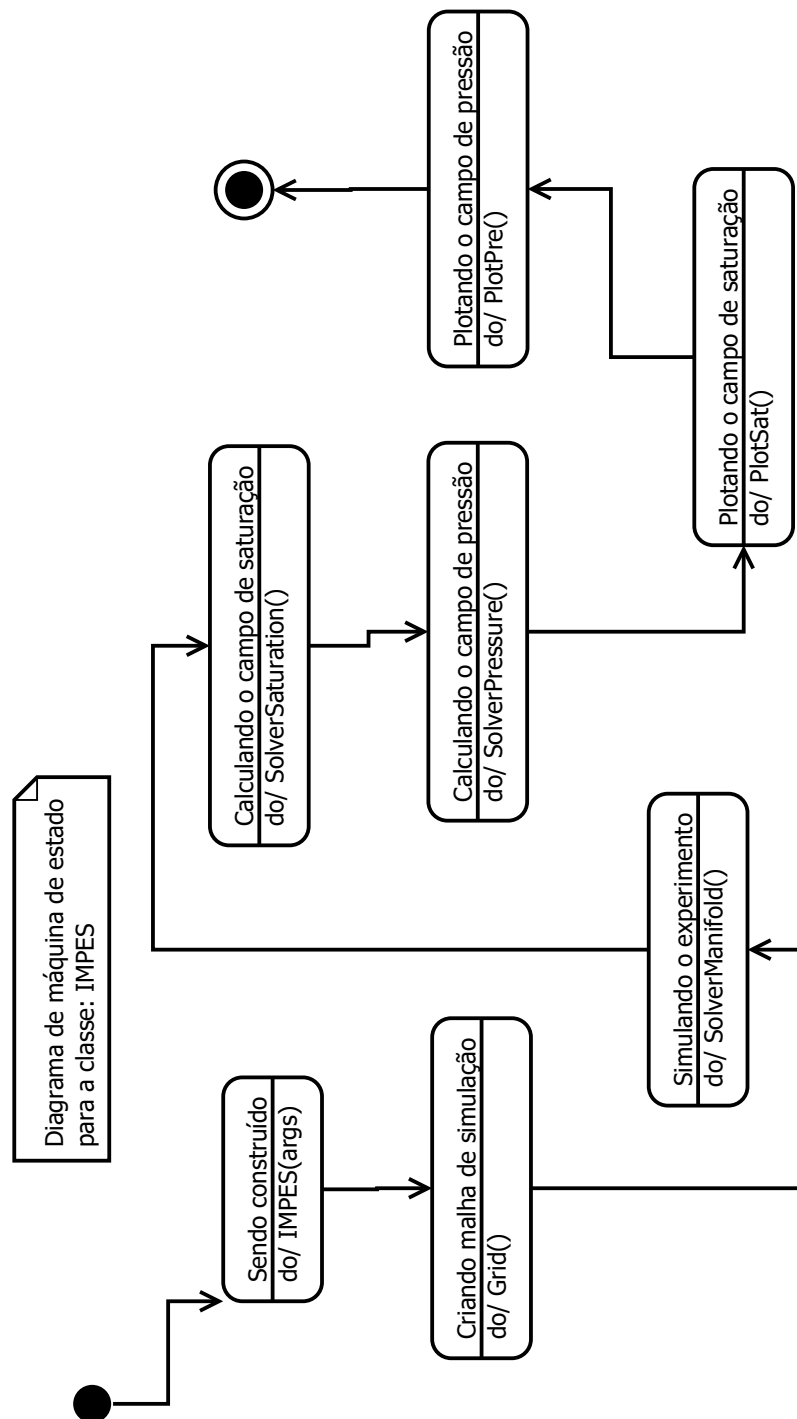


Figura 3.3: Diagrama de máquina de estado

3.4 Diagrama de atividades

O diagrama de atividades detalha as atividades que ocorrem em um estado específico do diagrama de máquina de estado. No diagrama de atividades da figura 3.4, tomou-se o

estado “Simulando o experimento” do diagrama da figura 3.3 e o detalhou conforme suas atividades.

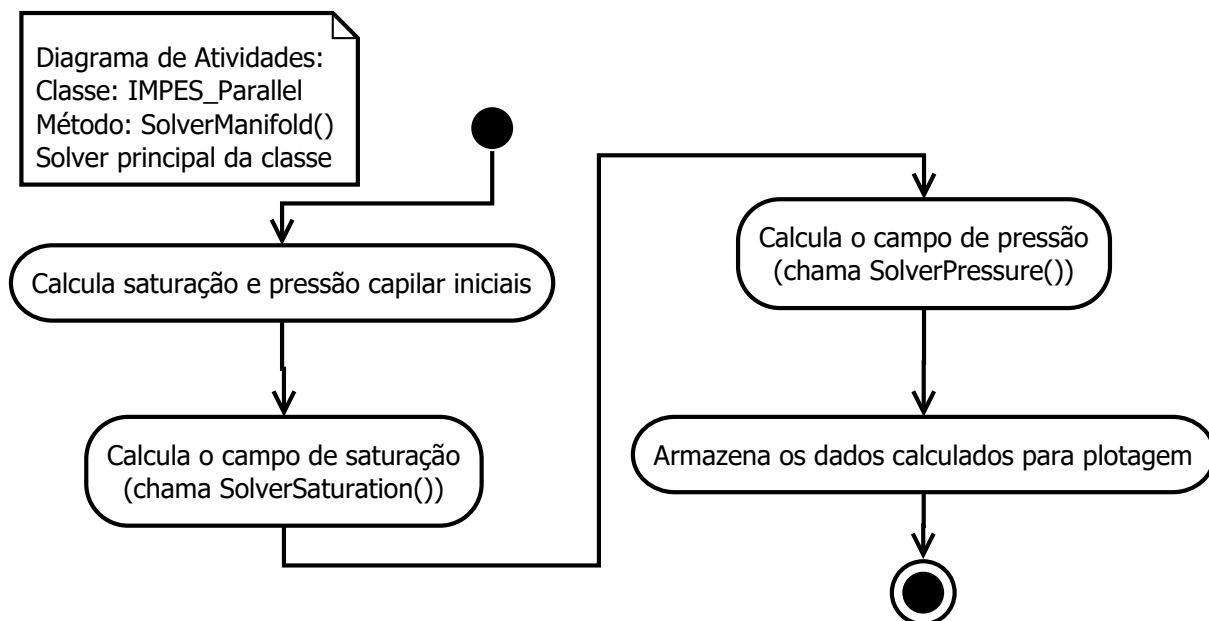


Figura 3.4: Diagrama de atividades

Capítulo 4

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.

4.1 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.

1. Protocolos

- Definição dos protocolos de comunicação entre os diversos elementos externos (como dispositivos). Por exemplo: se o sistema envolve o uso dos nós de um cluster, devem ser considerados aspectos como o protocolo de comunicação entre os nós do cluster.
 - Esta versão do software comunica-se com o programa externo gnuplot.
- Definição dos protocolos de comunicação entre os diversos elementos internos (como objetos).
 - Não se aplica.
- Definição do formato dos arquivos gerados pelo software. Por exemplo: prefira formatos abertos, como arquivos txt e xml.

- Neste projeto, o software gera dois tipos de arquivos: .txt (arquivo de texto) e .bmp (gráfico Gnuplot).

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.
 - A alocação de memória RAM e HD além do uso de periféricos (mouse e teclado) serão feitas de maneira automática pelo compilador em conjunto com o sistema operacional. Quanto ao uso da CPU, haverá o uso de múltiplas threads trabalhando em paralelo para o cálculo dos campos de saturação e pressão.
- Identificação da necessidade do uso de banco de dados. Implicam em modificações nos diagramas de atividades e de componentes.
 - Neste projeto não há uso de um banco de dados como, por exemplo, o SQLite3, MongoDB e o MS Access.
- Identificação da necessidade de sistemas de armazenamento de massa. Por exemplo: um *storage* em um sistema de cluster ou sistemas de backup.
 - Neste projeto não há a necessidade de armazenagem de dados em um sistema de armazenamento de massa.

3. Controle

- Identificação e seleção da implementação de controle, seqüencial ou concorrente, baseado em procedimentos ou eventos. Implicam modificações no diagrama de execução.
 - Neste projeto há a implementação de controle concorrente.
- Identificação da necessidade de otimização. Por exemplo: prefira sistemas com grande capacidade de memória; prefira vários hds pequenos a um grande.
 - Este projeto já é um projeto de otimização do algoritmo IMPES utilizando formulações que permitem sua paralelização. Além disso foi implementado o algoritmo de Thomas para resolução mais rápida dos sistemas lineares gerados para a solução do campo de pressão.
- Identificação de concorrências – quais algoritmos podem ser implementados usando processamento paralelo.
 - Os algoritmos que podem ser paralelizados são os que estão nos métodos: IMPES::SolverPressure(), IMPES::SolverSaturation.

4. Plataformas

- Identificação e definição das plataformas a serem suportadas: hardware, sistema operacional e linguagem de software.
 - Este projeto não é multiplataforma, tendo o desenvolvimento focado para a plataforma Windows.
 - A linguagem de software é a linguagem C++ 17.
- Seleção das bibliotecas externas a serem utilizadas.
 - Armadillo, para resolução de sistemas lineares.
- Seleção da biblioteca utilizada para montar a interface gráfica do software – GUI.
 - Não há interface gráfica para este software.
- Seleção do ambiente de desenvolvimento para montar a interface de desenvolvimento – IDE.
 - Para a plataforma Windows, temos o seguinte ambiente de desenvolvimento: Desktop com processador AMD FX-6100, 8GB RAM DDR3 e placa gráfica AMD Radeon RX 470. O programa será escrito no software Microsoft Visual Studio com o compilador MSVC.

5. Padrões de projeto

- Normalmente os padrões de projeto são identificados e passam a fazer parte de uma biblioteca de padrões da empresa. Mas isto só ocorre após a realização de diversos projetos.
 - Não se aplica.

4.2 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 softwareção). 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.

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.

Efeitos do projeto no modelo estrutural

- Estabelecer as dependências e restrições associadas à plataforma escolhida.
 - Na plataforma Windows, o gnuplot necessita estar instalado para o correto funcionamento do software.

Efeitos do projeto no modelo dinâmico

- Não foi necessário nesta etapa do projeto.

Efeitos do projeto nos atributos

- Não foi necessário nesta etapa do projeto.

Efeitos do projeto nos métodos

- Não foi necessário nesta etapa do projeto.

Efeitos do projeto nas heranças

- As relações de herança continuam inalteradas.

Efeitos do projeto nas associações

- Não foi necessário nesta etapa do projeto.

Efeitos do projeto nas otimizações

- Não foi necessário nesta etapa do projeto.

4.3 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 4.1 o diagrama de componentes para este software: as classes IMPES, IMPES_Parallel e IMPES_ParallelCV são dependentes da classe RCore para obter os dados para realizarem seus cálculos. A classe IMPES utiliza algoritmos da biblioteca *Armadillo* para realizar a solução de sistema lineares. As classes IMPES_Parallel e IMPES_ParallelCV dependem da classe `std::thread` para implementar o processamento paralelo em múltiplas threads, sendo necessária a linkagem com a biblioteca (-lpthread) no momento da compilação. Além disso, todas as classes dependem de classes da biblioteca padrão tais como: `string`, `vector`, `mutex`, `conditional_variable`, `iostream`; para realizarem seus cálculos. Por fim, os métodos na classe IMPES para plotagem dependem da classe CGnuplot.

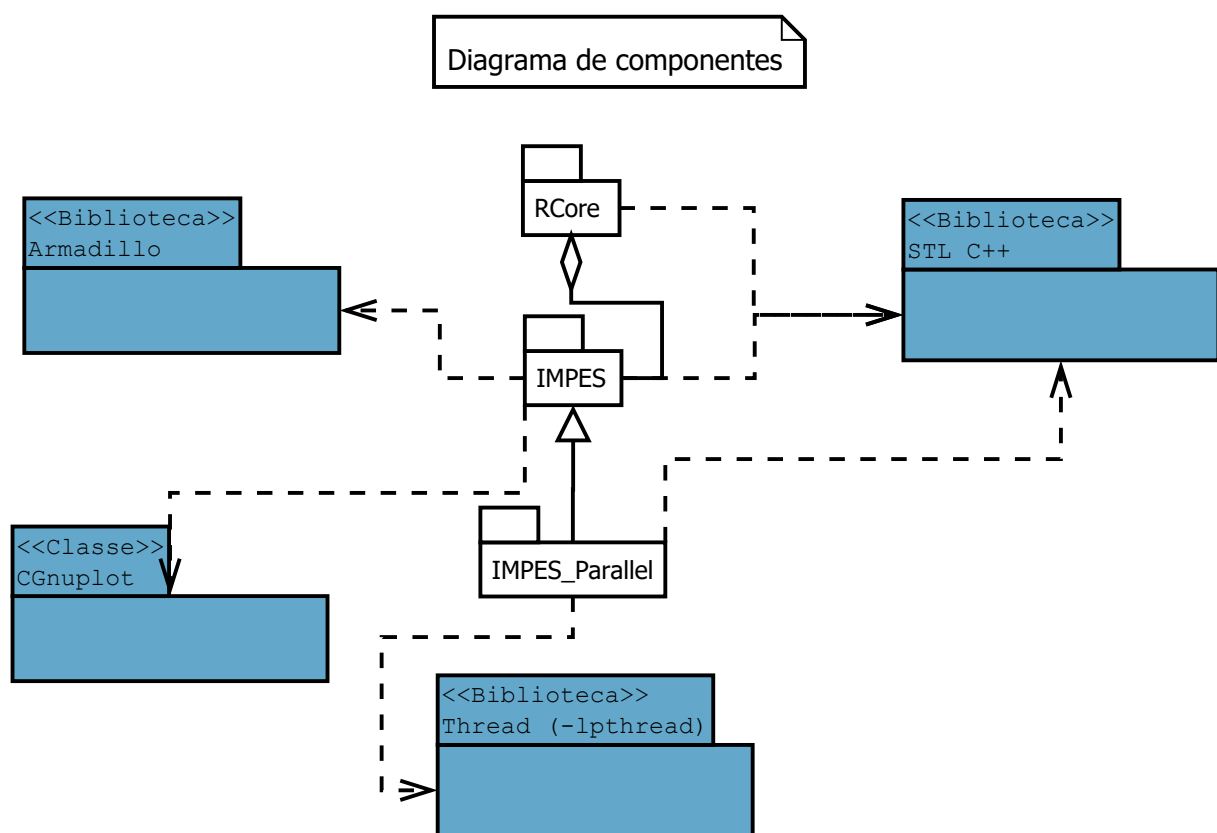


Figura 4.1: Diagrama de componentes

4.4 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.

Veja na Figura 4.2 o diagrama de implantação deste software. É possível notar que não só é preciso um computador convencional com mouse, teclado e monitor mas é preciso uma coleta de dados petrofísicos e do experimento, sem os quais não há como realizar os cálculos.

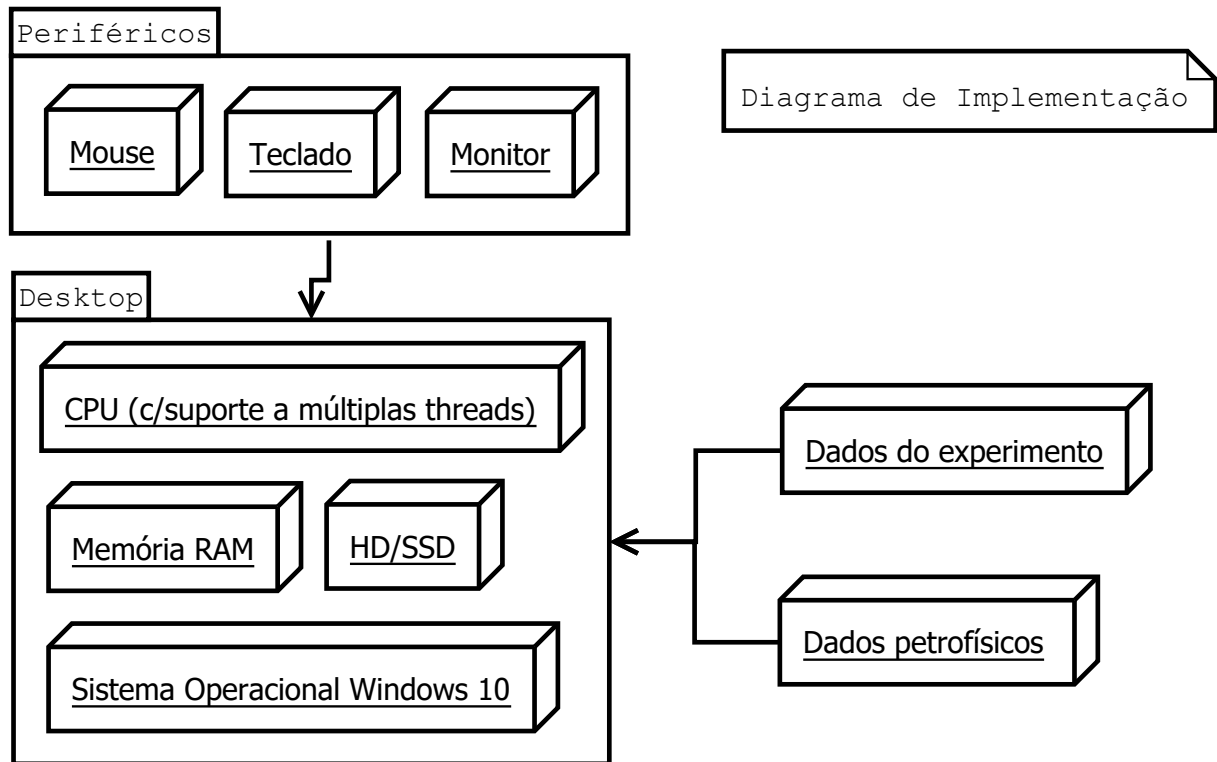


Figura 4.2: Diagrama de implantação.

Capítulo 5

Implementação

Neste capítulo do projeto de engenharia apresentamos os códigos fonte que foram desenvolvidos.

5.1 Código fonte

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

Apresenta-se na listagem 5.1 o arquivo com código da classe IMPES.

Listing 5.1: Arquivo de cabeçalho da classe IMPES.

```
1 #pragma once
2 #include <iostream>
3 #include <string>
4 #include <vector>
5 #include <algorithm>
6 #include <numeric>
7 #include <assert.h>
8 #include <map>
9 #include <exception>
10 #include <ctime>
11 #include <thread>
12
13 #include "CGnuplot.h"
14 #include "RCore.h"
15
16 struct IMPESArgs {
17     double Patm;
18     double vazao_inj;
19     double uo;
20     double uw;
21     double DsMax;
22     double vpi1;
23     double vpi2;
```

```

24         double si;
25         int nbl;
26 };
27
28 class IMPES {
29 protected:
30     bool drenagem;
31     bool embebicao;
32     int nbl;
33     double uw;
34     double uo;
35     double Pl;
36     double Qinj;
37     double DSMAX;
38     double h;
39     double vpi1;
40     double vpi2;
41     double t_drenagem;
42     double t_embebicao;
43     double tempo_sim;
44     double si;
45
46     std::shared_ptr<RCore> core;
47
48     std::vector<std::vector<double>> pc;
49     std::vector<std::vector<double>> p;
50     std::vector<std::vector<double>> s;
51     std::vector<double> g;
52
53     std::vector<std::string> tempo_plot;
54     std::vector<std::string> tempo_plot_emb;
55     std::vector<std::vector<double>> Plot_s;
56     std::vector<std::vector<double>> Plot_p;
57     std::vector<std::vector<double>> Plot_s_emb;
58     std::vector<std::vector<double>> Plot_p_emb;
59     std::vector<double> Plot_h;
60
61 public:
62     IMPES(std::shared_ptr<RCore>& ncore, IMPESArgs ia);
63     void Grid();
64     virtual void SolverManifold();
65     virtual void SolverPressure(size_t n);
66     virtual double SolverSaturation1(double n);
67     virtual double SolverSaturation2(double n);
68     double SatEq1(size_t n, size_t i);
69     double SatEq2(size_t n, size_t i);
70     std::vector<double> PreEq(size_t n, size_t i);
71     std::vector<double> ThomasSolver(const std::vector<double>& a,

```

```

        const std::vector<double>& b, std::vector<double>& c, std::
        vector<double>& d);
72    double upstream_weighting(const size_t& n, const size_t& i,
        const int& orientation);
73    double u();
74    double dt();
75    std::pair<double, double> tempo_inj();
76    void PlotSat(bool multiplot = false);
77    void PlotPre();
78
79};

```

Apresenta-se na listagem 5.2 o arquivo de implementação da classe IMPES.

Listing 5.2: Arquivo de implementação da classe IMPES.

```

80#include "IMPES.h"
81
82IMPES::IMPES(std::shared_ptr<RCore>& ncore, IMPESArgs ia)
83    : core(ncore), uw(ia.uw), uo(ia.uo), Pl(ia.Patm), Qinj(ia.
        vazao_inj), DSMAX(ia.DsMax), nbl(ia.nbl), vpi1(ia.vpi1), vpi2
        (ia.vpi2), si(ia.si) {
84    std::pair<double, double> tempos = tempo_inj();
85    t_drenagem = tempos.first;
86    t_embebicao = tempos.second;
87}
88void IMPES::Grid() {
89    h = core->length / nbl;
90    for (double k = h; k < nbl; k += h) {
91        Plot_h.push_back(k - h / 2);
92    }
93    std::vector<double> aux(nbl, 0);
94    g = aux;
95}
96void IMPES::SolverManifold() {
97    clock_t time_drenagem = clock();
98    try {
99        //Calculando a saturação S0 inicial de todos os blocos;
100        std::vector<double> s0(nbl, si);
101        std::vector<double> pc0(nbl, core->Pct);
102        s.push_back(s0);
103        Plot_s.push_back(s0);
104        tempo_plot.push_back(std::to_string(0));
105        pc.push_back(pc0);
106        //PROBLEMA AQUI
107        SolverPressure(0);
108        size_t n = 1;
109        drenagem = true;
110        double t; double t_fim = 0;
111        for (double t_sim = 0; t_sim < t_drenagem; t_sim += t) {

```

```

112         t = SolverSaturation1(n);
113         for (int j = 0; j < nbl; j++) {
114             pc0[j] = core->pc(s[n][j]);
115         }
116         pc.push_back(pc0);
117         SolverPressure(n);
118         n += 1;
119         if (n < 3000 && n % 500 == 0) {
120             Plot_p.push_back(p[n - 1]);
121             Plot_s.push_back(s[n - 1]);
122             tempo_plot.push_back(std::to_string(
123                 t_fim));
124             std::cout << "Terminou-se o passo de
125                 tempo" << n << " com Dt=" << t <<
126                 '\n';
127             std::cout << "Tempo da simulação total:
128                 " << t_sim << "\n";
129         }
130         else if (n > 3000 && n % 2000 == 0) {
131             Plot_p.push_back(p[n - 1]);
132             Plot_s.push_back(s[n - 1]);
133             tempo_plot.push_back(std::to_string(
134                 t_fim));
135             std::cout << "Terminou-se o passo de
136                 tempo" << n << " com Dt=" << t <<
137                 '\n';
138             std::cout << "Tempo da simulação total:
139                 " << t_sim << "\n";
140         }
141         t_fim = t_sim;
142     }
143
144     drenagem = false;
145     std::cout << "TEMPO_FINAL_DRENAGEM:" << (clock() -
146         time_drenagem) / CLOCKS_PER_SEC << '\n';
147     std::vector<double> semb(nbl, core->swi);
148     s.push_back(semb);
149     for (int j = 0; j < nbl; j++) {
150         pc0[j] = core->pcMAX;
151     }
152     pc.push_back(pc0);
153     SolverPressure(n);
154     n += 1;
155     Plot_s.push_back(s[n - 1]);
156     Plot_p.push_back(p[n - 1]);
157     Plot_s_emb.push_back(s[n - 1]);
158     Plot_p_emb.push_back(p[n - 1]);
159     tempo_plot_emb.push_back(std::to_string(t_drenagem));

```

```

151         tempo_plot.push_back(std::to_string(t_drenagem));
152         std::cout << '\n';
153         std::cout << "Terminou-se o passo de tempo" << n << "
            com Dt=" << t << '\n';
154         std::cout << "Tempo da simulação total:" << t_fim << "
            s\n";

155
156         embebicao = true;
157         if (embebicao) {
158             for (double t_sim = t_drenagem; t_sim <
                t_embebicao; t_sim += t) {
159                 t = SolverSaturation1(n);
160                 for (int j = 0; j < nbl; j++) {
161                     pc0[j] = core->pc(s[n][j]);
162                 }
163                 pc.push_back(pc0);
164                 SolverPressure(n);
165                 n += 1;
166                 if (n % 100 == 0) {
167                     Plot_s_emb.push_back(s[n - 1]);
168                     Plot_p_emb.push_back(p[n - 1]);
169                     tempo_plot_emb.push_back(std::
                        to_string(t_sim));
170                     std::cout << '\n';
171                     std::cout << "Terminou-se o
                        passo de tempo" << n << "
                        com Dt=" << t << '\n';
172                     std::cout << "Tempo da simulação
                        total:" << t_sim << "s\n";
173                 }
174                 t_fim = t_sim;
175             }
176         }
177         embebicao = false;
178         std::cout << "TEMPO_FINAL_EMBEBICAO" << '\n';
179
180         Plot_s_emb.push_back(s[n - 1]);
181         Plot_p_emb.push_back(p[n - 1]);
182         tempo_plot_emb.push_back(std::to_string(t_embebicao));
183     }
184     catch (std::exception& e) {
185         std::cerr << "Erro no método IMPES::SolverManifold() - "
            << e.what();
186     }
187 }

188 void IMPES::SolverPressure(size_t n) {
189     std::vector<double> a(nbl, 0.0);
190     std::vector<double> b(nbl, 0.0);

```

```

191     std::vector<double> c(nbl, 0.0);
192     std::vector<double> d(nbl, 0.0);
193     std::vector<double> coefficients;
194     for (int i = 0; i < nbl; i++) {
195         coefficients = PreEq(n, i);
196         if (i > 0) {
197             a[i] = coefficients[0];
198         }
199         if (i < nbl - 1) {
200             c[i] = coefficients[2];
201         }
202         b[i] = coefficients[1];
203         d[i] = coefficients[3];
204     }
205
206     p.push_back(ThomasSolver(a, b, c, d));
207
208
209 }
210 double IMPES::SolverSaturation1(double n) {
211     std::vector<double> s_n1(nbl);
212     g.clear(); g.resize(nbl);
213     for (int i = 0; i < nbl; i++) {
214         SatEq2(n, i);
215     }
216     double t = dt();
217     try {
218         for (int i = 0; i < nbl; ++i) {
219             s_n1[i] = g[i] * t + s[n - 1][i];
220         }
221         s.push_back(s_n1);
222     }
223     catch (std::exception& e) {
224         std::cerr << "Erro no método IMPES::SolverSaturation" <<
                e.what() << '\n';
225     }
226     return t;
227 }
228 double IMPES::SolverSaturation2(double n) {
229     std::vector<double> s_n1(nbl);
230     double t = dt();
231     g.clear(); g.resize(nbl);
232     double gi;
233     for (int i = 0; i < nbl; i++) {
234         gi = SatEq1(n, i);
235         s_n1[i] = gi * t + s[n - 1][i];
236     }
237     s.push_back(s_n1);

```

```

238         return dt();
239     }
240     double IMPES::SatEq1(size_t n, size_t i) {
241         double gi;
242         if (i == 0) {
243             double Sp = upstream_weighting(n, i, 1);
244             double Ao_POS = core->lambda_o(Sp, uo);
245             if (drenagem) {
246                 gi = (-core->permeability / (core->porosity *
                pow(h, 2))) * (Ao_POS * (p[n - 1][i + 1] - p[
                n - 1][i]) + (u() * h / core->permeability));
247             }
248             else if (embebicao) {
249                 gi = (-core->permeability / (core->porosity *
                pow(h, 2))) * (Ao_POS * (p[n - 1][i + 1] - p[
                n - 1][i]));
250             }
251         }
252         else if (i == nbl - 1) {
253             double Sp = upstream_weighting(n, i, 1);
254             double Sn = upstream_weighting(n, i, -1);
255             double Ao_POS = core->lambda_o(Sp, uo);
256             double Ao_NEG = core->lambda_o(Sn, uo);
257             gi = (-4 * core->permeability / (3 * core->porosity * h
                * h)) * (2 * Ao_POS * (Pl - p[n - 1][i]) - Ao_NEG * (
                p[n - 1][i] - p[n - 1][i - 1]));
258         }
259         else {
260             double Sp = upstream_weighting(n, i, 1);
261             double Sn = upstream_weighting(n, i, -1);
262             double Ao_POS = core->lambda_o(Sp, uo);
263             double Ao_NEG = core->lambda_o(Sn, uo);
264             gi = (-core->permeability / (core->porosity * pow(h, 2))
                ) * (Ao_POS * (p[n - 1][i + 1] - p[n - 1][i]) -
                Ao_NEG * (p[n - 1][i] - p[n - 1][i - 1]));
265         }
266         g[i] = gi;
267         return gi;
268     }
269 }
270     double IMPES::SatEq2(size_t n, size_t i) {
271         double gi = 0;
272         if (i == 0) {
273             double ts = upstream_weighting(n, i, 1);
274             double fwPOS = core->fw(ts, uw, uo);
275             double lamb_oPOS = core->lambda_o(ts, uo);
276             double T = fwPOS * lamb_oPOS;
277             if (drenagem) {

```



```

278         gi = -1 / core->porosity * (core->permeability /
279             pow(h, 2) * (T * (pc[n - 1][i + 1] - pc[n -
280                 1][i])) + u() / h * fwPOS);
281     }
282     else if (embebicao) {
283         gi = -1 / core->porosity * (core->permeability /
284             pow(h, 2) * (T * (pc[n - 1][i + 1] - pc[n -
285                 1][i])) + u() / h * (fwPOS - 1));
286     }
287 }
288 else if (i == nbl - 1) {
289     double tsp = upstream_weighting(n, i, 1);
290     double tsn = upstream_weighting(n, i, -1);
291     double fwPOS = core->fw(tsp, uw, uo);
292     double fwNEG = core->fw(tsn, uw, uo);
293     double lamb_oPOS = core->lambda_o(tsp, uo);
294     double lamb_oNEG = core->lambda_o(tsn, uo);
295     double T = fwNEG * lamb_oNEG;
296     gi = -1 / core->porosity * ((2 * u() / h) * (fwPOS -
297         fwNEG) + ((4 * core->permeability) / (3 * pow(h, 2)))
298         * ((2 * fwPOS * lamb_oPOS * (Pl - pc[n - 1][i])) - (
299             T * (pc[n - 1][i] - pc[n - 1][i - 1]))));
300 }
301 else {
302     double tsp = upstream_weighting(n, i, 1);
303     double tsn = upstream_weighting(n, i, -1);
304     double fwPOS = core->fw(tsp, uw, uo);
305     double fwNEG = core->fw(tsn, uw, uo);
306     double lamb_oPOS = core->lambda_o(tsp, uo);
307     double lamb_oNEG = core->lambda_o(tsn, uo);
308     gi = -1 / core->porosity * (core->permeability / pow(h,
309         2) * (lamb_oPOS * fwPOS * pc[n - 1][i + 1] - (
310             lamb_oNEG * fwNEG + lamb_oPOS * fwPOS) * pc[n - 1][i]
311             + lamb_oNEG * fwNEG * pc[n - 1][i - 1]) + u() / h *
312             (fwPOS - fwNEG));
313 }
314 g[i] = gi;
315 return gi;
316 }
317 std::vector<double> IMPES::PreEq(size_t n, size_t i) {
318     try {
319         std::vector<double> c(4, 0);
320         //c[0] p_i-1
321         //c[1] p_i
322         //c[2] p_i+1
323         //c[3] lado direito da eq;
324         if (i == 0) {

```

```

315         double tsp = upstream_weighting(n, i, 1);
316         double lambda_wPOS = core->lambda_w(tsp, uw);
317         double lambda_tPOS = core->lambda_t(tsp, uw, uo)
318         ;
319         c[0] = 0;
320         c[1] = -lambda_tPOS;
321         c[2] = lambda_tPOS;
322         c[3] = ((-u() * h / core->permeability) +
323             lambda_wPOS * (pc[n][i + 1] - pc[n][i]));
324     }
325     else if (i == nbl - 1) {
326         double tsp = upstream_weighting(n, i, 1);
327         double tsn = upstream_weighting(n, i, -1);
328         double lambda_wPOS = core->lambda_w(tsp, uw);
329         double lambda_wNEG = core->lambda_w(tsn, uw);
330         double lambda_tPOS = core->lambda_t(tsp, uw, uo)
331         ;
332         double lambda_tNEG = core->lambda_t(tsn, uw, uo)
333         ;
334         c[0] = lambda_tNEG;
335         c[1] = -(2 * lambda_tPOS + lambda_tNEG);
336         c[2] = 0;
337         c[3] = -2 * lambda_tPOS * P1 - (2 * lambda_wPOS
338             + lambda_wNEG) * pc[n][i] + lambda_wNEG * pc[
339             n][i - 1];
340     }
341     else {
342         double tsp = upstream_weighting(n, i, 1);
343         double tsn = upstream_weighting(n, i, -1);
344         double lambda_wPOS = core->lambda_w(tsp, uw);
345         double lambda_wNEG = core->lambda_w(tsn, uw);
346         double lambda_tPOS = core->lambda_t(tsp, uw, uo)
347         ;
348         double lambda_tNEG = core->lambda_t(tsn, uw, uo)
349         ;
350         c[0] = lambda_tNEG;
351         c[1] = -(lambda_tNEG + lambda_tPOS);
352         c[2] = lambda_tPOS;
353         c[3] = (lambda_wPOS * pc[n][i + 1] - (
354             lambda_wPOS + lambda_wNEG) * pc[n][i] +
355             lambda_wNEG * pc[n][i - 1]);
356     }
357     return c;
358 }
359 catch (std::exception& e) {
360     std::cerr << "Erro no método IMPES::PreEq() << e.
361     what();

```

```

352     }
353 }
354 std::vector<double> IMPES::ThomasSolver(const std::vector<double>& a,
      const std::vector<double>& b, std::vector<double>& c, std::vector<
      double>& d) {
355     size_t N = d.size();
356
357     c[0] = c[0] / b[0];
358     d[0] = d[0] / b[0];
359
360     double m;
361     for (int i = 1; i < N; i++) {
362         m = 1.0 / (b[i] - a[i] * c[i - 1]);
363         c[i] = c[i] * m;
364         d[i] = (d[i] - a[i] * d[i - 1]) * m;
365     }
366     for (int i = N - 1; i-- > 0;) {
367         d[i] = d[i] - (c[i] * d[i + 1]);
368     }
369     return d;
370 }
371 double IMPES::upstream_weighting(const size_t& n, const size_t& i, const
      int& orientation) {
372     if (n == 0) {
373         return s[n][i];
374     }
375     if (orientation == 1) {
376         return s[n - 1][i];
377     }
378     else if (orientation == -1) {
379         return s[n - 1][i - 1];
380     }
381 }
382 double IMPES::u() {
383     return Qinj / (core->Area());
384 };
385 double IMPES::dt() {
386     assert(g.size() != 0);
387     std::vector<double> absG;
388     for (double& e : g) {
389         absG.push_back(abs(e));
390     }
391     double t = DSMAX / *std::max_element(absG.begin(), absG.end());
392     if (isinf(t)) {
393         return DSMAX;
394     }
395     if (drenagem) {
396         if (tempo_sim + t > t_drenagem) {

```

```

397         return t_drenagem - tempo_sim;
398     }
399     else if (t > 20) { return t / 100; }
400     else return t;
401 }
402 else if (embebicao) {
403     if (tempo_sim + t > t_embebicao) {
404         return t_embebicao - tempo_sim;
405     }
406     else return t;
407 }
408 }
409 std::pair<double, double> IMPES::tempo_inj() {
410     double param1 = core->volPoroso() / Qinj;
411     return std::make_pair<double, double>(vpi1 * param1, vpi2 *
        param1);
412 }
413 void IMPES::PlotSat(bool multiplot) {
414     //PLOT DRENAGEM
415     try {
416         CGnuplot plot; std::string title;
417         int NUM_PLOTS = Plot_s.size();
418         std::cout << "Número total de plots: " << NUM_PLOTS << '
            \n';
419
420         std::ofstream tmpfile("sat.txt");
421         for (int i = 0; i < nbl; ++i) {
422             tmpfile << Plot_h[i] << "\t";
423             for (int j = 0; j < NUM_PLOTS; ++j) {
424                 tmpfile << Plot_s[j][i] << "\t";
425             }
426             tmpfile << '\n';
427         }
428         tmpfile.close();
429
430         plot << "set xlabel \"X\"";
431         plot << "set ylabel \"Sw\"";
432
433         if (multiplot) {
434             int plots = Plot_s.size() / 6;
435             plots += 1;
436             size_t j = 2;
437             for (int p = 1; p < plots; p++) {
438                 plot << "set multiplot layout 3,2 title
                    \"Distribuicao de Saturacao no Meio
                    Poroso - DRENAGEM\"";
439                 while (j < 2 + (p * 6)) {
440                     title = tempo_plot[j - 2];

```

```

441         //plot_command += ", 'sat.txt'
            using 1:" + std::to_string(j)
            + "title \"Tempo\"+std::
            to_string(j)+\"\" with lines";
442 plot << "set_xlabel \"X(t= \" +
            title + "s)\";";
443 plot << "plot 'sat.txt' using 1:
            \" + std::to_string(j) + \"w
            filledcu above y1=0 lc rgb \"
            blue \" notitle, 'sat.txt'
            using 1:\" + std::to_string(j)
            + \"w filledcu below y1=1 lc
            rgb \"black \" notitle";
444         j += 1;
445     }
446     plot << "unset multiplot";
447     std::cout << "J- WHILE: \" << j << '\n';
448     system("Pause");
449 }
450 std::cout << "J: \" << j << '\n';
451 plot << "set multiplot layout 3,2 title \"
            Distribuicao de Saturacao no Meio Poroso -
            DRENAGEM\"";
452 for (int k = j; k <= Plot_s.size() + 1; ++k) {
453     //plot_command += ", 'sat.txt' using 1:\"
            + std::to_string(j) + "title \"Tempo
            \"+std::to_string(j)+\"\" with lines";
454     plot << "set_xlabel \"X(t= \" + title +
            \"s)\";";
455     plot << "plot 'sat.txt' using 1:\" + std
            ::to_string(k) + \"w filledcu above
            y1=0 lc rgb \"blue \" notitle, 'sat.
            txt' using 1:\" + std::to_string(j) +
            \"w filledcu below y1=1 lc rgb \"
            black \" notitle";
456 }
457 plot << "unset multiplot";
458 system("Pause");
459 }
460 else {
461     plot << "set title \"Distribuicao de Saturacao
            no Meio Poroso - DRENAGEM\"";
462     plot << "set yrange [0:1]";
463     plot << "plot for [i=2: \" + std::to_string(Plot_s.
            size()) + \"] 'sat.txt' using 1: i notitle with
            lines";
464     system("Pause");
465 }

```

```

466         //system("erase sat.txt");
467     }
468     catch (std::exception& e) {
469         std::cerr << e.what() << '\n';
470     }
471     //PLOT EMBEBICA0
472     try {
473         CGnuplot emb_plot; std::string title;
474         int NUM_PLOTS = Plot_s_emb.size();
475         std::cout << "Número total de plots: " << NUM_PLOTS << '\n';
476         std::ofstream tmpfile("sat_emb.txt");
477         for (int i = 0; i < nbl; ++i) {
478             tmpfile << Plot_h[i] << "\t";
479             for (int j = 0; j < NUM_PLOTS; ++j) {
480                 tmpfile << Plot_s_emb[j][i] << "\t";
481             }
482             tmpfile << '\n';
483         }
484         tmpfile.close();
485         emb_plot << "set xlabel \"X\"";
486         emb_plot << "set ylabel \"So\"";
487         if (multiplot) {
488             int plots = Plot_s_emb.size() / 6;
489             plots += 1;
490             int j = 2;
491             for (int p = 1; p < plots; p++) {
492                 emb_plot << "set multiplot layout 3,2"
493                     << "title \"Distribuicao de Saturacao no"
494                     << "Meio Poroso - EMBEBICA0\"";
495                 while (j < 2 + (p * 6)) {
496                     //plot_command += ", 'sat.txt'
497                         using 1: " + std::to_string(j)
498                         + " title \"Tempo\" + std::
499                             to_string(j) + \" with lines\";
500                     title = tempo_plot_emb[j - 2];
501                     emb_plot << "set xlabel \"X(t=\"
502                         + title + "s)\"";
503                     emb_plot << "plot 'sat_emb.txt'
504                         using 1: " + std::to_string(j)
505                         + " w filledcu above y1=0 lc
506                             rgb \"blue\" notitle,
507                         'sat_emb.txt' using 1: " + std
508                             ::to_string(j) + " w filledcu
509                             below y1=1 lc rgb \"black\"
510                             notitle\";
511                     j += 1;
512                 }
513             }
514         }
515     }

```

```

500         emb_plot << "unset_multiplot";
501         std::cout << "J- WHILE:" << j << '\n';
502         system("Pause");
503     }
504     std::cout << "J:" << j << '\n';
505     emb_plot << "set_multiplot_layout_3,2_title \"
        Distribuicao de Saturacao no Meio Poroso-
        DRENAGEM\"";
506     for (int k = j; k <= Plot_s_emb.size() + 1; ++k)
        {
507         //plot_command += ", 'sat.txt' using 1:"
            + std::to_string(j) + "title \"Tempo
            \"+std::to_string(j)+\" \" with lines";
508         title = tempo_plot_emb[k - 2];
509         emb_plot << "set_xlabel \"X(t= \" +
            title + "s)\";
510         emb_plot << "plot 'sat_emb.txt' using 1:
            \" + std::to_string(k) + \"w filledcu
            above_y1=0 lc rgb \"blue\" notitle, '
            sat_emb.txt' using 1: \" + std::
            to_string(j) + \"w filledcu below_y1
            =1 lc rgb \"black\" notitle";
511     }
512     emb_plot << "unset_multiplot";
513     system("Pause");
514 }
515 else {
516     emb_plot << "set_title \"Distribuicao de
        Saturacao no Meio Poroso- EMBEBICAO\"";
517     emb_plot << "set_yrange [0:1]";
518     emb_plot << "plot for[i=2: \" + std::to_string(
        Plot_s_emb.size()) + \"] 'sat_emb.txt' using
        1:i notitle with lines";
519     system("Pause");
520 }
521 //system("erase sat.txt");
522 }
523 catch (std::exception& e) {
524     std::cerr << e.what() << '\n';
525 }
526
527}
528 void IMPES::PlotPre() {
529     //PLOT DRENAGEM
530     try {
531         CGnuplot plot;
532         int NUM_PLOTS = Plot_p.size();
533         std::cout << "Número total de plots:" << NUM_PLOTS << '

```

```

        \n';
534     std::ofstream tmpfile("pre_dre.txt");
535     for (int i = 0; i < nbl; ++i) {
536         tmpfile << Plot_h[i] << "\t";
537         for (int j = 0; j < NUM_PLOTS; ++j) {
538             tmpfile << Plot_p[j][i] << "\t";
539         }
540         tmpfile << '\n';
541     }
542     tmpfile.close();
543     plot << "set_xlabel \"X\"";
544     plot << "set_ylabel \"P(atm)\"";
545     plot << "set_title \"Distribuicao de Pressao no Meio
        Poroso - DRENAGEM\"";
546     plot << "plot for [i=2:" + std::to_string(Plot_p.size() +
        1) + "]" _'pre_dre.txt' using 1:i title 'T'.(i-1) with
        lines";
547     system("Pause");
548     //system("erase sat.txt");
549 }
550 catch (std::exception& e) {
551     std::cerr << e.what() << '\n';
552 }
553 //PLOT EMBEBICAO
554 try {
555     CGnuplot emb_plot;
556     int NUM_PLOTS = Plot_p_emb.size();
557     std::ofstream tmpfile("pre_emb.txt");
558     for (int i = 0; i < nbl; ++i) {
559         tmpfile << Plot_h[i] << "\t";
560         for (int j = 0; j < NUM_PLOTS; ++j) {
561             tmpfile << Plot_p_emb[j][i] << "\t";
562         }
563         tmpfile << '\n';
564     }
565     tmpfile.close();
566     emb_plot << "set_xlabel \"X\"";
567     emb_plot << "set_ylabel \"P(atm)\"";
568     emb_plot << "set_title \"Distribuicao de Pressao no Meio
        Poroso - EMBEBICAO\"";
569     emb_plot << "plot for [i=2:" + std::to_string(Plot_p_emb.
        size()) + "]" _'pre_emb.txt' using 1:i title 'T'.(i-1)
        with lines";
570     system("Pause");
571     //system("erase sat.txt");
572 }
573 catch (std::exception& e) {
574     std::cerr << e.what() << '\n';

```



```

575     }
576
577 }

```

Apresenta-se na listagem 5.3 o arquivo com código da classe IMPESParallel.

Listing 5.3: Arquivo de cabeçalho da classe IMPESParallel.

```

578 #pragma once
579
580 #include "CGnuplot.h"
581 #include "RCore.h"
582 #include "IMPES.h"
583 #include <future>
584
585 class IMPES_Parallel : public IMPES {
586 protected:
587     std::vector<std::shared_ptr<std::thread>> threads;
588     std::vector<double> s_n1;
589     std::vector<double> pc0;
590     std::vector<int> wkload;
591     int num_of_threads;
592     bool isPR1;
593
594 public:
595     IMPES_Parallel(std::shared_ptr<RCore>& ncore, IMPESArgs ia, bool
        _isPR1);
596     void SolverManifold() override;
597     //void parallelPressureSolver(size_t start, size_t end);
598     void parallelPressureSolver(std::vector<int> wkload);
599     //void SolverPressure(size_t n) override;
600     void SolverPressure(int start, int end);
601     double SolverSaturation1(double n) override;
602     double SolverSaturation2(double n) override;
603     void SatIncrementParallel(int& start, int& end, double& n);
604     static std::vector<int> Workload(size_t tasks, int
        num_of_workers = std::thread::hardware_concurrency());
605 };

```

Apresenta-se na listagem 5.4 o arquivo de implementação da classe IMPESParallel.

Listing 5.4: Arquivo de implementação da classe IMPESParallel.

```

606 #include "IMPESParallel.h"
607
608 IMPES_Parallel::IMPES_Parallel(std::shared_ptr<RCore>& ncore, IMPESArgs
    ia, bool _isPR1) : IMPES(ncore, ia), threads(std::thread::
    hardware_concurrency(), nullptr), s_n1(nbl), pc0(nbl) {
609     num_of_threads = std::thread::hardware_concurrency();
610     wkload = Workload(nbl, num_of_threads);
611     isPR1 = _isPR1;

```

```

612};
613void IMPES_Parallel::SolverManifold() {
614    //Calculando a saturação S0 inicial de todos os blocos;
615    std::vector<int> dre;
616    std::vector<int> emb;
617    std::vector<double> s0(nbl, si);
618    std::vector<double> pc0(nbl, core->Pct);
619    s.push_back(s0);
620    Plot_s.push_back(s0);
621    tempo_plot.push_back(std::to_string(0));
622    pc.push_back(pc0);
623    //PROBLEMA AQUI
624    size_t n = 1;
625    drenagem = true;
626
627    double t; double t_fim = 0;
628    for (double t_sim = 0; t_sim < t_drenagem; t_sim += t) {
629        if (isPR1) {
630            t = SolverSaturation1(n);
631        }
632        else {
633            t = SolverSaturation2(n);
634        }
635        n += 1;
636        if (n % 2000 == 0) {
637            dre.push_back(n-1);
638            std::cout << "Terminou-se o passo de tempo " <<
639                n << " com Dt = " << t << '\n';
640            std::cout << "Tempo da simulação total: " <<
641                t_sim << "s\n";
642        }
643        t_fim = t_sim;
644    }
645    dre.push_back(n-1);
646    drenagem = false;
647    embebicao = true;
648    if (embebicao) {
649        for (double t_sim = t_drenagem; t_sim < t_embebicao;
650            t_sim += t) {
651            if (isPR1) {
652                t = SolverSaturation1(n);
653            }
654            else {
655                t = SolverSaturation2(n);
656            }
657            n += 1;
658            if (n % 100 == 0) {
659                emb.push_back(n-1);

```

```

657         std::cout << '\n';
658         std::cout << "Terminou-se o passo de
        tempo" << n << " com Dt=" << t <<
        '\n';
659         std::cout << "Tempo da simulação total:
        " << t_sim << "\n";

660     }
661     t_fim = t_sim;
662 }
663 }
664 emb.push_back(n-1);
665 embebicao = false;
666
667 std::cout << "Resolvendo o campo da pressao...\n";
668 parallelPressureSolver(Workload(s.size(), std::thread::
        hardware_concurrency()));
669 std::cout << "Simulacao completa!\n";
670
671 for(auto& pos:dre){
672     Plot_s.push_back(s[pos]);
673     Plot_p.push_back(p[pos]);
674 }
675
676 for(auto& pos:emb){
677     Plot_s_emb.push_back(s[pos]);
678     Plot_p_emb.push_back(p[pos]);
679 }
680 }
681 void IMPES_Parallel::parallelPressureSolver(std::vector<int> wkload) {
682     p.resize(s.size());
683     std::vector<std::shared_ptr<std::thread>> threads;
684     for (int i = 0; i < std::thread::hardware_concurrency(); i++) {
685         threads.push_back(std::make_shared<std::thread>(&
            IMPES_Parallel::SolverPressure, this, wkload[i],
            wkload[i+1]));
686     }
687     for (auto& t : threads) {
688         t->join();
689     }
690 }
691 void IMPES_Parallel::SolverPressure(int start, int end) {
692     for (int n = start; n < end; n++) {
693         std::vector<double> a(nbl, 0.0);
694         std::vector<double> b(nbl, 0.0);
695         std::vector<double> c(nbl, 0.0);
696         std::vector<double> d(nbl, 0.0);
697         std::vector<double> coefficients;
698         for (int i = 0; i < nbl; i++) {

```

```

699             coefficients = PreEq(n, i);
700             if (i > 0) {
701                 a[i] = coefficients[0];
702             }
703             if (i < nbl - 1) {
704                 c[i] = coefficients[2];
705             }
706             b[i] = coefficients[1];
707             d[i] = coefficients[3];
708         }
709         p[n] = ThomasSolver(a, b, c, d);
710     }
711 }
712 double IMPES_Parallel::SolverSaturation1(double n) {
713     std::vector<double> s_n1(nbl);
714     std::vector<double> pc0(nbl);
715     g.clear(); g.resize(nbl);
716     for (int i = 0; i < nbl; i++) {
717         SatEq2(n, i);
718     }
719     double t = dt();
720
721     for (int i = 0; i < nbl; ++i) {
722         s_n1[i] = g[i] * t + s[n - 1][i];
723         pc0[i] = core->pc(s_n1[i]);
724     }
725     s.push_back(s_n1);
726     pc.push_back(pc0);
727     return t;
728 }
729 double IMPES_Parallel::SolverSaturation2(double n) {
730     g.clear(); g.resize(nbl);
731     //Calcular valores função G
732     for (size_t i = 0; i < num_of_threads; i++) {
733         threads[i] = std::make_shared<std::thread>(std::thread(&
734             IMPES_Parallel::SatIncrementParallel, this, std::ref(
735                 wkload[i]), std::ref(wkload[i + 1]), std::ref(n)));
736     }
737     for (auto& t : threads) {
738         t->join();
739     }
740     //Terminou de calcular função G
741     //Calcular passo de tempo depois de ter calculado G
742     //Incrementar saturação no tempo
743     double time_step = dt();
744     for (int j = 0; j < nbl; j++) {
745         s_n1[j] = g[j] * time_step + s[n - 1][j];
746         pc0[j] = core->pc(s_n1[j]);

```

```

745     }
746     s.push_back(s_n1);
747     pc.push_back(pc0);
748     return time_step;
749 }
750 void IMPES_Parallel::SatIncrementParallel(int& start, int& end, double&
    n) {
751     for (int i = start; i < end; i++) {
752         SatEq2(n, i);
753     }
754 }
755 std::vector<int> IMPES_Parallel::Workload(size_t tasks, int
    num_of_workers) {
756     int remainder = tasks % num_of_workers;
757     int equal_tasks = (tasks - remainder) / num_of_workers;
758
759     std::vector<int> tasks_for_each_thread(num_of_workers);
760     int last = 0;
761     for (int j = 0; j < tasks_for_each_thread.size(); j++) {
762         tasks_for_each_thread[j] = last + equal_tasks;
763         last += equal_tasks;
764     }
765     for (int i = 0; i < remainder; i++) {
766         tasks_for_each_thread[i] += 1;
767         for (int k = i + 1; k < tasks_for_each_thread.size(); k
            ++){
768             tasks_for_each_thread[k] += 1;
769         }
770     }
771     tasks_for_each_thread.emplace(tasks_for_each_thread.begin(), 0);
772     return tasks_for_each_thread;
773 }

```

Apresenta-se na listagem 5.5 o arquivo com código da classe RCore.

Listing 5.5: Arquivo de cabeçalho da classe RCore.

```

774 #pragma once
775 #include <string>
776 #include <vector>
777 #include <exception>
778 #include <iostream>
779 #include <fstream>
780 #include <math.h>
781 #include "CGnuplot.h"
782
783 struct coreArgs {
784     double porosidade;
785     double diametro;
786     double comprimento;

```

```

787     double permeabilidade;
788     double exp_krw;
789     double exp_kro;
790     double exp_pc;
791     double krwMax;
792     double kroMax;
793     double pcMax;
794     double sor;
795     double swi;
796     double pct;
797 };
798
799 class RCore {
800 protected:
801     double a;
802     double b;
803     double c;
804     double porosity;
805     double diameter;
806     double length;
807     double permeability;
808     double krwMAX;
809     double kroMAX;
810     double pcMAX;
811     double swi;
812     double sor;
813     double Pct;
814     static const double pi;
815     std::string wettability;
816
817 public:
818     RCore(coreArgs cr, std::string _w= "W");
819     inline void setC(double _c) { c = _c; }
820     double volPoroso();
821     double Area();
822     double krw(double s);
823     double kro(double s);
824     double lambda_w(double s, double uw);
825     double lambda_o(double s, double uo);
826     double lambda_t(double s, double uw, double uo);
827     double pc(double s);
828     double fw(double s, double uw, double uo);
829
830     void PlotKr();
831     void PlotPc();
832
833     friend class IMPES;
834     friend class IMPES_Parallel;

```

```

835         friend class IMPES_ParallelCV;
836 };

```

Apresenta-se na listagem 5.6 o arquivo de implementação da classe RCore.

Listing 5.6: Arquivo de implementação da classe RCore.

```

838 #include "RCore.h"
839
840 const double RCore::pi = 3.14159265358979323846;
841
842 RCore::RCore(coreArgs cr, std::string _w) : porosity(cr.porosidade),
843     diameter(cr.diametro), length(cr.comprimento), permeability(cr.
        permeabilidade), wettability(_w), swi(cr.swi), sor(cr.sor), a
        (cr.exp_krw), b(cr.exp_kro),
844     c(cr.exp_pc), krwMAX(cr.krwMax), kroMAX(cr.kroMax), pcMAX(cr.
        pcMax), Pct(cr.pct) {
845 }
846 double RCore::volPoroso() { return 0.25 * RCore::pi * length * diameter
    * diameter * porosity; }
847 double RCore::Area() { return 0.25 * RCore::pi * pow(diameter, 2); }
848 double RCore::krw(double s) {
849     double r;
850     if (s >= 1 - sor) { r = krwMAX; }
851     else if (s <= swi) { r = 0; }
852     else {
853         r = krwMAX * pow(((s - swi) / (1 - sor - swi)), a);
854     }
855     return r;
856 }
857 double RCore::kro(double s) {
858     double r;
859     if (s >= 1 - sor) { r = 0; }
860     else if (s <= swi) { r = kroMAX; }
861     else {
862         r = kroMAX * pow(((1 - sor - s) / (1 - sor - swi)), a);
863     }
864     return r;
865 }
866 double RCore::lambda_w(double s, double uw) { return krw(s) / uw; }
867 double RCore::lambda_o(double s, double uo) { return kro(s) / uo; }
868 double RCore::lambda_t(double s, double uw, double uo) { return lambda_w
    (s, uw) + lambda_o(s, uo); }
869 double RCore::pc(double s) {
870     double r;
871     if (s >= 1 - sor) { r = Pct; }
872     else {
873         r = pcMAX * pow(((1 - sor - s) / (1 - sor - swi)), c) +
            Pct;
874     }

```

```

875         return r;
876 }
877 double RCore::fw(double s, double uw, double uo) { return lambda_w(s, uw
    ) / lambda_t(s, uw, uo); }
878 void RCore::PlotKr() {
879     try {
880         CGnuplot plot;
881         std::ofstream tmpfile("kr.txt");
882         std::vector<double> s;
883         for (double k = swi; k < 1 - sor; k += 0.01) {
884             s.push_back(k);
885         }
886         for (int i = 0; i < s.size(); i++) {
887             tmpfile << s[i] << "    " << krw(s[i]) << "    "
                << kro(s[i]) << '\n';
888         }
889         tmpfile.close();
890         plot << "set title \"Permeabilidade Relativa\"";
891         plot << "set xlabel \"Sw\"";
892         plot << "set ylabel \"Kr\"";
893         //plot << "plot 'krw.txt' title \"krw\" with lines";
894         plot << "plot 'kr.txt' using 1:2 title \"krw\" with
            lines, 'kr.txt' using 1:3 title \"kro\" with lines";
895         //plot << "set legend \"Series1\"";
896         system("Pause");
897         system("erase kr.txt");
898     }
899     catch (std::exception& e) {
900         std::cerr << e.what() << '\n';
901     }
902 }
903 void RCore::PlotPc() {
904     try {
905         CGnuplot plot;
906         std::ofstream tmpfile("pc.txt");
907         std::vector<double> s;
908         for (double k = swi; k < 1; k += 0.01) {
909             s.push_back(k);
910         }
911         for (int i = 0; i < s.size(); i++) {
912             tmpfile << s[i] << "    " << pc(s[i]) << '\n';
913         }
914         tmpfile.close();
915         plot << "set title \"Pressao Capilar\"";
916         plot << "set xlabel \"Sw\"";
917         plot << "set ylabel \"Pc(atm)\"";
918         plot << "set yrange [0:6]";
919         //plot << "plot 'krw.txt' title \"krw\" with lines";

```



```

920         plot << "plot_pc.txt", using 1:2, title "\"pc\" with lines
          ";
921         //plot << "set legend \"Series1\"";
922         system("Pause");
923         system("erase_pc.txt");
924     }
925     catch (std::exception& e) {
926         std::cerr << e.what() << '\n';
927     }
928 }

```

Apresenta-se na listagem 5.7 o arquivo que contém o ponto de entrada do programa.

Listing 5.7: Arquivo de implementação da função main().

```

929 #include "IMPESParallel.h"
930 #include <variant>
931
932
933 typedef std::chrono::steady_clock cronometro;
934
935 void EntradaManual(coreArgs& a, IMPESArgs& b) {
936     std::cout << "Entrada_manual_selecionada!\n";
937     std::cout << "Entre com o valor da porosidade: "; std::cin >> a.
        porosidade; std::cin.get();
938     std::cout << "Entre com o valor do diametro: "; std::cin >> a.
        diametro; std::cin.get();
939     std::cout << "Entre com o valor do comprimento: "; std::cin >> a.
        comprimento; std::cin.get();
940     std::cout << "Entre com o valor da permeabilidade: "; std::cin
        >> a.permeabilidade; std::cin.get();
941     std::cout << "Entre com o valor do expoente para a lei de perm.
        relativa a agua: "; std::cin >> a.exp_krw; std::cin.get();
942     std::cout << "Entre com o valor do expoente para a lei de perm.
        relativa ao oleo: "; std::cin >> a.exp_kro; std::cin.get();
943     std::cout << "Entre com o valor do expoente para a lei de
        pressao capilar: "; std::cin >> a.exp_pc; std::cin.get();
944     std::cout << "Entre com o valor mÃximo de permeabilidade
        relativa a agua: "; std::cin >> a.krwMax; std::cin.get();
945     std::cout << "Entre com o valor mÃximo de permeabilidade
        relativa ao oleo: "; std::cin >> a.kroMax; std::cin.get();
946     std::cout << "Entre com o valor mÃximo de pressÃo capilar: ";
        std::cin >> a.pcMax; std::cin.get();
947     std::cout << "Entre com o valor da saturacao residual de oleo: ";
        ; std::cin >> a.sor; std::cin.get();
948     std::cout << "Entre com o valor da saturacao irredutÃvel de
        agua: "; std::cin >> a.swi; std::cin.get();
949     std::cout << "Entre com o valor da pressao capilar de threshold:
        "; std::cin >> a.pct; std::cin.get();
950

```

```

951     std::cout << "Entre com o valor da pressao atmosferica: "; std::
        cin >> b.Patm; std::cin.get();
952     std::cout << "Entre com o valor da vazao de injeção do
        experimento: "; std::cin >> b.vazao_inj; std::cin.get();
953     std::cout << "Entre com o valor da viscosidade do oleo: "; std::
        cin >> b.uo; std::cin.get();
954     std::cout << "Entre com o valor da viscosidade da agua: "; std::
        cin >> b.uw; std::cin.get();
955     std::cout << "Entre com o valor da variacao maxima de saturacao
        em 1 passo de tempo: "; std::cin >> b.DsMax; std::cin.get();
956     std::cout << "Entre com o valor dos volumes porosos de oleo a
        ser injetado na drenagem: "; std::cin >> b.vpi1; std::cin.get
        ();
957     std::cout << "Entre com o valor dos volumes porosos de agua a
        ser injetado na embebecao: "; std::cin >> b.vpi2; std::cin.
        get();
958     std::cout << "Entre com o valor da saturacao inicial de agua no
        meio poroso: "; std::cin >> b.si; std::cin.get();
959 }
960
961 void EntradaDisco(coreArgs& a, IMPESArgs& b) {
962     std::cout << "Iniciando leitura\n";
963     std::ifstream fin("argsDarcy.txt");
964     if (fin.good()) {
965         for (int i = 0; i < 3; i++) {
966             fin.ignore(5000, '\n');
967         }
968         fin.ignore(17); fin >> a.porosidade; fin.ignore(5000, '\n');
969         fin.ignore(17); fin >> a.diametro; fin.ignore(5000, '\n');
970         fin.ignore(17, '\n'); fin >> a.comprimento; fin.ignore
            (5000, '\n');
971         fin.ignore(17, '\n'); fin >> a.permeabilidade; fin.
            ignore(5000, '\n');
972         fin.ignore(17, '\n'); fin >> a.exp_krw; fin.ignore(5000,
            '\n');
973         fin.ignore(17, '\n'); fin >> a.exp_kro; fin.ignore(5000,
            '\n');
974         fin.ignore(17, '\n'); fin >> a.exp_pc; fin.ignore(5000,
            '\n');
975         fin.ignore(17, '\n'); fin >> a.krwMax; fin.ignore(5000,
            '\n');
976         fin.ignore(17, '\n'); fin >> a.kroMax; fin.ignore(5000,
            '\n');
977         fin.ignore(17, '\n'); fin >> a.pcMax; fin.ignore(5000, '\n');
978         fin.ignore(17, '\n'); fin >> a.sor; fin.ignore(5000, '\n

```

```

        ');
979         fin.ignore(17, '\n'); fin >> a.swi; fin.ignore(5000, '\n
        ');
980         fin.ignore(17, '\n'); fin >> a.pct; fin.ignore(5000, '\n
        ');
981         fin.ignore(17, '\n'); fin >> b.Patm; fin.ignore(5000, '\n
        n');
982         fin.ignore(17, '\n'); fin >> b.vazao_inj; fin.ignore
        (5000, '\n');
983         fin.ignore(17, '\n'); fin >> b.uo; fin.ignore(5000, '\n'
        );
984         fin.ignore(17, '\n'); fin >> b.uw; fin.ignore(5000, '\n'
        );
985         fin.ignore(17, '\n'); fin >> b.DsMax; fin.ignore(5000, '
        \n');
986         fin.ignore(17, '\n'); fin >> b.vpi1; fin.ignore(5000, '\n
        ');
987         fin.ignore(17, '\n'); fin >> b.vpi2; fin.ignore(5000, '\n
        ');
988         fin.ignore(17, '\n'); fin >> b.si; fin.ignore(5000, '\n'
        );
989         fin.close();
990     }
991     else {
992         std::cout << "Arquivo de argumentos nao encontrado!\n";
993         exit(5);
994     }
995
996 }
997
998 void Teste(coreArgs argsCORE, IMPESArgs argsIMPES) {
999     std::cout << "Press enter to initialize performance tests:";
        std::cin.get(); std::cout << '\n';
1000    std::cout << "#####_INITIALIZING_PERFORMANCE_
        TESTS_#####\n";
1001
1002    std::shared_ptr<RCore> tcore1 = std::make_shared<RCore>(argsCORE
        );
1003    std::shared_ptr<RCore> tcore2 = std::make_shared<RCore>(argsCORE
        );
1004    std::shared_ptr<RCore> tcore3 = std::make_shared<RCore>(argsCORE
        );
1005
1006    std::cout << "
        #####\n";
1007    std::cout << "Numero de blocos:" << argsIMPES.nbl << '\n';
1008    std::cout << "

```

```

#####\
n";
1009
1010     IMPES SR1(tcCore1, argsIMPES);
1011     IMPES_Parallel PR1(tcCore2, argsIMPES, true);
1012     IMPES_Parallel PR2(tcCore3, argsIMPES, false);
1013
1014     SR1.Grid();
1015     PR1.Grid();
1016     PR2.Grid();
1017
1018     auto start = cronometro::now();
1019     SR1.SolverManifold();
1020     auto SR1end = cronometro::now();
1021     auto PR1start = cronometro::now();
1022     PR1.SolverManifold();
1023     auto PR1end = cronometro::now();
1024     auto PR2start = cronometro::now();
1025     PR2.SolverManifold();
1026     auto PR2end = cronometro::now();
1027
1028     std::cout << "
        #####
        ";
1029     std::cout << "Tempo_simulacao_serial_SR1:" << std::chrono::
        duration_cast<std::chrono::milliseconds>(SR1end - start).
        count() << '\n';
1030     std::cout << "Tempo_simulacao_paralela_PR1:" << std::chrono::
        duration_cast<std::chrono::milliseconds>(PR1end - PR1start).
        count() << '\n';
1031     std::cout << "Tempo_simulacao_paralela_PR2:" << std::chrono::
        duration_cast<std::chrono::milliseconds>(PR2end - PR2start).
        count() << '\n';
1032     std::cout << "
        #####
        ";
1033 }
1034
1035 int main()
1036 {
1037     std::cout << "#####\n";
1038     std::cout << "Simulador_IMPES" << '\n';
1039     std::cout << "Simulacao_de_Reservatorios_-LENEP/UENF\n";
1040     std::cout << "#####\n";
1041     std::cout << "Selecione_o_modulo_de_entrada_de_dados:\n";
1042     std::cout << "1_-Manual\n";
1043     std::cout << "2_-Por_arquivo_de_disco\n";
1044     std::cout << "Opcao:"; int op; std::cin >> op; std::cin.get();

```

```

1045     std::cout << "#####\n";
1046
1047     IMPESArgs argsIMPES;
1048     coreArgs argsCORE;
1049
1050     switch (op) {
1051     case 1: EntradaManual(argsCORE, argsIMPES); break;
1052     case 2: EntradaDisco(argsCORE, argsIMPES); break;
1053     default: EntradaDisco(argsCORE, argsIMPES); break;
1054     }
1055
1056     std::cout << "Entre com o número de blocos: "; std::cin >>
        argsIMPES.nbl; std::cin.get(); std::cout << '\n';
1057
1058     std::cout << "Selecione o modo de simulacao do experimento:\n";
1059     std::cout << "1- Serial\n";
1060     std::cout << "2- Paralela para poucos blocos\n";
1061     std::cout << "3- Paralela para muitos blocos\n";
1062     std::cout << "4- Teste\n";
1063     std::cout << "Opcao: "; std::cin >> op; std::cin.get();
1064     std::cout << "#####\n";
1065
1066     std::shared_ptr<RCore> tcore = std::make_shared<RCore>(argsCORE)
        ;
1067     IMPES* sim = nullptr;
1068     switch (op) {
1069     case 1: {
1070         sim = new IMPES(tcore, argsIMPES);
1071         break;
1072     }
1073     case 2: {
1074         sim = new IMPES_Parallel(tcore, argsIMPES, true)
            ;
1075         break;
1076     };
1077     case 3: {
1078         sim = new IMPES_Parallel(tcore, argsIMPES, false)
            );
1079         break;
1080     }
1081     case 4: {
1082         Teste(argsCORE, argsIMPES);
1083         break;
1084     }
1085     default: {
1086         sim = new IMPES_Parallel(tcore, argsIMPES, true)
            ;
1087         break;

```

```
1088         }
1089     }
1090     if (sim != nullptr) {
1091         sim->Grid();
1092         sim->SolverManifold();
1093         sim->PlotSat();
1094         sim->PlotPre();
1095     }
1096     delete sim;
1097 }
```

Capítulo 6

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).

6.1 Teste 1: Simulação de embebição e drenagem

Tomou-se para realizar a simulação de teste o seguinte experimento:

Para um plug de 10 cm de comprimento e 1.5 polegadas de diâmetro, de meio poroso molhável à água com permeabilidade de 300 mD e porosidade de 20%, injetou-se óleo a uma viscosidade de 10 cP a 3mL/min até que 10 volumes porosos fossem injetados. Este foi o processo de drenagem. A seguir, injetou-se por mais 15 volumes porosos à mesma vazão, água de viscosidade de 1 cP. Ao término deste, termina-se o processo de embebição, e consequentemente o experimento. Para realizar a simulação, foi preciso converter os dados de entrada em um sistema de unidades consistente, esta conversão se encontra na tabela 6.1:

	Sistema Darcy	Sistema Internacional
ϕ	0.2	0.2
q_{inj}	0.05 cm ³ /s	0.00000005 m ³ /s
d	3.81 cm	0.0381 m
L	10 cm	0.1 m
K	0.3 D	3.703E-14 m ²
μ_w	1 cP	0.001 Pa.s
μ_o	10 cP	0.01 Pa.s

Tabela 6.1: Conversão dos dados de entrada em um sistema de unidades consistente

Viu-se na seção 2, que as curvas de permeabilidade relativa e pressão capilar são leis de potência que precisam de expoentes e constantes adequadas para uma boa expressão analítica. Para as curvas de pressão capilar, tomou-se por base as curvas contidas nas

figuras 2.42 e 2.47 do livro Engenharia de Reservatórios de Petróleo (Rosa et al., 2011), de forma que, a expressão da pressão capilar ficou:

$$P_c(S_w) = \frac{P_{c_{Sw=Swi}}(1 - S_w - S_{or})^4}{(1 - S_{or} - S_{wi})^4} + P_{c_{threshold}}$$

As constantes: $P_{c_{Sw=Swi}}$ e $P_{c_{threshold}}$ valem: 4.08 e 1.5 atm, respectivamente; ou, 413685 e 151988 Pa, em conversão direta.

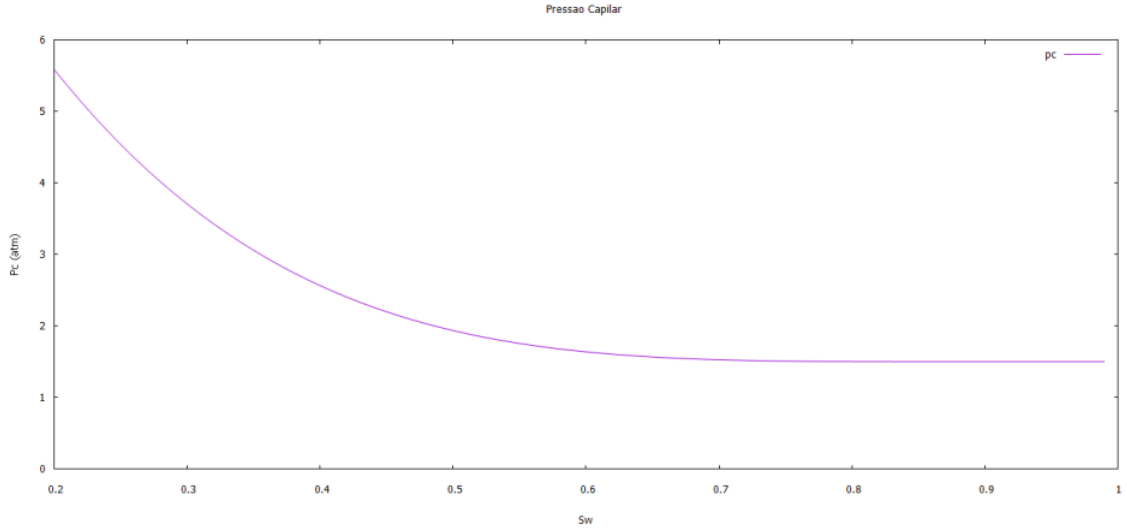


Figura 6.1: Curva de pressão capilar gerada pelo software

Semelhantemente, as curvas de permeabilidade relativa foram:

$$K_{rw}(S_w) = \frac{0.4(S_w - S_{wi})^3}{(1 - S_{or} - S_{wi})^3}$$

$$K_{ro}(S_w) = \frac{0.8(1 - S_w - S_{or})^3}{(1 - S_{or} - S_{wi})^3}$$

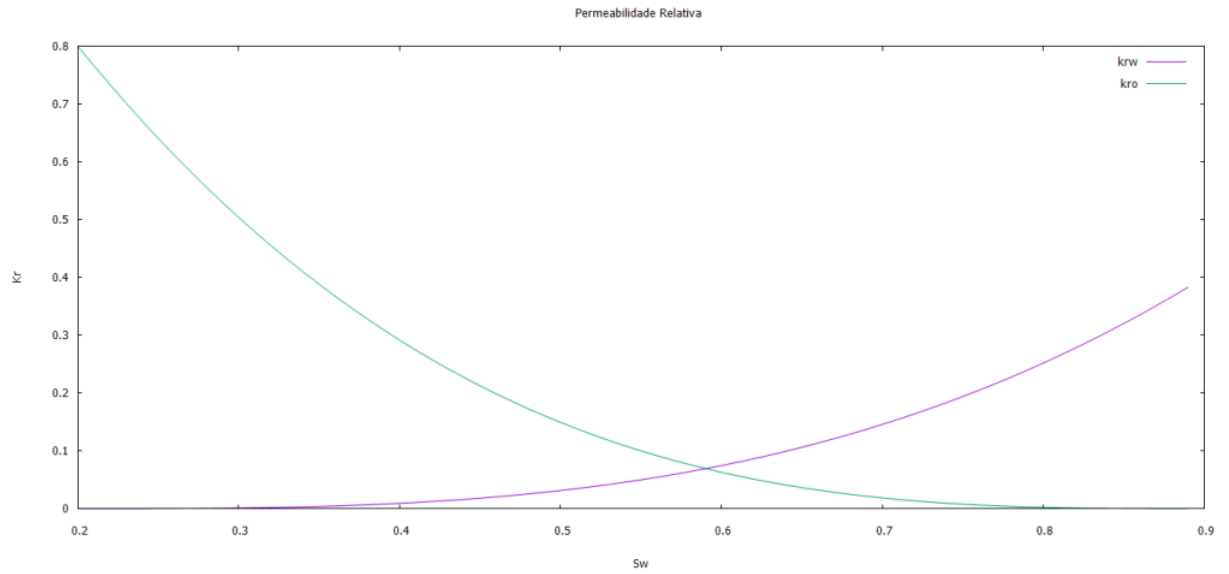


Figura 6.2: Curva de permeabilidade relativa gerada pelo software

Para uma malha de 25 blocos, a distribuição de saturação (em azul, água e, em preto, óleo) no meio poroso foi a seguinte para o processo de drenagem:

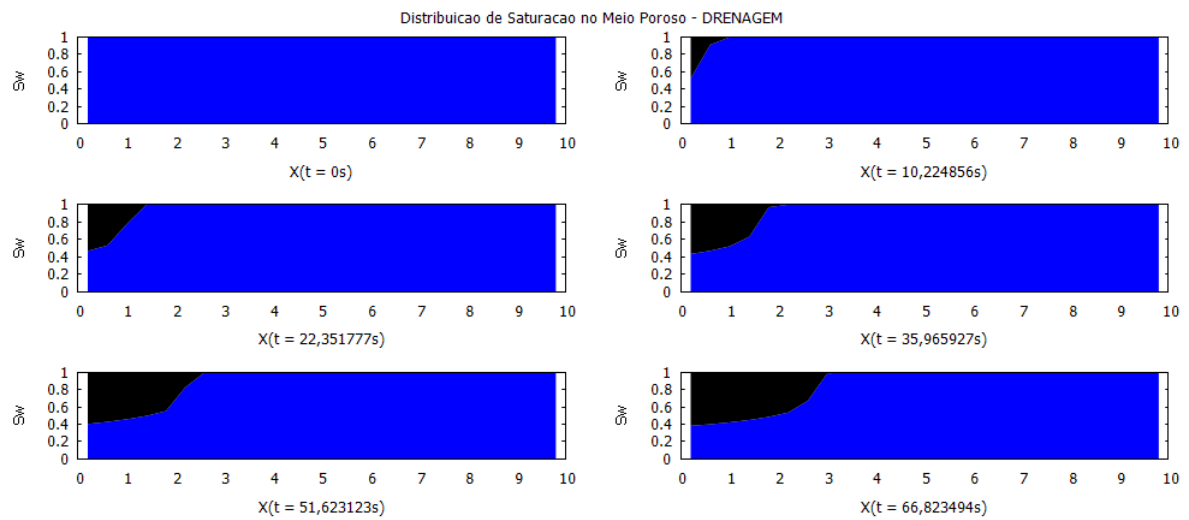


Figura 6.3: Distribuição de saturação no meio poroso - DRENAGEM - parte 1

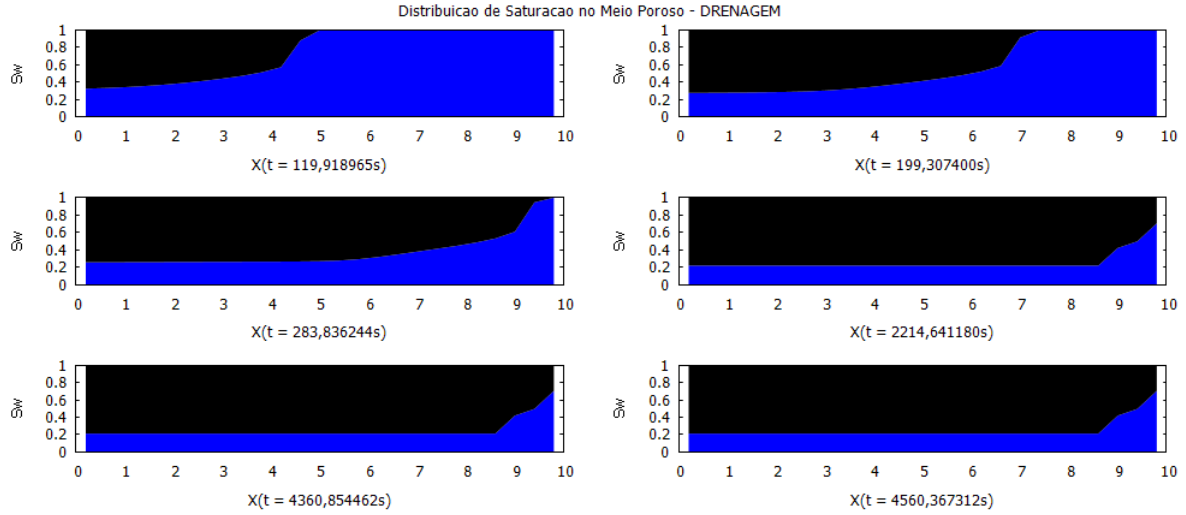


Figura 6.4: Distribuição de saturação no meio poroso - DRENAGEM - parte 2

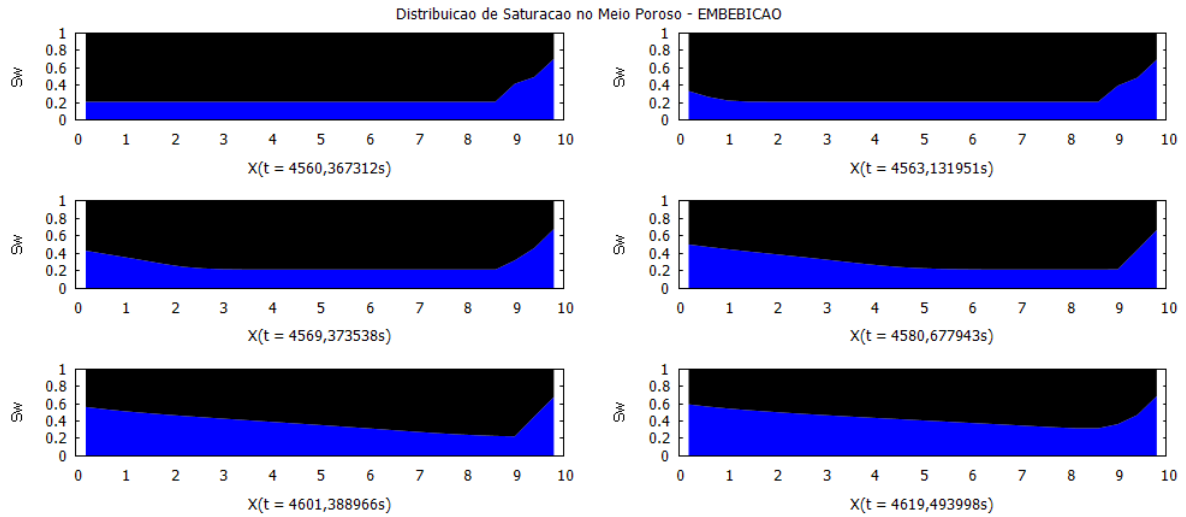


Figura 6.5: Distribuição de saturação no meio poroso - EMBEBIÇÃO - parte 1

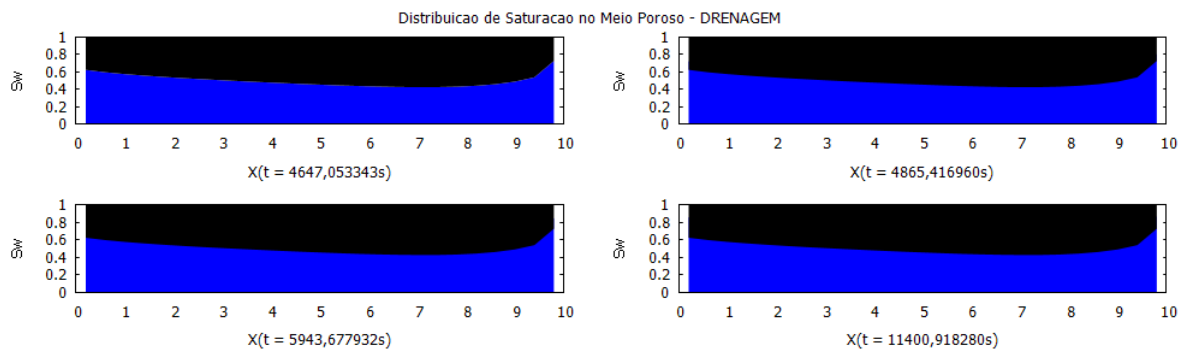


Figura 6.6: Distribuição de saturação no meio poroso - EMBEBIÇÃO - parte 2

Uma outra forma de analisar estes resultados é através dos seguintes gráficos:

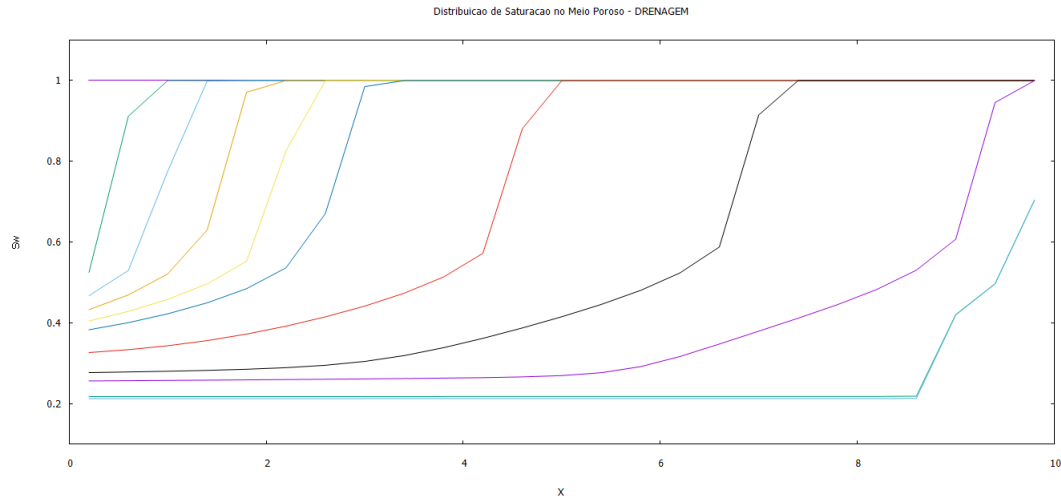


Figura 6.7: Distribuição de saturação no meio poroso - DRENAGEM - gráfico cartesiano

É possível perceber a frente de água se movendo pelo meio poroso durante o processo da drenagem através dos diversos passos de tempo. No processo de embebição, é possível ver que a saturação de água pelo meio poroso começa a subir novamente, porém, há um acúmulo de água na borda direita do testemunho, isso se dá pelo capillary end effect (que também distorce a forma da frente água da embebição), pois, somente uma fase vai fluir através da fronteira direita por vez, isto é, primeiramente o óleo fluirá até que este atinja a sua saturação residual e, a partir daí, a água acumulada será produzida.

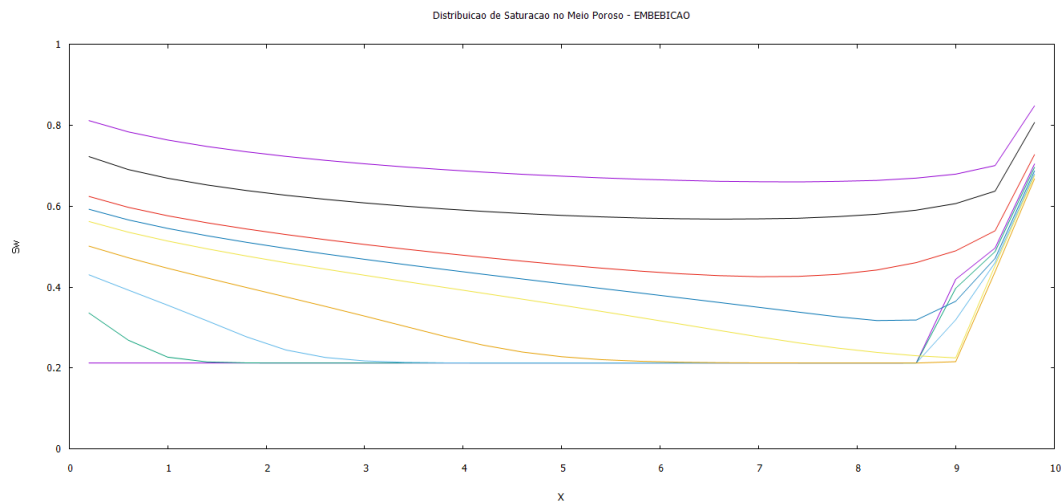


Figura 6.8: Distribuição de saturação no meio poroso - EMBEBIÇÃO - gráfico cartesiano

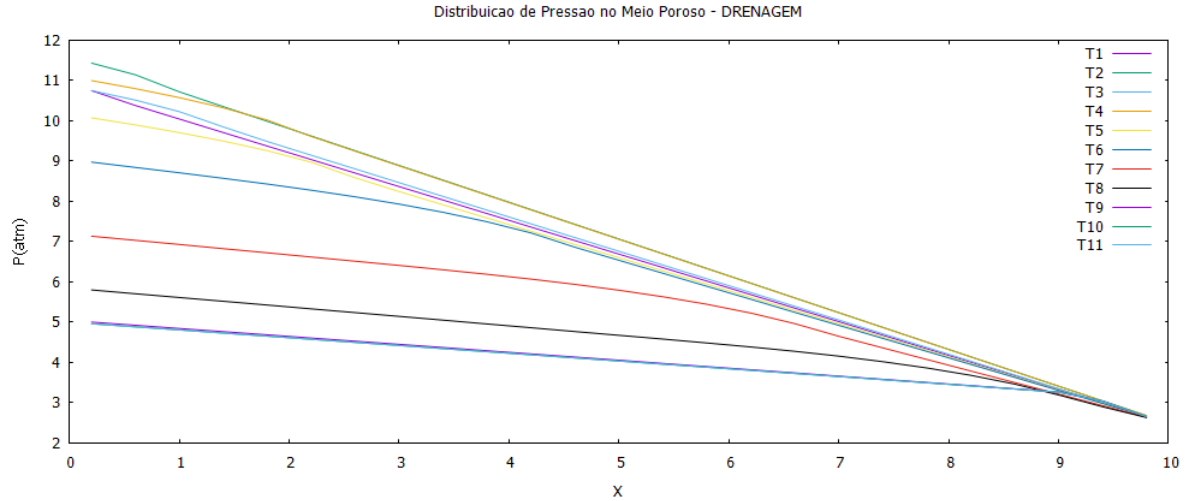


Figura 6.9: Distribuição de pressão no meio poroso - DRENAGEM - gráfico cartesiano

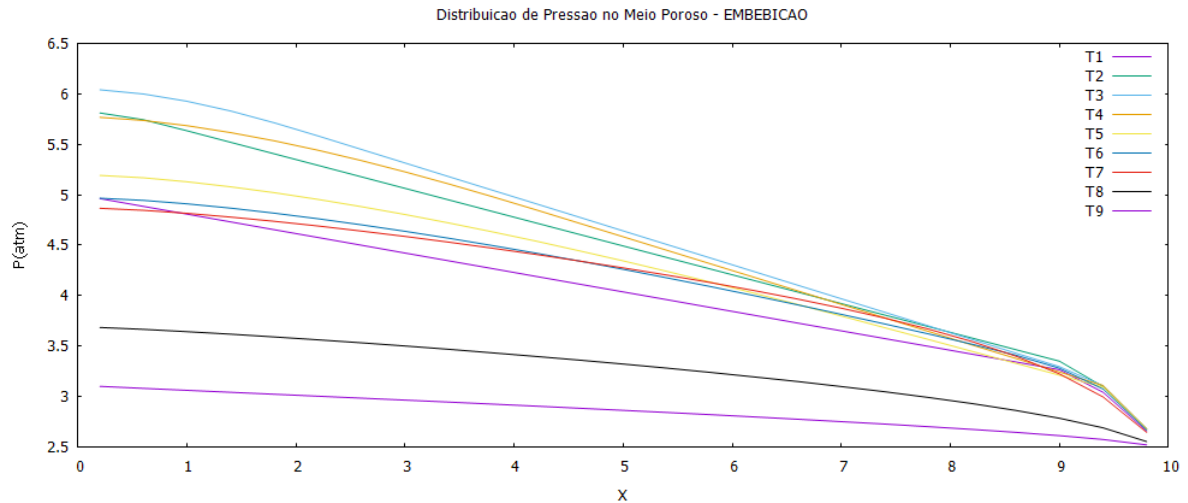


Figura 6.10: Distribuição de pressão no meio poroso - EMBEBIÇÃO - gráfico cartesiano

É notório que o comportamento dos dois campos (drenagem e embebição) são semelhantes, porém, a pressão de óleo na drenagem é maior que durante a embebição justamente pela sua maior saturação no meio poroso. E a pressão na fronteira esquerda sendo a maior do campo e a da fronteira direita, menor, é condizente com as condições físicas do problema (injeção à montante e produção em pressão atmosférica à jusante).

6.2 Teste 2: Comparação entre a simulação serial e as diversas simulações paralelas

Neste teste, executou-se várias comparações a fim de determinar a eficiência da paralelização do algoritmo contido neste projeto. Sabe-se que grande parte do tempo computacional em simulações de reservatórios é gasto na resolução de sistemas de equações

(lineares ou não-lineares), para o método IMPES, o campo de saturação é obtido explicitamente enquanto que o campo de pressão é obtido implicitamente através da solução de sistemas lineares. Para fins de teste, a versão serial da simulação será referida como SR1.

Assim, uma primeira versão paralela da simulação é a obtenção do campo de saturação de forma sequencial tal é como na simulação serial, porém o campo de pressões, ao final da resolução do campo de saturação, é obtido de forma paralela - distribuindo cada passo de tempo a um dos processadores. Por exemplo, se na simulação ocorresse 15000 passos de tempo, o campo de saturação seria obtido em uma thread; já para o campo de pressões, os 15000 passos seriam divididos entre a quantidade disponível de processadores na máquina. Para fins de teste, esta paralelização será referida como PR1.

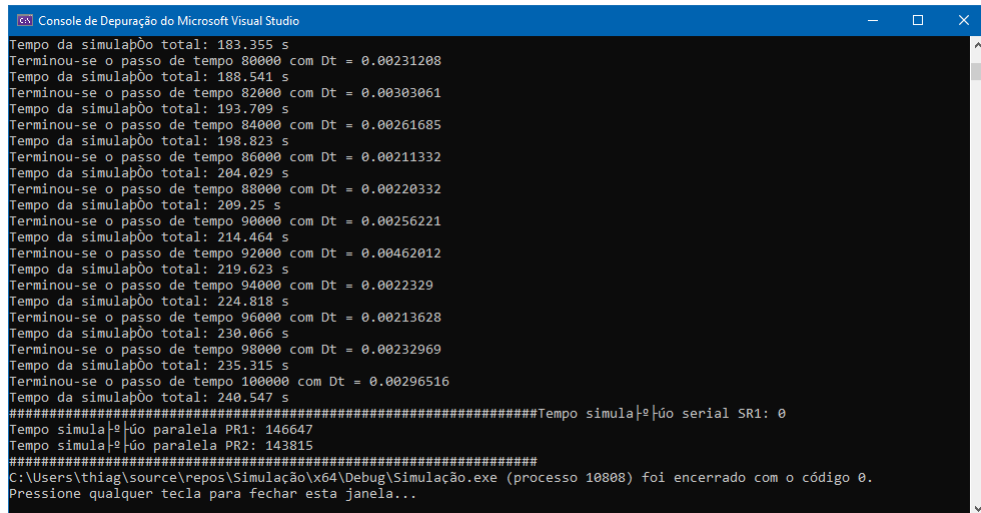
Por fim, a última maneira de paralelização do algoritmo prevista para este projeto, seria calcular o campo de saturação de maneira paralela no espaço e sequencial no tempo, e após o cálculo de todo o campo de saturação, o campo de pressão seria obtido de maneira semelhante. Para fins de teste, esta paralelização será referida como PR2.

Através da biblioteca *chrono* marcou-se o tempo gasto para as simulações SR1, PR1, PR2 para 10, 25, 50, 75 e 100 blocos. A seguir, os resultados:

Simulação / Número de blocos	10	25	50	75	100
SR1	285	1044	4027	13361	20954
PR1	174	605	2316	7825	12097
PR2	4856	9851	20555	48212	59000

Tabela 6.2: Tempos de simulação IMPES - Serial vs Paralelas (em milisegundos)

Percebe-se que PR2 é uma paralelização mais ineficiente que PR1 já que seus tempos de execução são maiores que PR1 e até maiores que SR1 para 10 blocos. Após testes com o *profiler* do Microsoft Visual Studio 2019, percebeu-se que esta ineficiência se deve à múltiplas chamadas à função *std::invoke*, isto é, o *overhead* de se implementar múltiplas threads para o cálculo explícito da saturação paralelizado no espaço é maior que seu benefício de performance. Porém, para um número maior de blocos, PR2 pode ser mais eficiente que PR1, já que o overhead não interferirá de maneira tão expressiva nos resultados. Para checar a veracidade desta hipótese, foi realizado um teste com 400 blocos para 100000 passos de tempo, a fim de ver se PR2 seria mais rápido que PR1 para o cálculo do campo de saturação. Os resultados se encontram na figura 6.11:



```

Console de Depuração do Microsoft Visual Studio
Tempo da simulação total: 183.355 s
Terminou-se o passo de tempo 80000 com Dt = 0.00231208
Tempo da simulação total: 188.541 s
Terminou-se o passo de tempo 82000 com Dt = 0.00303061
Tempo da simulação total: 193.709 s
Terminou-se o passo de tempo 84000 com Dt = 0.00261685
Tempo da simulação total: 198.823 s
Terminou-se o passo de tempo 86000 com Dt = 0.00211332
Tempo da simulação total: 204.029 s
Terminou-se o passo de tempo 88000 com Dt = 0.00220332
Tempo da simulação total: 209.25 s
Terminou-se o passo de tempo 90000 com Dt = 0.00256221
Tempo da simulação total: 214.464 s
Terminou-se o passo de tempo 92000 com Dt = 0.00462012
Tempo da simulação total: 219.623 s
Terminou-se o passo de tempo 94000 com Dt = 0.0022329
Tempo da simulação total: 224.818 s
Terminou-se o passo de tempo 96000 com Dt = 0.00213628
Tempo da simulação total: 230.066 s
Terminou-se o passo de tempo 98000 com Dt = 0.00232969
Tempo da simulação total: 235.315 s
Terminou-se o passo de tempo 100000 com Dt = 0.00296516
Tempo da simulação total: 240.547 s
#####Tempo simulação serial SR1: 0
Tempo simulação paralela PR1: 146647
Tempo simulação paralela PR2: 143815
#####
C:\Users\thiag\source\repos\Simulação\x64\Debug\Simulação.exe (processo 10808) foi encerrado com o código 0.
Pressione qualquer tecla para fechar esta janela...

```

Figura 6.11: Simulação do campo de saturação para 400 blocos e 100000 passos de tempo - PR1 vs. PR2

Como pode se ver, PR2 é uma paralelização ineficiente para um número menor de blocos (< 400), porém para números maiores de blocos, PR2 é mais eficiente que PR1.

Índice Remissivo

A

Análise orientada a objeto, 14

AOO, 14

Associações, 23

atributos, 23

C

Concepção, 2

Controle, 21

D

Diagrama de componentes, 23

Diagrama de execução, 24

Diagrama de máquina de estado, 17

Diagrama de sequência, 16

E

Efeitos do projeto nas associações, 23

Efeitos do projeto nas heranças, 23

Efeitos do projeto nos métodos, 23

Elaboração, 5

especificação, 2

estado, 17

Eventos, 16

H

Heranças, 23

heranças, 23

I

Implementação, 26

M

Mensagens, 16

métodos, 23

modelo, 23

O

otimizações, 23

P

Plataformas, 22

POO, 22

Projeto do sistema, 20

Projeto orientado a objeto, 22

Protocolos, 20

R

Recursos, 21