

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

PROJETO DE ENGENHARIA
DESENVOLVIMENTO DO SOFTWARE
SIMULAÇÃO DO EFEITO SISMOELÉTRICO
TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA

Versão 1:
IZIS ROSA PINHEIRO
PROF. ANDRÉ DUARTE BUENO
PROF. VIATCHESLAV IVANOVICH PRIIMENKO

MACAÉ - RJ
Julho - 2025

Sumário

1	Introdução	1
1.1	Escopo do problema	1
1.2	Objetivos	2
2	Especificação	6
2.1	Nome do Sistema/Produto	6
2.2	Especificação	6
2.3	Requisitos	7
2.3.1	Requisitos funcionais	7
2.3.2	Requisitos não funcionais	7
2.4	Casos de Uso	7
2.4.1	Diagrama de caso de uso geral	8
2.4.2	Diagrama de caso de uso específico	8
3	Elaboração	10
3.1	Análise de domínio	10
3.2	Formulação teórica	11
3.3	Equações de Pride	11
3.4	Sistema de Pride no formato de Ursin	12
3.5	Identificação de pacotes – assuntos	19
3.6	Diagrama de pacotes – assuntos	19
4	AOO – Análise Orientada a Objeto	21
4.1	Diagramas de classes	21
4.1.1	Dicionário de classes	21
4.2	Diagrama de seqüência – eventos e mensagens	22
4.2.1	Diagrama de sequência geral	22
4.3	Diagrama de comunicação – colaboração	23
4.4	Diagrama de estado	24
4.5	Diagrama de atividades	24
5	Projeto	26
5.1	Projeto do Sistema	26

5.2	Projeto Orientado a Objeto – POO	27
5.3	Diagrama de Componentes	28
5.4	Diagrama de Implantação	29
6	Implementação	31
6.1	Código fonte	31
7	Teste	64
7.1	Teste 1	65
7.2	Teste 2	66
8	Documentação	69
8.1	Documentação do usuário	69
8.1.1	Como rodar o software	69
8.2	Documentação para desenvolvedor	69
8.2.1	Dependências	69
	Referências Bibliográficas	71

Capítulo 1

Introdução

No presente projeto de engenharia desenvolve-se o software SIMULADOR DO EFEITO SISMOE-LÉTRICO, um código em linguagem orientada a objeto que tem como principal objetivo modelar e analisar o acoplamento entre ondas sísmicas e os campos eletromagnéticos gerados pela movimentação dos fluidos em meio porosos saturados.

O simulador é projetado para integrar módulos de cálculos que englobam a solução de autovalores para sistemas acoplados, análise no domínio do tempo e da frequência, e representação gráfica dos resultados.

1.1 Escopo do problema

O efeito sismoelétrico é um fenômeno físico que consiste na geração de campos eletromagnéticos induzidos pela propagação de ondas sísmicas em meios porosos saturados por fluidos ionizados (Peng et al., 2017). Esse fenômeno representa uma ferramenta promissora para a caracterização do subsolo, oferecendo dados complementares aos métodos sísmicos convencionais, sobretudo para a identificação de propriedades hidrogeológicas e a presença de fluidos (Garambois & Dietrich, 2002; Revil & Mahardika, 2013).

No entanto, a simulação computacional precisa do efeito sismoelétrico envolve desafios significativos devido à complexidade das interações acopladas entre as ondas sísmicas e os campos eletromagnéticos, exigindo o tratamento simultâneo de equações que descrevem a mecânica dos meios porosos (Biot, 1956) e as equações de Maxwell para os campos eletromagnéticos (Maxwell, 1873; Revil & Linde, 2006). A dificuldade aumenta pela necessidade de incorporar dados multidisciplinares, como propriedades físico-químicas dos fluidos (viscosidade, condutividade elétrica), características reológicas do meio poroso e parâmetros geológicos heterogêneos (Luo et al., 2016).

Além disso, a modelagem computacional deve lidar com a variabilidade espacial e temporal dessas propriedades e garantir a convergência numérica em simulações no domínio do tempo e da frequência (Peng et al., 2017). Outro desafio importante é a implementação eficiente de algoritmos para resolução dos sistemas acoplados, como métodos baseados em autovalores e transformadas rápidas de Fourier, garantindo precisão e escalabilidade computacional (Lacroix et al., 2015).

Portanto, o problema central deste projeto consiste no desenvolvimento de um software capaz de

analisar o efeito sismoelétrico e visualização dos resultados.

1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:
 - Desenvolver um projeto matemático-numérico relacionado com a propagação das ondas em meios elásticos e condutivos, considerando os processos de acoplamento, através do efeito eletrocinético, destas ondas com o campo eletromagnético.
- Objetivos específicos:
 - Modelar física e matematicamente o problema.
 - Desenvolver algoritmos numéricos para resolver o sistema acoplado no domínio do tempo e da frequência.
 - Realizar simulações para análises paramétricas sob diferentes cenários geológicos.
 - Gerar gráficos externos a partir do software externo Gnuplot.

Lista de Figuras

2.1	Diagrama de caso de uso – Caso de uso geral	8
2.2	Diagrama de caso de uso específico – Simulação do efeito em baixas frequências	9
3.1	Diagrama de Pacotes	20
4.1	Diagrama de classes	21
4.2	Diagrama de sequência	23
4.3	Diagrama de comunicação	23
4.4	Diagrama de máquina de estado	24
4.5	Diagrama de atividades	25
5.1	Diagrama de componentes	29
5.2	Diagrama de implantação	30
7.1	Telas de execução do programa escolhendo baixas frequências	65
7.2	Velocidade da fase fluida em baixas frequências	66
7.3	Telas de execução do programa escolhendo altas frequências	67
7.4	Velocidade da fase fluida em altas frequências	68

Lista de Tabelas

2.1	Caso de uso 1	8
7.1	As propriedades petrofísicas e físicas do meio poroso e dos fluidos.	64

Listagens

6.1	Implementação da função main().	31
6.2	Arquivo de cabeçalho da classe seismicSimulator.	34
6.3	Arquivo de implementação da classe seismicSimulator.	35
6.4	Arquivo de cabeçalho da classe eigenValueSolver.	49
6.5	Arquivo de implementação da classe eigenValueSolver.	50
6.6	Arquivo de cabeçalho da classe fft.	51
6.7	Arquivo de implementação da classe fft.	52
6.8	Arquivo de cabeçalho da classe fluidProperties.	57
6.9	Arquivo de cabeçalho da classe rockProperties.	57
6.10	Arquivo de cabeçalho da classe systemParameters.	58
6.11	Arquivo de cabeçalho da classe gnuPlotter.	59
6.12	Arquivo de implementação da classe gnuPlotter.	60

Capítulo 2

Especificação

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

2.1 Nome do Sistema/Produto

Nome	SIMULAÇÃO DO EFEITO SISMOELÉTRICO
Componentes principais	O simulador do efeito sismoelétrico incluem modelagem física e matemática, execução de simulações, análise e visualização gráfica dos resultados
Missão	Modelar o comportamento das respostas sismoelétricas em diferentes cenários geológicos

2.2 Especificação

Deseja-se desenvolver um projeto de engenharia que seja capaz de simular o efeito sismoelétrico através em meios porosos saturados. Fundamentado na teoria da poroelasticidade de Biot (1956) e em princípios eletro-hidrodinâmicos, o simulador modela a interação entre as ondas sísmicas mecânicas e os campos elétricos gerados devido à movimentação dos fluidos dentro da matriz porosa. O código recebe como entrada um conjunto completo de parâmetros físicos e geométricos, incluindo propriedades da rocha (densidade, porosidade, permeabilidade, coeficiente e módulo de Biot, módulos elásticos) e propriedades dos fluidos saturantes (densidade, viscosidade, constante dielétrica, condutividade), além de parâmetros do sistema como frequência dominante da fonte, profundidade da fonte, tempo total da simulação e número de pontos para discretização temporal.

A interação do usuário com o simulador ocorre nos arquivos de código-fonte, arquivos de cabeçalho (.h) e de implementação (.cpp). Nos arquivos .h, o usuário define os parâmetros globais da simulação em systemParameters.h, configura as propriedades da rocha em rockProperties.h e insere as características

dos fluidos em `fluidProperties.h`. Já nos arquivos `.cpp`, como `seismicSimulator.cpp`, é possível ajustar o fluxo da simulação, como etapas de inicialização, cálculos no domínio da frequência, transformadas e normalização. Além disso, o usuário pode customizar a geração de gráficos em `gnuPlotter.cpp` para visualizar os resultados. Essa abordagem permite que o simulador seja adaptado a diferentes cenários geofísicos.

2.3 Requisitos

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

2.3.1 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

RF-01	O sistema deverá permitir a definição e ajuste de parâmetros da simulação por meio dos arquivos de cabeçalho (<code>arquivos.h</code>).
RF-02	O programa deve calcular a solução sismoelétrica no domínio da frequência, considerando a interação entre ondas sísmicas e o acoplamento eletrocinético dos fluidos nos poros da rocha.
RF-03	O simulador deve aplicar a FFT para transformar os dados temporais no domínio da frequência.
RF-04	Os resultados obtidos devem ser normalizados e preparados para análise, garantindo dados consistentes e interpretáveis.
RF-05	O sistema deve gerar gráficos que representem os sinais temporais simulados que deverão ser mostrados na tela.

2.3.2 Requisitos não funcionais

RNF-01	Os cálculos devem ser feitos utilizando-se transformadas numéricas.
RNF-02	O programa deverá ser multi-plataforma, podendo ser executado em <i>Windows</i> , <i>GNU/Linux</i> ou <i>Mac</i> .

2.4 Casos de Uso

A Tabela 2.1 mostra a descrição de um caso de uso.

Tabela 2.1: Caso de uso 1

Nome do caso de uso:	Simulação do Efeito Sismoelétrico
Resumo/descrição:	Modela a resposta eletrocinética de meios poroelásticos a ondas sísmicas, utilizando FFT, autovalores e plotagem com Gnuplot.
Etapas:	1. Executa o software 2. Determina o modelo de frequência 4. Gerar gráfico 5. Analisar Resultados.
Cenários alternativos:	Um cenário alternativo envolve analisar o efeito sismoelétrico na fase sólida.

2.4.1 Diagrama de caso de uso geral

O diagrama de caso de uso geral da Figura 2.1 mostra o usuário acessando os sistemas de ajuda do software, o cálculo para solução ou analisando resultados. Este diagrama de caso de uso ilustra as etapas a serem executadas pelo usuário ou sistema, ou seja, a interação do usuário com o sistema.

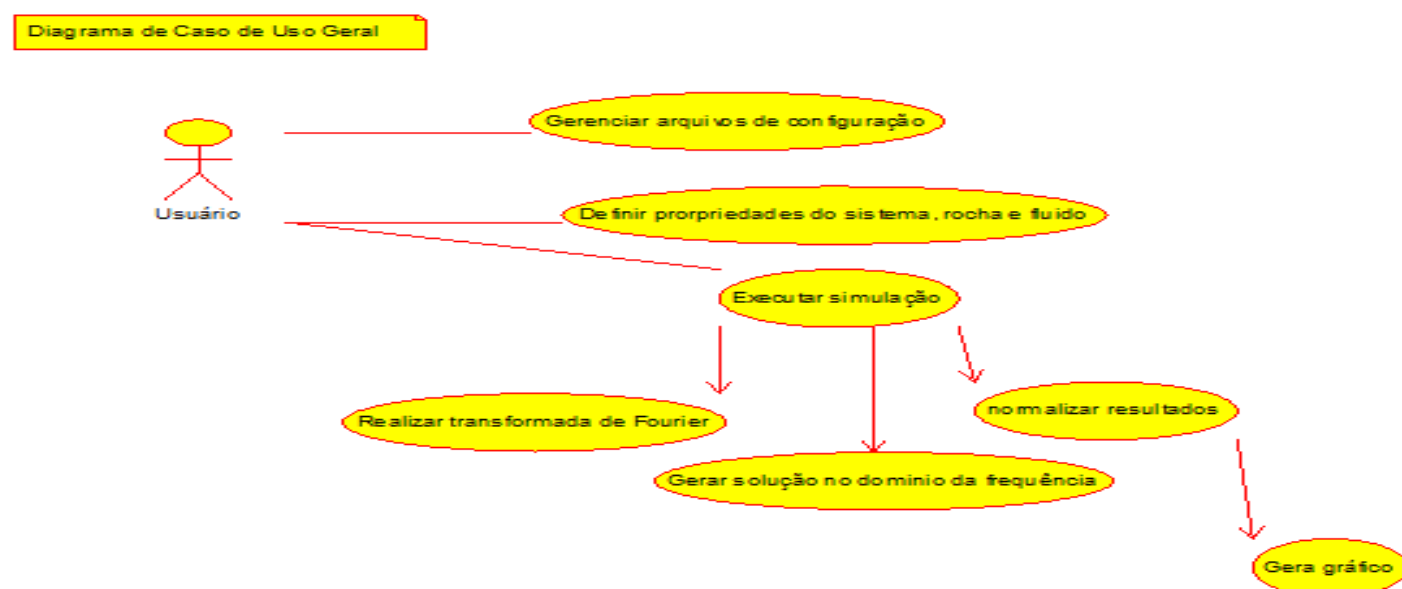


Figura 2.1: Diagrama de caso de uso – Caso de uso geral

2.4.2 Diagrama de caso de uso específico

Neste caso de uso específico, o usuário irá escolher baixas frequências e irá analisar os resultados obtidos pelo software.

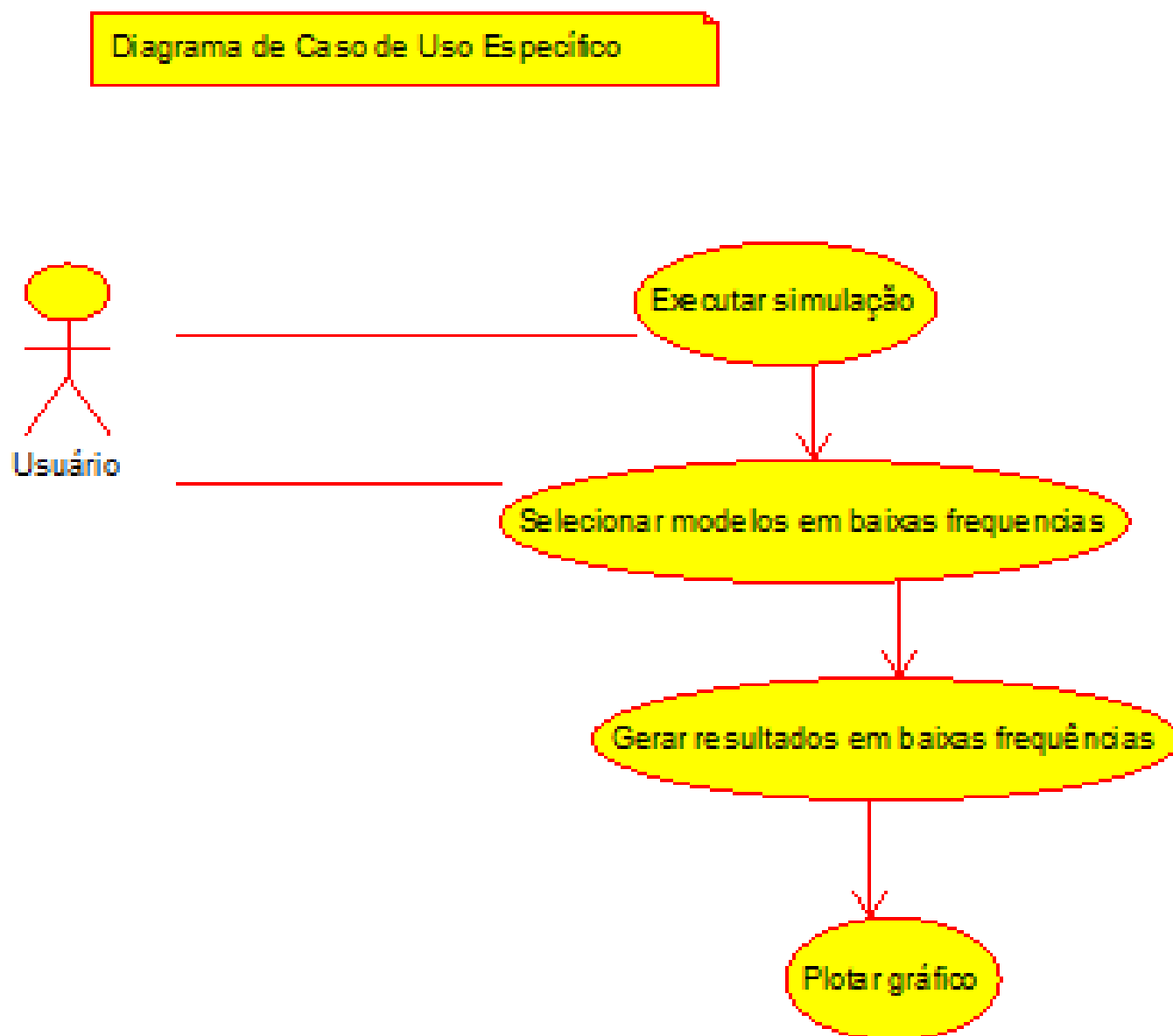


Figura 2.2: Diagrama de caso de uso específico – Simulação do efeito em baixas frequências

Capítulo 3

Elaboração

Depois da definição dos objetivos, da especificação do software e da montagem dos primeiros diagramas de caso de uso, a equipe de desenvolvimento do projeto de engenharia passa por um processo de elaboração que envolve o estudo de conceitos relacionados ao sistema a ser desenvolvido, a análise de domínio e a identificação de pacotes.

Na elaboração fazemos uma 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.

3.1 Análise de domínio

Após estudo dos requisitos/especificações do sistema, algumas entrevistas, estudos na biblioteca e disciplinas do curso foi possível identificar nosso domínio de trabalho:

- Geofísica: parte fundamental na qual sustenta este projeto. O software desenvolvido, utiliza conceitos de propagação de ondas sísmicas e a interação destas com as propriedades elétricas dos meios geológicos. Essencial para compreensão do efeito estudado, no qual integra respostas eletromagnéticas induzidas por ondas sísmicas.
- Álgebra linear e Cálculo Integral e Diferencial na resolução de equações diferenciais, sistema de matrizes, por exemplo.
- Métodos Numéricos e Computacionais: é essencial para o desenvolvimento de algoritmos eficientes para a solução de equações diferenciais, transformadas de Fourier, cálculo de autovalores e autovetores.
- Eletromagnetismo: é fundamental compreender os princípios de campos elétricos e magnéticos em meios condutores e dielétricos para compreender a geração dos potenciais elétricos que são associados ao efeito sismoelétrico. Além disso, o domínio das equações de Maxwell.
- Pacote Gráfico: usar-se-á um pacote gráfico para plotar velocidade de deslocamento das fases fluídas em altas e baixas frequências.

- Software: serão utilizadas métodos e funções já existentes para a resolução de transformadas de Fourier.

3.2 Formulação teórica

3.3 Equações de Pride

As equações de Pride que modelam o efeito eletrocinético (para baixas frequências) em meios porosos, em cada ponto $\mathbf{x} = (x, y, z) \in \mathbb{R}^3$, são:

$$\begin{aligned}
 \nabla \times \mathbf{E} &= i\omega\mu_0\mathbf{H}, \nabla \cdot \mathbf{H} = 0 \\
 \nabla \times \mathbf{H} &= (\sigma - i\epsilon\omega)\mathbf{E} + L(-\nabla p + \omega^2\rho_f\mathbf{u} + \mathbf{f}) + \mathbf{j} \\
 -\omega^2(\rho\mathbf{u} + \rho_f\mathbf{w}) &= \nabla \cdot \tau + \mathbf{F} \\
 -i\omega\mathbf{w} &= L\mathbf{E} + \frac{\kappa}{\eta}(-\nabla p + \omega^2\rho_f\mathbf{u} + \mathbf{f}) \\
 \tau &= (\lambda\nabla \cdot \mathbf{u} + C\nabla \cdot \mathbf{w})\mathbf{I} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T) \\
 -p &= C\nabla \cdot \mathbf{u} + M\nabla \cdot \mathbf{w}.
 \end{aligned} \tag{3.1}$$

Neste conjunto de equações, um sólido poroso saturado por um fluido viscoso de fase única e compressível é considerado, e também é assumido que todo o agregado é isotrópico; além disso, uma dependência de tempo de $e^{-i\omega t}$, onde ω é frequência, é assumida. As fontes nas equações de Pride são as seguintes: \mathbf{F} , a força imposta no volume material, \mathbf{f} , a força imposta no fluido contido nos poros e \mathbf{j} , a corrente elétrica aplicada externamente. As seguintes são as quantidades a serem calculadas: \mathbf{E} , o campo elétrico, \mathbf{H} , o campo magnético, \mathbf{u} , o deslocamento do sólido, \mathbf{w} , o deslocamento relativo do fluido, τ , o tensor de tensão, e p , a pressão no fluido de poros; \mathbf{I} é a matriz de identidade 3×3 . Os parâmetros materiais nas equações de Pride são os seguintes: μ , a permeabilidade magnética, ϵ , a constante dielétrica, λ e μ , os parâmetros Lamé, c e m , os módulos de Biot, $\rho = \varphi\rho_f + (1 - \varphi)\rho_s$, a densidade aparente, φ , a porosidade efetiva de meio, ρ_f , a densidade do fluido, κ , a permeabilidade, η , a viscosidade do fluido, e L , o coeficiente de acoplamento eletrocinético. Quando $L \neq 0$ temos os sistemas de Lamé e de Maxwell acoplados. Para $L = 0$, as equações se reduzem a sistemas desacoplados, com as equações de Maxwell governando o eletromagnetismo e as equações de Biot governando o movimento de fluido e de sólido em um meio poroso.

Vamos escrever

$$\bar{\sigma} = \sigma - i\epsilon\omega \tag{3.2}$$

onde $\sigma \gg \epsilon\omega$ na subsuperfície $z > 0$, de modo que σ possa ser aproximado por σ nesta região. Essa aproximação é equivalente a ignorância das correntes de deslocamento na Terra. No ar, $z < 0$, a condutividade σ é zero, e a constante dielétrica $\epsilon \neq 0$, de modo que temos $\sigma = -i\epsilon_0\omega$ no ar. Para construir um algoritmo para um código computacional rápido, restringiremos ao caso em que os parâmetros de meio são constantes por partes. Assim, assume-se que as propriedades do meio são constantes dentro de cada camada, mas mudam de forma à medida que z varia através de uma interface horizontal. Nas fronteiras das camadas, aplicamos a condição de interface da Pride, onde \mathbf{u} , p , os

componentes normais de \mathbf{w} e τ , e os componentes tangenciais de \mathbf{E} e \mathbf{H} são contínuas. Resta fornecer as condições de contorno na interface terra/ar em $z = 0$. Aplicando as condições da interface de Pride, temos que as condições de contorno são

$$z = 0 : \tau_{13} = \tau_{23} = \tau_{33} = 0, p = 0, \tilde{H}_2 = -\epsilon_0 q_0 \tilde{E}_1, \tilde{H}_1 = -q_0 \mu_0 \tilde{E}_2 \quad (3.3)$$

onde

$$\tilde{F}(k_1, k_2, z) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} e^{i(k_1 x + k_2 y)} F(x, y, z) dx dy. \quad (3.4)$$

Na Equação 3 esses relacionamentos são derivados da condição de haver apenas ondas *EM* ascendentes no ar, e q_0 é a lentidão vertical de uma onda *EM* no ar, isto é,

$$q_0 = \sqrt{\epsilon_0 \mu_0 - \gamma^2} \quad (3.5)$$

onde μ_0 é a permeabilidade magnética do ar, de modo que $\epsilon_0 \mu_0$ é o quadrado da velocidade da luz.

3.4 Sistema de Pride no formato de Ursin

Quando escrevemos o sistema das equações da interação eletrocinética no formato de Ursin, o sistema nas variáveis x_1, x_2 e x_3 é substituído por um sistema de equações diferenciais que depende apenas da variável $z = x_3$. Assim, aplica-se a transformada de Fourier bidimensional nas duas coordenadas laterais x_1, x_2 do sistema.

$$\hat{X}(k_1, k_2, z) \equiv \mathcal{F}[f(t)] = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} e^{-i(k_1 x_1 + k_2 x_2)} X(k_1, k_2, z) dx_1 dx_2, \quad (3.6)$$

enquanto a transformada inversa de \hat{X} é

$$X(k_1, k_2, z) \equiv \mathcal{F}^{-1}[\hat{X}] = \frac{1}{4\pi^2} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} e^{-i(k_1 x_1 + k_2 x_2)} \hat{X}(k_1, k_2, z) dk_1 dk_2, \quad (3.7)$$

A magnitude do número de onda horizontal é

$$k = \sqrt{k_1^2 + k_2^2}, \quad (3.8)$$

e a vagarosidade horizontal,

$$\gamma = \frac{k}{\omega}. \quad (3.9)$$

Assim,

$$\mathcal{F} \left[\frac{\partial X}{\partial x_i} \right] = i k \hat{H}(k_1, k_2, z), i = 1, 2. \quad (3.10)$$

Aplicando a transformada bidimensional de Fourier nas equações do sistema de Pride, utilizando a propriedade da derivação, obtêm-se

$$\left(ik_3 \hat{E}_3 - \frac{\partial \hat{E}}{\partial z}, \frac{\partial \hat{E}}{\partial z} - ik_1 \hat{E}_3, ik_1 \hat{E}_2 - ik_2 \hat{E}_1 \right) = i\omega\mu \left(\hat{H}_1, \hat{H}_2, \hat{H}_3 \right), \quad (3.11)$$

$$\begin{aligned} & \left(ik_3 \hat{E}_3 - \frac{\partial \hat{E}}{\partial z}, \frac{\partial \hat{E}}{\partial z} - ik_1 \hat{E}_3, ik_1 \hat{E}_2 - ik_2 \hat{E}_1 \right) = \bar{\sigma} \left(\hat{H}_1, \hat{H}_2, \hat{H}_3 \right) + \\ & + L \left(- \left(ik_1 \hat{p}, ik_2 \hat{p}, \frac{\partial \hat{p}}{\partial z} \right) + \omega^2 \rho_f (\hat{u}_1, \hat{u}_2, \hat{u}_3) + (\hat{f}_1, \hat{f}_2, \hat{f}_3) \right) + (\hat{j}_1, \hat{j}_2, \hat{j}_3), \end{aligned} \quad (3.12)$$

$$\begin{aligned} & -\omega^2 (\rho (\hat{u}_1, \hat{u}_2, \hat{u}_3) + \rho_f (\hat{w}_1, \hat{w}_2, \hat{w}_3)) = \\ & = \left(i\kappa_1 \hat{\tau}_{11} + i\kappa_2 \hat{\tau}_{21} + \frac{\partial \hat{\tau}_{31}}{\partial z}, i\kappa_1 \hat{\tau}_{12} + i\kappa_2 \hat{\tau}_{22} + \frac{\partial \hat{\tau}_{32}}{\partial z}, i\kappa_1 \hat{\tau}_{13} + i\kappa_2 \hat{\tau}_{23} + \frac{\partial \hat{\tau}_{33}}{\partial z} \right) + (\hat{F}_1, \hat{F}_2, \hat{F}_3), \end{aligned} \quad (3.13)$$

$$-i\omega (\hat{w}_1, \hat{w}_2, \hat{w}_3) = L(\hat{E}_1, \hat{E}_2, \hat{E}_3) + \frac{\kappa}{\eta} \left(- \left(ik_1 \hat{p}, ik_2 \hat{p}, \frac{\partial \hat{p}}{\partial z} \right) + \omega^2 \rho_f (\hat{u}_1, \hat{u}_2, \hat{u}_3) + (\hat{f}_1, \hat{f}_2, \hat{f}_3) \right), \quad (3.14)$$

$$\hat{\tau} = \left(\lambda \left(i\kappa_1 \hat{u}_1 + i\kappa_2 \hat{u}_2 + \frac{\partial \hat{u}_3}{\partial z} \right) + C \left(i\kappa_1 \hat{w}_1 + i\kappa_2 \hat{w}_2 + \frac{\partial \hat{w}_3}{\partial z} \right) \right) I + GN \quad (3.15)$$

onde

$$N = \begin{pmatrix} 2i\kappa_1 \hat{u}_1 & i\kappa_2 \hat{u}_1 + i\kappa_1 \hat{u}_2 & \frac{\partial \hat{u}_1}{\partial z} + i\kappa_1 \hat{u}_3 \\ i\kappa_1 \hat{u}_2 + i\kappa_2 \hat{u}_1 & 2i\kappa_2 \hat{u}_2 & \frac{\partial \hat{u}_2}{\partial z} + i\kappa_2 \hat{u}_3 \\ i\kappa_1 \hat{u}_3 + \frac{\partial \hat{u}_1}{\partial z} & i\kappa_2 \hat{u}_3 + \frac{\partial \hat{u}_2}{\partial z} & 2\frac{\partial \hat{u}_3}{\partial z} \end{pmatrix}, \quad (3.16)$$

$$-\hat{p} = \left(C \left(i\kappa_1 \hat{u}_2 + i\kappa_2 \hat{u}_1 + \frac{\partial \hat{u}_3}{\partial z} \right) + M \left(i\kappa_1 \hat{w}_1 + i\kappa_2 \hat{w}_2 + \frac{\partial \hat{w}_3}{\partial z} \right) \right). \quad (3.17)$$

Quando usamos a notação $\dot{\hat{u}}$ omitimos o termo $-i\omega$. O termos \hat{u} representa a velocidade do deslocamento absoluto da fase sólida e \hat{w} a velocidade do deslocamento relativo do fluido.

Da Equação 11, obtemos:

$$\hat{H}_3 = \frac{\kappa_1 \hat{E}_2}{\omega\mu} - \frac{\kappa_2 \hat{E}_1}{\omega\mu} \quad (3.18)$$

e da Equação 12:

$$\hat{E}_3 = \frac{1}{\bar{\sigma}} \left(i\kappa_1 \hat{H}_1 + i\kappa_2 \hat{H}_2 + L \frac{\partial \hat{p}}{\partial z} - L\omega^2 \rho_f \hat{u}_3 - L\hat{f}_1 - \hat{j}_3 \right). \quad (3.19)$$

Assim, da Equação 14:

$$\dot{\hat{w}}_1 = L\hat{E}_1 - \frac{\kappa}{\eta} ik_1 \hat{p} + \omega^2 \rho_f \frac{\kappa}{\eta} \hat{u}_1 + \frac{\kappa}{\eta} \hat{f}_1, \quad (3.20)$$

$$\dot{\hat{w}}_2 = L\hat{E}_2 - \frac{\kappa}{\eta} ik_2 \hat{p} + \omega^2 \rho_f \frac{\kappa}{\eta} \hat{u}_2 + \frac{\kappa}{\eta} \hat{f}_2, \quad (3.21)$$

$$\dot{\hat{w}}_3 = L\hat{E}_3 - \frac{\kappa}{\eta} \frac{\partial \hat{p}}{\partial z} + \omega^2 \rho_f \frac{\kappa}{\eta} \hat{u}_3 + \frac{\kappa}{\eta} \hat{f}_3. \quad (3.22)$$

Das Equações 15 e 16:

$$\hat{\tau}_{12} = -\frac{G\kappa_1}{\omega}\dot{u}_1 - \frac{G\kappa_1}{\omega}\dot{u}_2, \quad (3.23)$$

$$\hat{\tau}_{11} = -\frac{2G\kappa_1}{\omega}\dot{u}_1 - \frac{2Gi}{\omega}\frac{d\dot{u}_3}{dz} + \hat{\tau}_{33}, \quad (3.24)$$

$$\hat{\tau}_{22} = -\frac{2G\kappa_2}{\omega}\dot{u}_2 - \frac{2Gi}{\omega}\frac{d\dot{u}_3}{dz} + \hat{\tau}_{33}. \quad (3.25)$$

Explicitando a derivada em relação a z de cada uma das variáveis envolvidas no sistema de equações das transformadas e das Equações 12 e 18,

$$\frac{d\hat{H}_1}{dz} = i\omega L\rho_f\dot{u}_2 - \frac{i\kappa_1\kappa_2}{\omega\mu}\hat{E}_1 + \left(\frac{i\kappa_1^2}{\omega\mu} + \bar{\sigma}\right)\hat{E}_2 - Li\kappa_2\hat{p} + L\hat{f}_2 + \hat{j}_2, \quad (3.26)$$

$$\frac{d\hat{H}_2}{dz} = i\omega L\rho_f\dot{u}_1 - \left(\frac{i\kappa_2^2}{\omega\mu} + \bar{\sigma}\right)\hat{E}_1 + \frac{i\kappa_1\kappa_2}{\omega\mu}\hat{E}_2 - Li\kappa_1\hat{p} + L\hat{f}_1 + \hat{j}_1, \quad (3.27)$$

das Equações 14 e 19:

$$\begin{aligned} \frac{d\hat{p}}{dz} = & -\left(\frac{\kappa}{\eta} - \frac{L^2}{\bar{\sigma}}\right)^{-1}\hat{w}_3 + i\omega\rho_f\dot{u}_3 - \left(\frac{\kappa}{\eta} - \frac{L^2}{\bar{\sigma}}\right)^{-1}\frac{Li\kappa_2}{\bar{\sigma}}\hat{H}_1 + \\ & + \left(\frac{\kappa}{\eta} - \frac{L^2}{\bar{\sigma}}\right)^{-1}\frac{Li\kappa_2}{\bar{\sigma}}\hat{H}_2 + \hat{f}_3 - \left(\frac{\kappa}{\eta} - \frac{L^2}{\bar{\sigma}}\right)^{-1}\frac{L}{\bar{\sigma}}\hat{j}_3. \end{aligned} \quad (3.28)$$

das Equações 15 e 17,

$$\frac{d\dot{u}_1}{dz} = i\omega G^{-1}\hat{\tau}_{13} - i\kappa_1\dot{u}_3, \quad (3.29)$$

$$\frac{d\dot{u}_2}{dz} = i\omega G^{-1}\hat{\tau}_{23} - i\kappa_2\dot{u}_3, \quad (3.30)$$

$$\frac{d\dot{u}_3}{dz} = \frac{-i\omega}{C^2 - M(\lambda + 2G)} \left(\frac{\kappa_1(C^2 - \lambda M)}{\omega}\dot{u}_1 + \frac{\kappa_2(C^2 - \lambda M)}{\omega}\dot{u}_2 - M\hat{\tau}_{33} - C\hat{p} \right). \quad (3.31)$$

E das outras Equações 15, 17, 20, 21 e 31, tem-se:

$$\frac{d(-\dot{w}_3)}{dz} = \frac{-i\omega}{C^2 - M\lambda} \left(-\frac{(C^2 - M\lambda)}{\omega}\kappa_1\dot{w}_1 + \frac{(C^2 - M\lambda)}{\omega}\kappa_2\dot{w}_2 + \frac{2CGi}{\omega}\frac{d\dot{u}_3}{dz} + C\hat{\tau}_{33} - \lambda\hat{p} \right). \quad (3.32)$$

A partir das Equações 11, 19 e 28 tem-se:

$$\frac{d\hat{E}_1}{dz} = \frac{L\omega\kappa_1\rho_f}{\bar{\sigma}}\dot{u}_1 + \frac{\kappa_1\kappa_2}{\bar{\sigma}}\hat{H}_1 - \left(i\omega\mu - \frac{\kappa_1^2}{\bar{\sigma}}\right)\hat{H}_2 - \frac{i\kappa_1L}{\bar{\sigma}}\frac{d\hat{p}}{dz} + \frac{i\kappa_1L}{\bar{\sigma}}\hat{f}_3 + \frac{i\kappa_1}{\bar{\sigma}}\hat{j}_3, \quad (3.33)$$

$$\frac{d\hat{E}_2}{dz} = \frac{L\omega\kappa_2\rho_f}{\bar{\sigma}}\dot{u}_3 + \left(\frac{\kappa_2^2}{\bar{\sigma}} - i\omega\mu\right)\hat{H}_1 - \frac{\kappa_1\kappa_2}{\bar{\sigma}}\hat{H}_2 - \frac{i\kappa_2L}{\bar{\sigma}}\frac{d\hat{p}}{dz} + \frac{i\kappa_2L}{\bar{\sigma}}\hat{f}_3 + \frac{i\kappa}{\bar{\sigma}}\hat{j}_3. \quad (3.34)$$

As demais Equações são decorrentes das Equações 13, 15, 20, 21, 31 e 32:

$$\frac{d\hat{\tau}_{33}}{dz} = -i\omega\rho\dot{u}_3 - i\omega\rho_f\dot{w}_3 - i\kappa_1\hat{\tau}_{13} - i\kappa_2\hat{\tau}_{23} - \hat{F}_3, \quad (3.35)$$

$$\begin{aligned} \frac{d\hat{\tau}_{13}}{dz} = & \left(-i\omega\rho + \frac{i(\lambda+2G)\kappa_1^2}{\omega} + \frac{Gi\kappa_2^2}{\omega} + \omega^2\rho_f^2\frac{\kappa}{\eta} - \kappa_1^2\rho_f C\frac{\kappa}{\eta}\right)\dot{u}_1 + \\ & + \left(\frac{i(\lambda+2G)\kappa_1\kappa_2}{\omega} - \kappa_1\kappa_2 C\rho_f\frac{\kappa}{\eta}\right)\dot{u}_2 + \\ & + L\left(-i\omega\rho_f + \frac{\kappa_1^2 Ci}{\omega}\right)\hat{E}_1 + \frac{L\kappa_1\kappa_2 Ci}{\omega}\hat{E}_2 + \left(-\omega\rho_f\kappa_1\frac{\kappa}{\eta} + \frac{\kappa_1^3 C\frac{\kappa}{\eta}}{\omega} + \kappa_1\kappa_2^2 C\frac{\kappa}{\omega\eta}\right)\hat{p} + \\ & + \frac{\kappa}{\eta}\left(-i\omega\rho_f + \frac{\kappa_1^2 Ci}{\omega}\right)\hat{f}_1 + \frac{\kappa_1\kappa_2 Ci}{\omega}\hat{f}_2 - \hat{F}_1 - \frac{\kappa_1\lambda}{\omega}\frac{d\dot{u}_3}{dz} - \frac{\kappa_1 C}{\omega}\frac{d\dot{w}_3}{dz}, \end{aligned} \quad (3.36)$$

$$\begin{aligned} \frac{d\hat{\tau}_{23}}{dz} = & \left(\frac{\kappa_1\kappa_2(\lambda+2G)i}{\omega} - \kappa_1\kappa_2 C\rho_f\frac{\kappa}{\eta}\right)\dot{u}_1 + \\ & + \left(-i\omega\rho - \left(\frac{i(\lambda+2G)\kappa_2^2}{\omega}\right) + \frac{Gi\kappa_1^2}{\omega} + \omega^2\rho_f^2 - \kappa_2^2\rho_f C\frac{\kappa}{\eta}\right)\dot{u}_2 + \\ & + \frac{L\kappa_1\kappa_2 Ci}{\omega}\hat{E}_1 + L\left(-i\omega\rho_f + \frac{\kappa_2^2 Ci}{\omega}\right)\hat{E}_2 + \left(-\omega\rho_f\kappa_2\frac{\kappa}{\eta} + \frac{\kappa_1^2\kappa_2 C\frac{\kappa}{\eta}}{\omega} + \kappa_2^3 C\frac{\kappa}{\omega\eta}\right)\hat{p} + \\ & + \left(-i\omega\rho_f + \frac{\kappa_2^2 Ci}{\omega}\frac{\kappa}{\eta}\right)\hat{f}_2 + \frac{\kappa_1\kappa_2 Ci}{\omega}\frac{\kappa}{\eta}\hat{f}_1 + -\hat{F}_2 - \frac{\kappa_2\lambda}{\omega}\frac{d\dot{u}_3}{dz} - \frac{\kappa_2 C}{\omega}\frac{d\dot{w}_3}{dz}. \end{aligned} \quad (3.37)$$

Para as Equações de 26 a 37 considera-se uma nova base ortogonal com o objetivo de simplificar o sistema. A matriz de mudança de base é:

$$\Omega = \begin{bmatrix} \frac{\kappa_1}{\kappa} & \frac{\kappa_2}{\kappa} & 0 \\ \frac{-\kappa_2}{\kappa} & \frac{\kappa_1}{\kappa} & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.38)$$

onde, dado um vetor ou campo \mathbf{X} do sistema, este é representado de acordo com a nova base da seguinte forma: $\tilde{X} = \Omega X$.

Para efeito de simplificação, definem-se as constantes

$$\beta_1 = (C^2 - M(\lambda + 2G))^{-1}, \quad (3.39)$$

$$\beta_2 = \left(1 - \frac{\kappa L^2}{\eta \bar{\sigma}}\right)^{-1}, \quad (3.40)$$

em que verificam-se as seguintes identidades

$$\beta_1 (C^2 - M\lambda) = 1 + 2\beta_1 MG, \quad (3.41)$$

$$C^2 = \beta_1^{-1} + 2MG + M\lambda, \quad (3.42)$$

$$(C^2 - M\lambda) = \beta_1 (C^2 - M\lambda)^2 - 2MG = -4MG\beta_1 (C^2 - M(\lambda + 2G)) \quad (3.43)$$

$$\frac{\beta_2 L^2 \kappa}{\eta \bar{\sigma}} + 1 = \beta^2 \quad (3.44)$$

Das Equações 31, 32 e 35, tem-se:

$$\frac{d\dot{\tilde{u}}_3}{dz} = i\omega \left(\beta_1 (C^2 - M\lambda) \gamma \dot{u}_1 - \beta_1 M \tilde{\tau}_{33} - \beta_1 C \tilde{p} \right), \quad (3.45)$$

$$\begin{aligned} \frac{d(-\dot{w}_3)}{dz} &= i\omega \left((2\beta_1 \gamma G C - i\omega \rho_f \gamma \frac{\kappa}{\eta}) \dot{u}_1 + \right. \\ &+ \left. (-\beta_1 (C^2 - 2G) + i\omega \gamma^2 \frac{\kappa}{\eta}) \tilde{p} - \gamma L \tilde{E}_1 - C \beta_1 \tilde{\tau}_{33} \right) + i\kappa \frac{\kappa}{\eta} \tilde{f}_1, \end{aligned} \quad (3.46)$$

$$\frac{d\tilde{\tau}_{33}}{dz} = i\omega (\rho \dot{\tilde{u}}_3 + \rho_f \dot{w}_3 + \gamma \tilde{\tau}_{13}) - \tilde{F}_3. \quad (3.47)$$

Da Equação 28,

$$\frac{d\hat{p}}{dz} = -i\omega \left(\rho_f \dot{u}_3 - \frac{\beta_2 \eta}{i\omega \kappa} \dot{w}_3 - \frac{\beta_2 \gamma L \eta}{\kappa \bar{\sigma}} \hat{H}_2 \right) + \hat{f}_3 - \frac{\beta_2 \gamma L \eta}{\kappa \bar{\sigma}} \hat{j}_3. \quad (3.48)$$

e das Equações 29 e 30,

$$\frac{d\dot{\tilde{u}}_1}{dz} = \frac{\kappa_1}{\kappa} \frac{d\dot{\tilde{u}}_1}{dz} + \frac{\kappa_2}{\kappa} \frac{d\dot{\tilde{u}}_1}{dz}, \quad (3.49)$$

$$\frac{d\dot{\tilde{u}}_2}{dz} = \frac{\kappa_2}{\kappa} \frac{d\dot{\tilde{u}}_1}{dz} + \frac{\kappa_1}{\kappa} \frac{d\dot{\tilde{u}}_2}{dz}, \quad (3.50)$$

conclui-se que

$$\frac{d\dot{\tilde{u}}_1}{dz} = -i\omega (G^{-1} \tilde{\tau}_{13} + \gamma \dot{u}_3) \quad (3.51)$$

$$\frac{d\dot{\tilde{u}}_2}{dz} = -i\omega (G^{-1} \gamma \tilde{\tau}_{23}). \quad (3.52)$$

Das Equações 26, 27, 33 e 34:

$$\frac{d(-\tilde{H}_1)}{dz} = -i\omega \left(\rho_f L \dot{\tilde{u}}_2 - \left(\frac{\gamma^2}{\mu} - \frac{i\bar{\sigma}}{\omega} \right) \tilde{E}_2 \right) + L \tilde{f}_2 - \tilde{j}_2 \quad (3.53)$$

$$\frac{d\tilde{H}_2}{dz} = -i\omega \left(\rho_f L \dot{\tilde{u}}_1 - \gamma L \tilde{p} - \frac{i\bar{\sigma}}{\omega} \tilde{E}_1 \right) + L \tilde{f}_1 - \tilde{j}_1, \quad (3.54)$$

$$\frac{d\tilde{E}_1}{dz} = -i\omega \left(\frac{\beta_2 L \eta \gamma}{\kappa \bar{\sigma}} \dot{w}_3 + \left(-\mu - \frac{i\omega \beta_2 \gamma^2}{\bar{\sigma}} \right) \tilde{H}_2 \right) - \frac{i\kappa \beta_2}{\bar{\sigma}} \tilde{j}_3. \quad (3.55)$$

$$\frac{d\tilde{E}_2}{dz} = -i\omega (-\mu \tilde{H}_1), \quad (3.56)$$

e das Equações 36 e 37, tem-se

$$\begin{aligned} \frac{d\tilde{\tau}_{13}}{dz} = & -i\omega(\rho + i\omega\rho_f^2\frac{\kappa}{\eta} - 4\beta_1\gamma^2G(C^2 - M(\lambda + G)))\dot{\tilde{u}}_1 + \rho_f L\tilde{E}_1 + \\ & + \left(2\beta_1\gamma CG - i\omega\rho_f\gamma\frac{\kappa}{\eta}\right)\tilde{p} + \beta_1\gamma(C^2 - M\lambda)\tilde{\tau}_{33} - \tilde{F}_1 - i\omega\frac{\kappa}{\eta}\tilde{f}_1, \end{aligned} \quad (3.57)$$

$$\frac{d\tilde{\tau}_{23}}{dz} = -i\omega\left((\rho - G\gamma^2 + i\omega\rho_f^2\frac{\kappa}{\eta})\dot{\tilde{u}}_2 + \rho_f L\tilde{E}_2\right) - \tilde{F}_2 - i\omega\rho_f\frac{\kappa}{\eta}\tilde{f}_2. \quad (3.58)$$

da Equação 18 têm-se

$$\tilde{H}_3 = \frac{\gamma}{\mu}\tilde{E}_2 \quad (3.59)$$

Da Equação 19, segue que

$$\tilde{E}_3 = \beta_2\left(\frac{i\kappa_1}{\sigma}\tilde{H}_2 - \frac{L\eta}{\kappa\sigma}\dot{\tilde{w}}_3 - \frac{1}{\sigma}\tilde{j}_3\right) \quad (3.60)$$

das Equações 20 e 21 têm-se

$$\dot{\tilde{w}}_1 = L\tilde{E}_1 - i\kappa\frac{\kappa}{\eta}\tilde{p} + i\omega\rho_f\frac{\kappa}{\eta}\dot{\tilde{u}}_1 + \frac{\kappa}{\eta}\tilde{f}_1, \quad (3.61)$$

$$\dot{\tilde{w}}_2 = L\tilde{E}_2 + i\omega\rho_f\frac{\kappa}{\eta}\dot{\tilde{u}}_2 + \frac{\kappa}{\eta}\tilde{f}_2. \quad (3.62)$$

de 23, têm-se

$$\tilde{\tau}_{12} = -G\gamma\dot{\tilde{u}}_2, \quad (3.63)$$

E das Equações 24 e 25 conclui-se que

$$\tilde{\tau}_{11} = \beta_1\left(-4G\gamma(C^2 - M(\lambda + G))\dot{\tilde{u}}_1 + (C^2 - \lambda M)\tilde{\tau}_{33} + 2GC\tilde{p}\right), \quad (3.64)$$

$$\tilde{\tau}_{22} = \beta_1\left(-2G\gamma(C^2 - \lambda M)\dot{\tilde{u}}_1 + (C^2 - \lambda M)\tilde{\tau}_{33} + 2GC\tilde{p}\right). \quad (3.65)$$

Pelas equações deduzidas anteriormente, 46 a 66, verifica-se que é possível representá-las no formalismo de Ursin, ou seja, em dois sistemas.

As variáveis que compõem os vetores $\Phi^{(1)}$ e $\Phi^{(2)}$, foram escolhidas por serem contínuas nas interfaces entre as camadas. Dessa forma, considerando que

$$\Phi^{(1)} = \left(\dot{\tilde{u}}_3, \tilde{\tau}_{13}, -\dot{\tilde{w}}_3, \tilde{H}_2, \tilde{\tau}_{33}, \dot{\tilde{u}}_1, \tilde{p}, \tilde{E}_1\right)^T \quad (3.66)$$

$$\Phi^{(2)} = \left(\dot{\tilde{u}}_2, \tilde{E}_2, \tilde{\tau}_{23}, -\tilde{H}_1\right)^T \quad (3.67)$$

escreve-se o sistema de equações de Pride como um sistema de equações diferenciais ordinárias representadas no sistema do tipo Ursin, conforme o modelo matemático apresentado em Ursin .

$$\frac{d\Phi}{dz} = -i\omega M\Phi + S = -i\omega \begin{bmatrix} 0 & M_1 \\ M_2 & 0 \end{bmatrix} \Phi + S, \quad (3.68)$$

$$\frac{d\Phi^{(j)}}{dz} = -i\omega M^{(j)}\Phi^{(j)} + S^{(j)}, j = 1, 2 \quad (3.69)$$

em que

$$M^{(j)} = \begin{bmatrix} 0_{n_j \times n_j} & M_1^{(j)} \\ M_2^{(j)} & 0_{n_j \times n_j} \end{bmatrix} \quad (3.70)$$

em que $n_j \times n_j$ é a matriz nula de ordem n_j e $M_1^{(j)}$ e $M_2^{(j)}$ são matrizes simétricas de ordem n_j .

O sistema 1 é equivalente ao que Haartsen e Pride - chamam de Sistema PSVTM, pois contém as ondas compressoriais rápida (P_f) e lenta (P_s), as ondas de cisalhamento vertical (SV) e as ondas magnéticas transversais (TM). Para este sistema, as submatrizes são

$$M_1^{(1)} = \begin{bmatrix} -\beta_1 M & \beta_1 \gamma (C^2 - \lambda M) & -\beta_1 C & 0 \\ \beta_1 \gamma (C^2 - \lambda M) & \rho + i\omega \rho_f^2 \frac{\kappa}{\eta} - 4\beta_1 \gamma^2 G (C^2 - M(\lambda + G)) & 2\beta_1 \gamma^2 G C - i\omega \rho_f \gamma \frac{\kappa}{\eta} & \rho_f L \\ -\beta_1 C & \beta_1 \gamma G C - i\omega \rho_f \gamma \frac{\kappa}{\eta} & -\beta_1 (\lambda + 2G) + i\omega \gamma^2 \frac{\kappa}{\eta} & -\gamma L \\ 0 & \rho_f L & -\gamma L & \frac{\bar{\sigma}}{i\omega} \end{bmatrix} \quad (3.71)$$

$$M_2^{(1)} = \begin{bmatrix} \rho & \gamma & -\rho_f & 0 \\ \gamma & G^{-1} & 0 & 0 \\ -\rho_f & 0 & -\frac{\beta_2 \eta}{i\omega \kappa} & -\frac{\beta_2 \gamma L \eta}{\kappa \bar{\sigma}} \\ 0 & 0 & -\frac{\beta_2 \gamma L \eta}{\kappa \bar{\sigma}} & -\mu - \frac{i\omega \beta_2 \gamma^2}{\bar{\sigma}} \end{bmatrix} \quad (3.72)$$

A fonte deste sistema é dada por

$$S^{(1)} = \left(0, -\tilde{F}_1 - i\omega \rho_f \frac{\kappa}{\eta} \tilde{f}_1, i\kappa \frac{\kappa}{\eta} \tilde{f}_1, -\tilde{j}_1 - L \tilde{f}_1, -\tilde{F}_3, 0, \tilde{f}_3 - \beta_2 \frac{L \eta}{\kappa \bar{\sigma}} \tilde{j}_3, -ik \frac{\beta_2}{\bar{\sigma}} \tilde{j}_3 \right)^T. \quad (3.73)$$

Determinando $\Phi^{(1)}$, pode-se calcular as quatro variáveis que dependem apenas do sistema 1:

$$\begin{aligned} \tilde{E}_3 &= \beta_2 \left(\frac{i\kappa_1}{\bar{\sigma}} \tilde{H}_2 - \frac{L\eta}{\kappa \bar{\sigma}} \dot{\tilde{w}}_3 - \frac{1}{\bar{\sigma}} \tilde{j}_3 \right), \\ \dot{\tilde{w}}_1 &= L \tilde{E}_1 - i\kappa \frac{\kappa}{\eta} \tilde{p} + i\omega \rho_f \frac{\kappa}{\eta} \dot{\tilde{u}}_1 + \frac{\kappa}{\eta} \tilde{f}_1, \\ \tilde{\tau}_{11} &= \beta_1 \left(-4G\gamma (C^2 - M(\lambda + G)) \dot{\tilde{u}}_1 + (C^2 - \lambda M) \tilde{\tau}_{33} + 2GC\tilde{p} \right), \\ \tilde{\tau}_{22} &= \beta_1 \left(-2G\gamma (C^2 - \lambda M) \dot{\tilde{u}}_1 + (C^2 - \lambda M) \tilde{\tau}_{33} + 2GC\tilde{p} \right). \end{aligned} \quad (3.74)$$

O sistema 2 é equivalente ao que Haartsen e Pride - chamam de Sistema SHTE, pois contém as ondas de cisalhamento horizontal (onda SH) e as ondas elétricas transversais (onda TE). Para este sistema,

$$M_1^{(2)} = \begin{bmatrix} \frac{1}{G} & 0 \\ 0 & -\mu \end{bmatrix}, \quad (3.75)$$

$$M_2^{(2)} = \begin{bmatrix} \rho - G\gamma^2 + i\omega\rho_f^2\frac{\kappa}{\eta} & \rho_f L \\ \rho_f L & \frac{\bar{\sigma}}{i\omega} + \frac{\gamma^2}{\mu} \end{bmatrix}, \quad (3.76)$$

e seu vetor fonte é dado por

$$S^{(2)} = \left(0, 0, -\tilde{F}_2 - i\omega\rho_f\frac{\kappa}{\eta}\tilde{f}_2, -\tilde{j}_2 - L\tilde{f}_2 \right)^T. \quad (3.77)$$

Determinando $\Phi^{(2)}$, podem-se calcular as três variáveis que dependem apenas do sistema 2:

$$\begin{aligned} \tilde{H}_3 &= \frac{\gamma}{\mu}\tilde{E}_2, \\ \dot{w}_2 &= L\tilde{E}_2 + i\omega\rho_f\frac{\kappa}{\eta}\dot{u}_2 + \frac{\kappa}{\eta}\tilde{f}_2, \\ \tilde{\tau}_{12} &= -G\gamma\dot{u}_2. \end{aligned} \quad (3.78)$$

3.5 Identificação de pacotes – assuntos

Em UML, um pacote é um mecanismo de agrupamento genérico que contém classes que fazem parte de um assunto e relacionam-se por um conceito comum. Em outras palavras, agrupam classes que se relacionam com maior frequência.

- Métodos Numéricos e Computacionais: Realiza operações de álgebra linear para resolver os sistemas de equações diferenciais parciais, além de implementar transformadas rápidas de Fourier e suas inversas para transição entre os domínios do tempo e frequência.
- Propriedades do Sistema: pacote que armazena as propriedades físicas e mecânica das rochas, além das propriedades dos fluidos e a configuração e gerenciamento dos parâmetros globais do sistema.
- Simulador: relaciona os pacotes, sendo responsável pela criação e destruição de objetos.
- Pacote Gráfico: é um pacote que utiliza o gnuplot para plotar os resultados obtidos.
- Biblioteca: serão utilizadas métodos e funções já existentes para a resolução dos cálculos, bibliotecas padrão de C++ tais como (STL), etc.

3.6 Diagrama de pacotes – assuntos

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

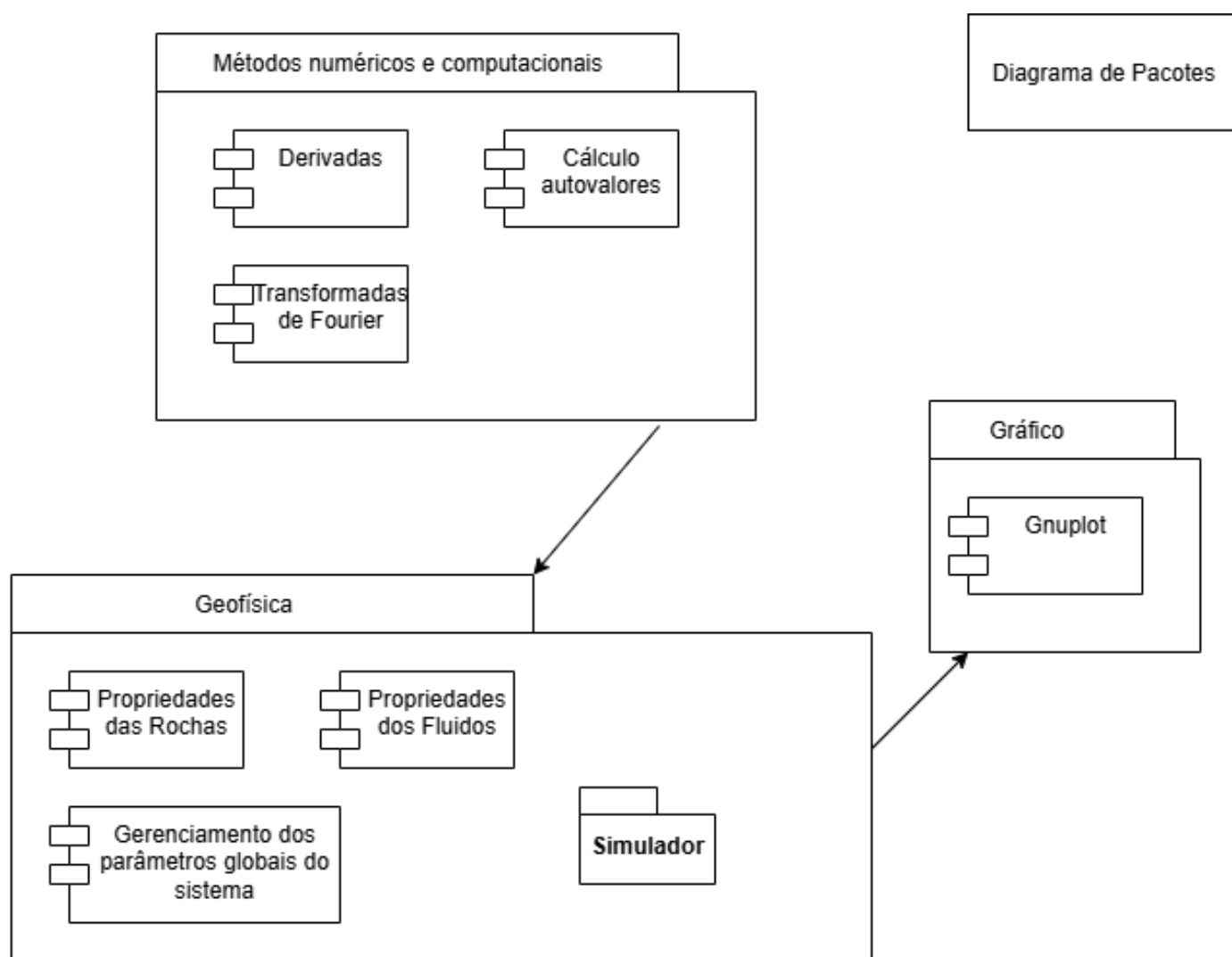


Figura 3.1: Diagrama de Pacotes

Capítulo 4

AOO – Análise Orientada a Objeto

A terceira etapa do desenvolvimento de um projeto de engenharia, no nosso caso um software aplicado a engenharia de petróleo, é a AOO – Análise Orientada a Objeto. A AOO utiliza algumas regras para identificar os objetos de interesse, as relações entre os pacotes, as classes, os atributos, os métodos, as heranças, as associações, as agregações, as composições e as dependências. O resultado da análise é um conjunto de diagramas que identificamos objetos e seus relacionamentos.

4.1 Diagramas de classes

O diagrama de classes é apresentado na Figura 4.1.

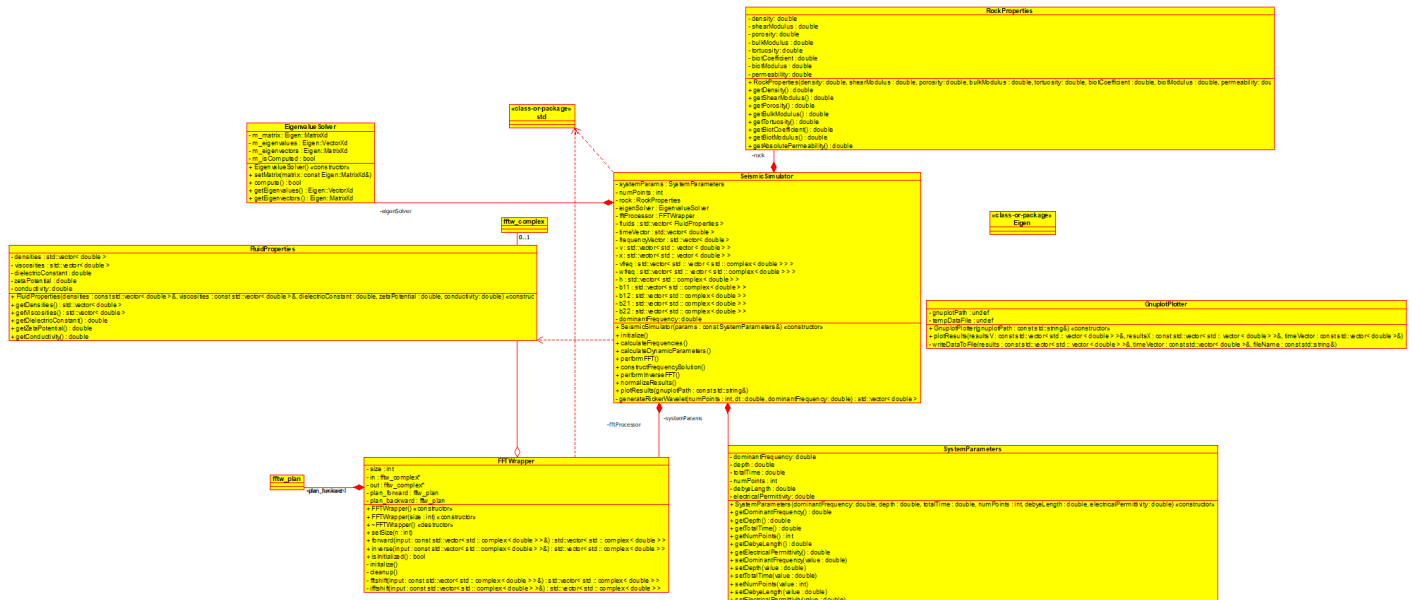


Figura 4.1: Diagrama de classes

4.1.1 Dicionário de classes

- Classe **seismicSimulator**: Representa o núcleo do simulador do efeito sísmicoelétrico. É responsável por gerenciar o fluxo completo da simulação, desde a inicialização dos parâmetros, cálculos no domínio da frequência e do tempo, até a normalização e visualização dos resultados. Atua como

orquestrador que integra todas as outras classes. Têm como função inicializar os parâmetros do sistema, executa as transformadas, construir as soluções.

- Classe **systemParameters**: Representa os parâmetros globais do sistema utilizados na simulação, como frequência dominante, profundidade da fonte sísmica, tempo total de simulação, número de pontos de discretização, comprimento de Debye e permissividade elétrica do meio. É a base de configuração do simulador.
- Classe **rockProperties**: Modela as propriedades físicas da rocha porosa onde o efeito sismoelétrico será simulado. Inclui parâmetros como porosidade, permeabilidade, densidade, módulo de cisalhamento, módulo de bulk, módulo de Biot e fator de tortuosidade.
- Classe **fluidProperties**: Representa as propriedades dos fluidos presentes nos poros da rocha. Inclui viscosidade, condutividade elétrica, densidade e constante dielétrica. Cada instância da classe pode modelar um fluido diferente, permitindo a simulação de cenários multi-fluídos. É responsável por fornecer os dados para o cálculo das respostas eletromecânicas no meio poroso saturado.
- Classe **FFTWrapper**: Fornece uma abstração para o uso da biblioteca FFTW (Fast Fourier Transform), facilitando a execução de transformadas rápidas de Fourier e suas inversas. É utilizada pelo simulador para transformar sinais entre os domínios do tempo e da frequência.
- Classe **eigenvalueSolver**: Implementa os cálculos necessários para resolver os autovalores e autovetores associados ao sistema dinâmico do simulador. Essencial para determinar os modos próprios de vibração e propagação de ondas no meio poroelástico. É base para calcular parâmetros críticos que descrevem as soluções no domínio da frequência.
- Classe **gnuplotPlotter**: classe que fornece os métodos necessários para a geração de gráficos.

4.2 Diagrama de seqüência – eventos e mensagens

O diagrama de seqüência enfatiza a troca de eventos e mensagens e sua ordem temporal. Contém informações sobre o fluxo de controle do software. Costuma ser montado a partir de um diagrama de caso de uso e estabelece o relacionamento dos atores (usuários e sistemas externos) com alguns objetos do sistema. O diagrama de seqüência pode ser geral, englobando todas as operações do sistema ou específico, que enfatiza uma determinada operação.

4.2.1 Diagrama de seqüência geral

Veja o diagrama de seqüência na Figura 4.2.

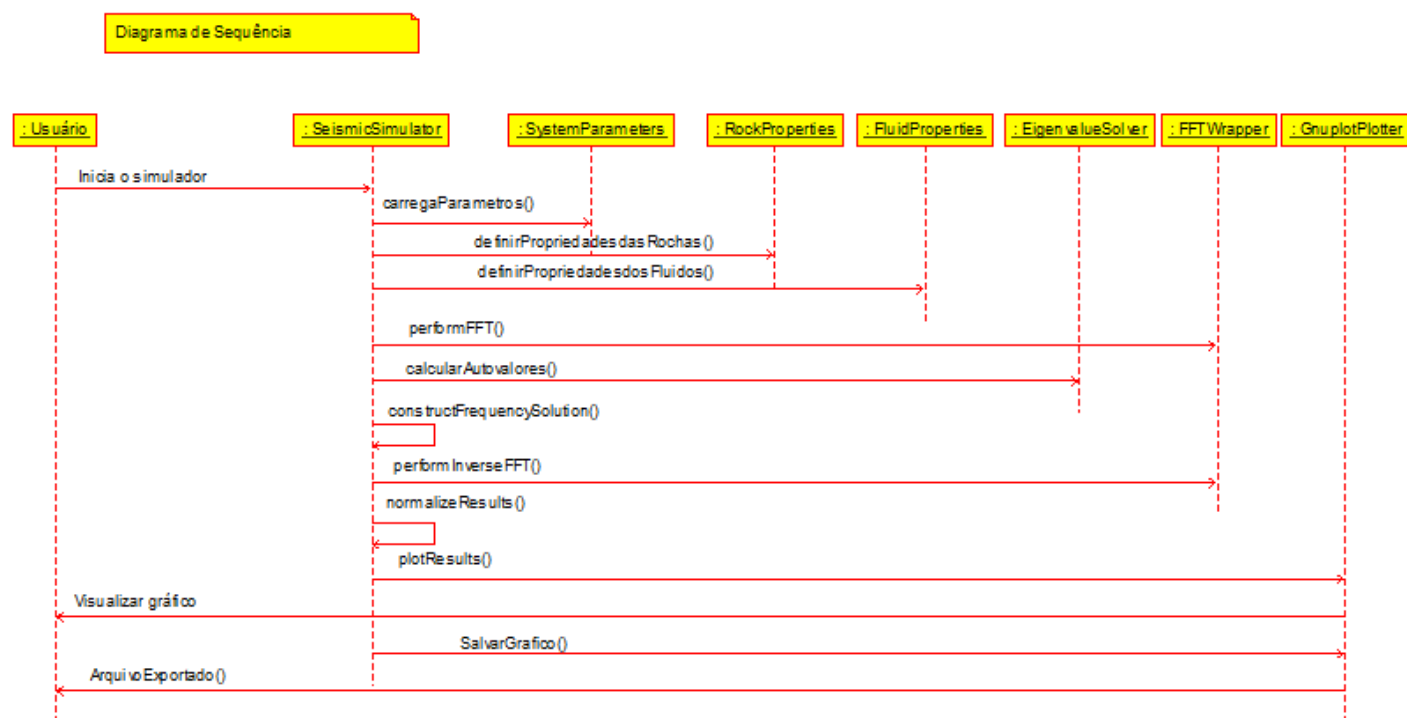


Figura 4.2: Diagrama de sequência

4.3 Diagrama de comunicação – colaboração

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

Veja na Figura 4.3 o diagrama de comunicação com foco no simulador propriamente dito. As linhas indicam ora criação de objetos ora acesso a funções das classes.

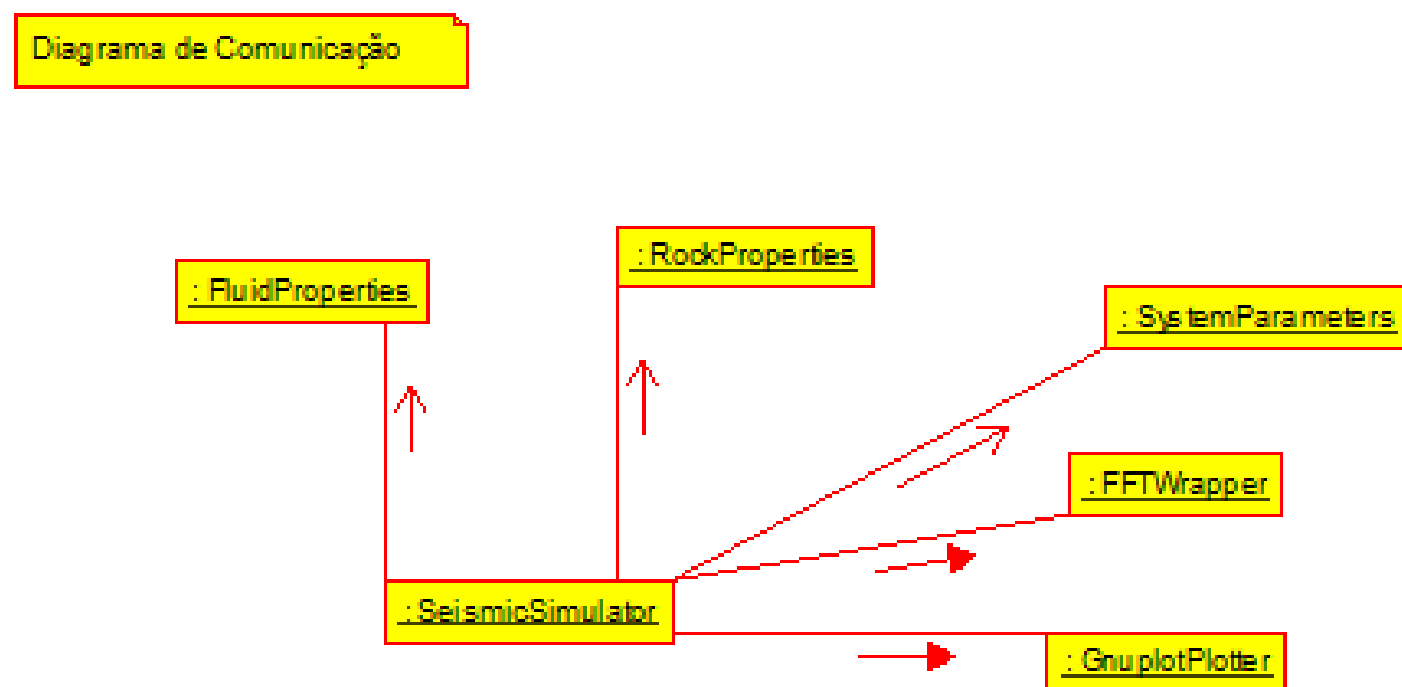


Figura 4.3: Diagrama de comunicação

4.4 Diagrama 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. Existem basicamente dois usos para máquinas de estado: máquinas de estado comportamentais e máquinas de estado para protocolos.

Veja na Figura 4.4 o diagrama de máquina de estado para o objeto seismicSimulator.

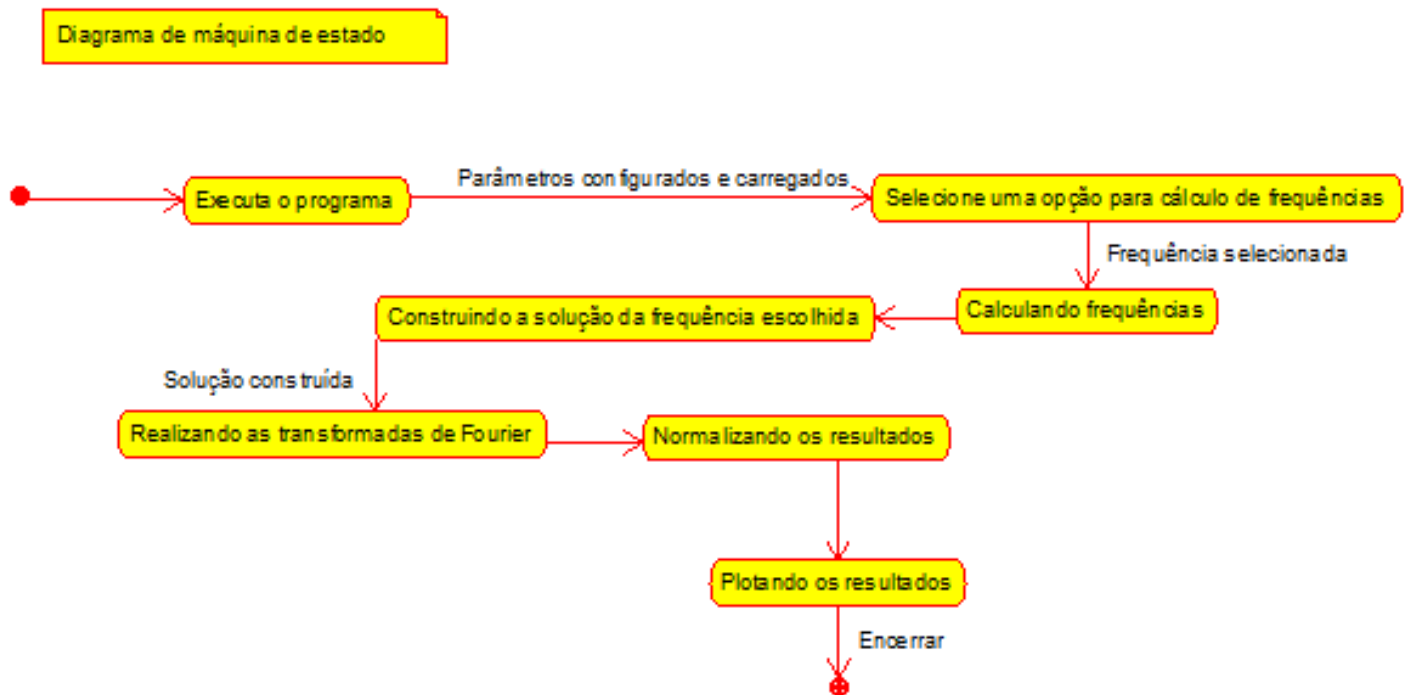


Figura 4.4: Diagrama de máquina de estado

4.5 Diagrama de atividades

O diagrama de atividades é um diagrama UML utilizado para modelar o aspecto comportamental de processos. Neste diagrama, uma atividade é modelada como uma sequência estruturada de ações controladas por nós de decisão e sincronismo.

Veja na Figura 4.5 o diagrama de atividades correspondente a uma atividade específica do diagrama de máquina de estado.

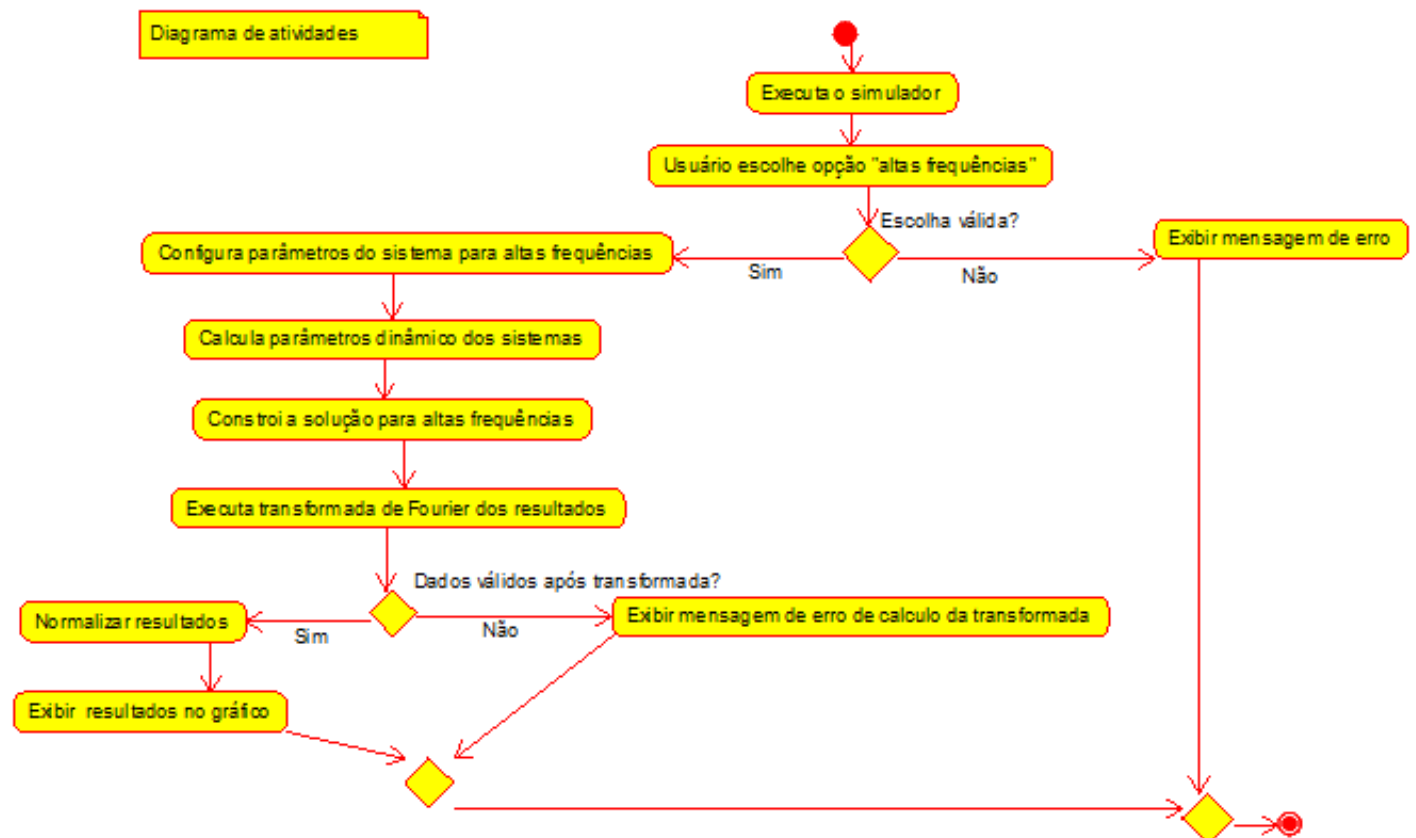


Figura 4.5: Diagrama de atividades

Capítulo 5

Projeto

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

5.1 Projeto do Sistema

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

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

1. Protocolos

- O programa realiza entrada de dados via arquivos de configuração em formato texto (.h e .cpp).
- Os dados gerados são exportados e visualizados por meio do componente externo Gnuplot, responsável pela geração de gráficos.
- A comunicação interna entre módulos segue protocolos baseados em interfaces orientadas a objetos.

2. Recursos

- O simulador utiliza recursos computacionais como CPU, memória RAM e disco rígido para armazenamento temporário e processamento dos dados.
- O simulador utiliza o programa externo Gnuplot que plota figuras e gráficos.

- O desenvolvimento requer compilador C++ com suporte à biblioteca FFTW para cálculos de transformadas rápidas de Fourier.

3. Controle

- O fluxo do programa é controlado sequencialmente através da classe principal `seismicSimulator`, que coordena etapas como inicialização, processamento no domínio da frequência e transformação inversa.

4. Plataformas

- O programa é multiplataforma, podendo ser executado em sistemas Windows, Linux e macOS.
- O ambiente de desenvolvimento principal é baseado em compiladores compatíveis com C++ moderno (ex: GCC, Clang, MSVC).
- O software utilizara a biblioteca externa `gnuPlotter` que permite acesso ao programa `Gnuplot` e também a biblioteca `FFTW`.

5.2 Projeto Orientado a Objeto – POO

O projeto orientado a objeto é a etapa posterior ao projeto do sistema. Baseia-se na análise, mas considera as decisões do projeto do sistema. Acrescenta a análise desenvolvida e as características da plataforma escolhida (hardware, sistema operacional e linguagem de 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

- O programa utiliza o HD, o processador e o teclado do computador.
- O Software pode ser executado nas plataformas GNU/Linux ou Windows.
- Existe a necessidade de instalação do software `Gnuplot` para o funcionamento do programa.

Efeitos do projeto no modelo dinâmico

- Revisar os diagramas de seqüência e de comunicação considerando a plataforma escolhida.
- Verificar a necessidade de se revisar, ampliar e adicionar novos diagramas de máquinas de estado e de atividades.

Efeitos do projeto nos atributos

- Atributos novos podem ser adicionados a uma classe, como, por exemplo, atributos específicos de uma determinada linguagem de softwareção (acesso a disco, ponteiros, constantes e informações correlacionadas).

Efeitos do projeto nas heranças

- Revise as heranças no diagrama de classes.

Efeitos do projeto nas associações

- Reorganização das associações.

Efeitos do projeto nas otimizações

- Uma melhoria do programa recomendada é a implementação de uma interface de entrada de dados ao inicializar o programa, permitindo que o usuário insira as propriedades diretamente no momento da execução. Essa abordagem tornaria o simulador mais acessível e amigável, dispensando a necessidade de interação com o código-fonte e aumentando a flexibilidade do sistema.

As dependências dos arquivos e bibliotecas podem ser descritos pelo diagrama de componentes, e as relações e dependências entre o sistema e o hardware podem ser ilustradas com o diagrama de implantação.

5.3 Diagrama de Componentes

O diagrama de componentes mostra a forma como os componentes do software se relacionam, suas dependências. Inclui itens como: componentes, subsistemas, executáveis, nós, associações, dependências, generalizações, restrições e notas. Exemplos de componentes são bibliotecas estáticas, bibliotecas dinâmicas, dlls, componentes Java, executáveis, arquivos de disco, código-fonte.

Veja na Figura 5.1 um exemplo de diagrama de componentes. A partir do diagrama de componentes, pode-se identificar todos os arquivos necessários para a compilação e execução do software, bem como compreender as relações de dependências dos módulos. Observa-se, que o componente seismicSimulator apresenta dependência direta de todos os demais componentes para seu funcionamento.

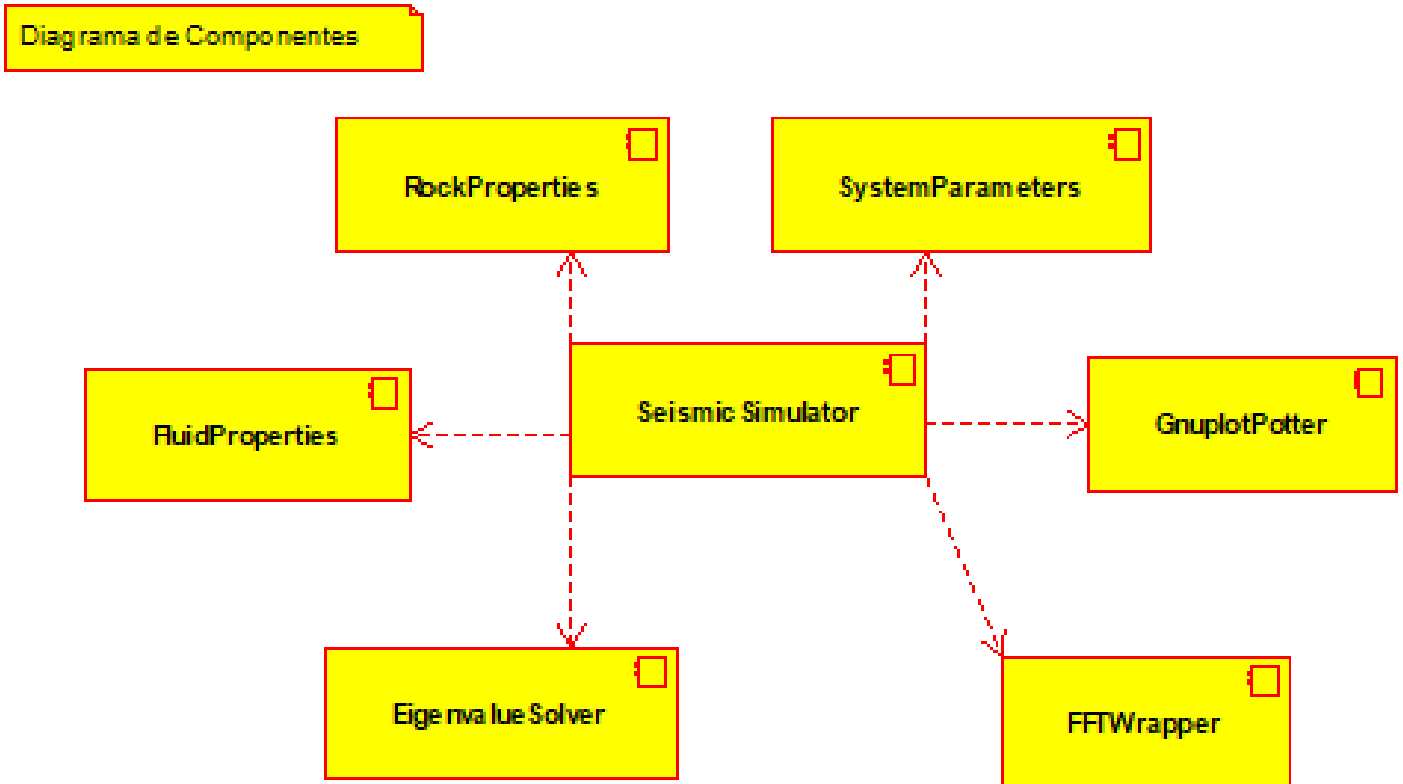


Figura 5.1: Diagrama de componentes

5.4 Diagrama de Implantação

O diagrama de implantação é um diagrama de alto nível que inclui relações entre o sistema e o hardware e que se preocupa com os aspectos da arquitetura computacional escolhida. Seu enfoque é o hardware, a configuração dos nós em tempo de execução. 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 5.2 um exemplo de diagrama de implantação utilizado. Para que haja um correto e realístico desempenho da simulação pelo software, é necessário que haja o computador com todos os hardwares requeridos.

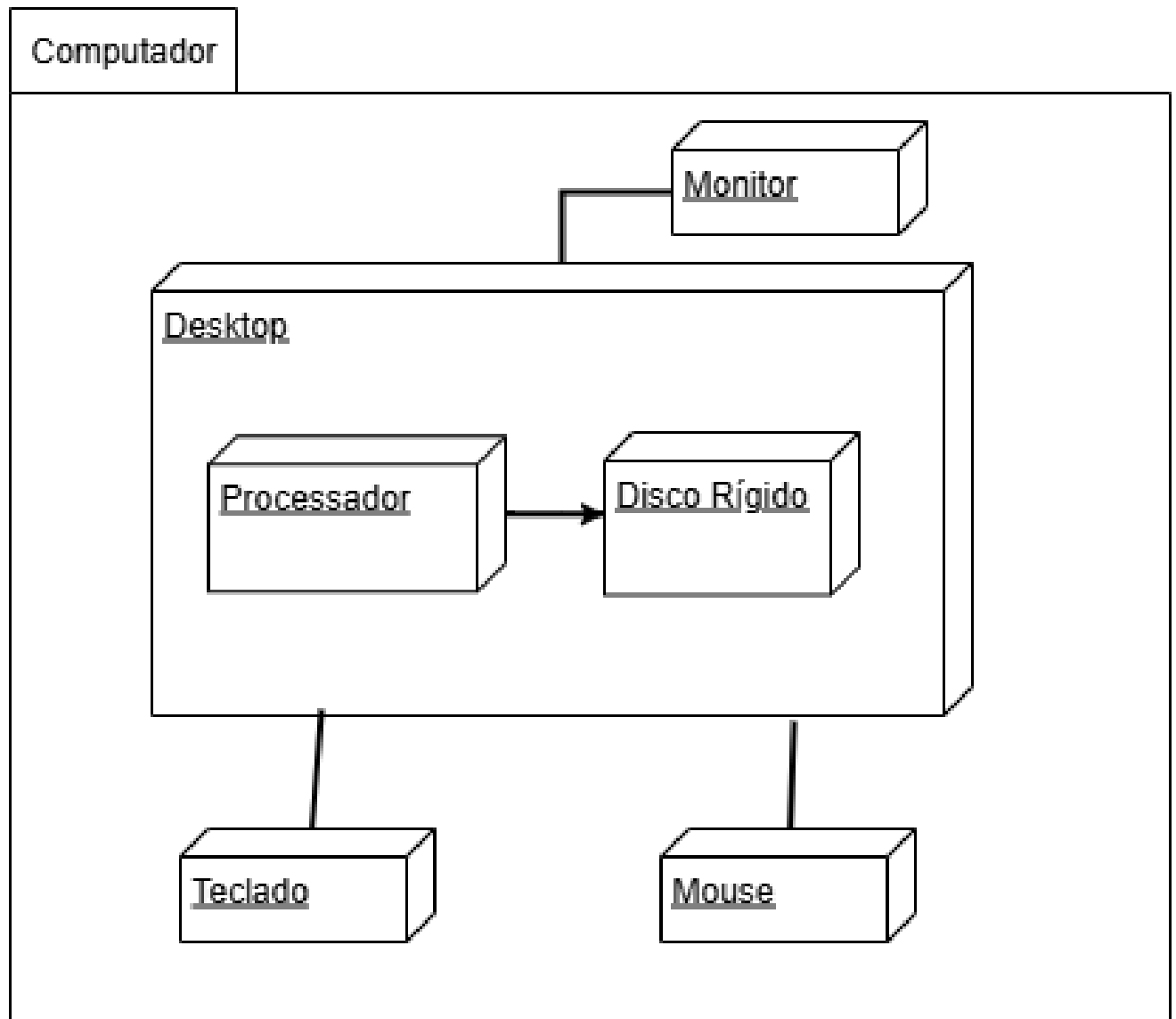
Diagrama de Implantação

Figura 5.2: Diagrama de implantação

Capítulo 6

Implementação

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

6.1 Código fonte

Apresenta-se a seguir um conjunto de classes (arquivos .h e .cpp) além do programa main. Apresenta-se na listagem 6.1 o arquivo com o código da função main.

Listing 6.1: Implementação da função main().

```
1 // main.cpp
2 #include "seismicSimulator.h"
3 #include <iostream>
4
5 void printSystemParameters(const SystemParameters& params) {
6     std::cout << "Parametros configurados:" << std::endl;
7     std::cout << "Frequencia dominante:" << params.getDominantFrequency() <<
8         "\nHz" << std::endl;
9     std::cout << "Profundidade:" << params.getDepth() << "\nm" << std::endl;
10    std::cout << "Tempo total:" << params.getTotalTime() << "\ns" << std::
11        endl;
12    std::cout << "Numero de pontos:" << params.getNumPoints() << std::endl;
13    std::cout << "Comprimento de Debye:" << params.getDebyeLength() << "\nm"
14        << std::endl;
15    std::cout << "Permissividade elétrica:" << params.
16        getElectricalPermittivity() << "\nF/m" << std::endl;
17 }
18
19 int main() {
20     // 1. Selecao do modelo de frequencia
21     int modelo;
22     std::cout << "Informe o MODELO que deseja simular:" << std::endl;
23     std::cout << "\t1-> Baixas frequencias" << std::endl;
```

```

20     std::cout << "\t2->Altas frequencias" << std::endl;
21     std::cin >> modelo;
22
23     double dominantFrequency, totalTime;
24     if (modelo == 1) {
25         dominantFrequency = 20.0; // Hz para baixas frequencias
26         totalTime = 2.0; // Tempo total [s]
27         std::cout << "Modelo de baixas frequencias selecionado." << std::endl
28             ;
29     } else if (modelo == 2) {
30         dominantFrequency = 200000; // Hz para altas frequencias
31         totalTime = 1; // Tempo total [s]
32         std::cout << "Modelo de altas frequencias selecionado." << std::endl;
33     } else {
34         std::cerr << "Opcao invalida. Saindo do programa." << std::endl;
35         return 1;
36     }
37
38     // 2. Configuracao dos parametros do sistema
39     double depth = 1000.0; // Profundidade
40     int numPoints = 320; // Numero de pontos de discretizacao
41     double debyeLength = 1e-2; // Comprimento de Debye
42     double electricalPermittivity = 8.85e-12; // Permissividade elétrica
43
44     std::cout << "Configurando os parametros do sistema." << std::endl;
45     SystemParameters systemParams(dominantFrequency, depth, totalTime,
46         numPoints, debyeLength, electricalPermittivity);
47     printSystemParameters(systemParams);
48     std::cout << "Parametros do sistema configurados com sucesso." << std::
49         endl;
50
51     // 3. Inicializacao do simulador
52     std::cout << "Inicializando o simulador." << std::endl;
53     try {
54         SeismicSimulator simulator(systemParams);
55
56         // 4. Calculo das frequencias e propriedades dinamicas
57         std::cout << "Calculando as frequencias." << std::endl;
58         simulator.calculateFrequencies();
59         std::cout << "Frequencias calculadas com sucesso." << std::endl;
60
61         std::cout << "Calculando os parametros dinamicos." << std::endl;
62         simulator.calculateDynamicParameters();

```

```

60     std::cout << "Parametros_dinamicos_calculados_com_sucesso." << std::
        endl;
61
62     // 5. Realizacao da FFT para transformar os dados para o dominio da
        frequencia
63     std::cout << "Realizando_a_FFT_para_cada_fluido." << std::endl;
64     simulator.performFFT(); // A geracao do sinal de entrada esta dentro
        deste método
65     std::cout << "FFT_realizada_com_sucesso." << std::endl;
66
67     // 6. Construção da solução no dominio da frequencia (vfreq e wfreq)
68     simulator.constructFrequencySolution();
69     std::cout << "Solucao_no_dominio_da_frequencia_construida_com_sucesso
        ." << std::endl;
70
71     // 7. Realizacao da transformada inversa para obter os sinais no
        dominio do tempo
72     simulator.performInverseFFT();
73     std::cout << "Transformada_inversa_realizada_com_sucesso." << std::
        endl;
74
75     // 8. Normalizacao dos resultados para manter a escala de amplitude
        identica ao MATLAB
76     simulator.normalizeResults();
77     std::cout << "Resultados_normalizados_com_sucesso." << std::endl;
78
79     // 9. Plotagem dos resultados
80     simulator.plotResults("c:/gnuplot");
81     std::cout << "Resultados_plotados_com_sucesso." << std::endl;
82
83     std::cout << "Simulacao_concluida_com_sucesso." << std::endl;
84 } catch (const std::exception& e) {
85     std::cerr << "Erro_durante_a_execucao_do_simulador:" << e.what() <<
        std::endl;
86     return 1;
87 }
88
89 std::cout << "Pressione_Enter_para_sair...";
90 std::cin.ignore();
91 std::cin.get();
92
93 return 0;
94 }

```

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

Listing 6.2: Arquivo de cabeçalho da classe seismicSimulator.

```

1 #ifndef SEISMIC_SIMULATOR_H
2 #define SEISMIC_SIMULATOR_H
3
4 #include "systemParameters.h"
5 #include "rockProperties.h"
6 #include "fluidProperties.h"
7 #include "eigenValueSolver.h"
8 #include "gnuPlotter.h"
9 #include "fft.h"
10 #include <vector>
11 #include <complex>
12 #include <iostream>
13
14 class SeismicSimulator {
15 public:
16     SeismicSimulator(const SystemParameters& params);
17
18     void initialize();
19     void calculateFrequencies();
20     void calculateDynamicParameters();
21     void performFFT();
22     void constructFrequencySolution();
23     void performInverseFFT();
24     void normalizeResults();
25     void plotResults(const std::string& gnuplotPath);
26
27 private:
28     SystemParameters systemParams;
29     int numPoints;
30     RockProperties rock;
31     EigenvalueSolver eigenSolver;
32     FFTWrapper fftProcessor;
33
34     std::vector<FluidProperties> fluids;
35     std::vector<double> timeVector;
36     std::vector<double> frequencyVector;
37     std::vector<std::vector<double>> v, x; // Resultados no domínio do tempo
38     std::vector<std::vector<std::complex<double>>> vfreq, wfreq; //
        Resultados no domínio da frequência
39     std::vector<std::complex<double>> h; // Vetor h
40     std::vector<std::complex<double>> b11, b12, b21, b22; // Parâmetros

```

```

        calculados
41     double dominantFrequency;
42
43 private:
44     std::vector<double> generateRickerWavelet(int numPoints, double dt,
        double dominantFrequency);
45 };
46
47 #endif // SEISMIC_SIMULATOR_H

```

Apresenta-se na listagem 6.3 o arquivo com o código da implementação da classe seismicSimulator.

Listing 6.3: Arquivo de implementação da classe seismicSimulator.

```

1
2 #include "seismicSimulator.h"
3 #include <functional>
4 #include <iostream>
5 #include <numeric>
6 #include <cmath>
7
8 using namespace std::literals::complex_literals;
9
10 SeismicSimulator::SeismicSimulator(const SystemParameters& params)
11     : systemParams(params), numPoints(params.getNumPoints()),
12     rock(2619.0, 1590000000.0, 0.4, 262000000.0, 2.4, 0.88, 8150000000.0,
        1.97e-3),
13     fftProcessor(numPoints) { // Inicializa o fftProcessor com numPoints
14
15     std::cout << "Construtor do SeismicSimulator chamado com sucesso." << std
        ::endl;
16
17     dominantFrequency = systemParams.getDominantFrequency();
18     std::cout << "Frequencia dominante configurada: " << dominantFrequency <<
        " Hz." << std::endl;
19
20     // Inicializar os fluidos conforme os parametros fornecidos
21     fluids.push_back(FluidProperties({1040.0}, {0.0015}, 80.0, -0.044, 7.54e
        -4));
22     fluids.push_back(FluidProperties({844.8}, {0.008989}, 80.0, -0.044, 7.54e
        -4));
23     fluids.push_back(FluidProperties({905.1}, {0.007379}, 80.0, -0.044, 7.54e
        -4));
24     fluids.push_back(FluidProperties({967.2}, {0.0085378}, 80.0, -0.044, 7.54
        e-4));

```

```

25     std::cout << "Propriedades dos fluidos inicializadas com sucesso." << std
        ::endl;
26
27     // Inicializacao adicional
28     initialize();
29
30     std::cout << "Construtor do SeismicSimulator concluido com sucesso." <<
        std::endl;
31 }
32
33 void SeismicSimulator::initialize() {
34     std::cout << "Inicializando vetores de tempo e frequencia." << std::endl;
35
36     //double T = systemParams.getTotalTime();
37     double T = 20;
38     int N = numPoints;
39     double dt = T / (N - 1); // Passo de tempo
40     double df = (N - 1) / (N * T); // Passo de discretizacao da frequencia
41     double fNyq = 1.0 / (2.0 * dt); // frequencia de Nyquist
42
43     // Inicializar vetores de tempo e frequencia
44     timeVector.resize(N);
45     frequencyVector.resize(N);
46     h.resize(N, 0.0); // Inicializa h com zeros
47
48     // Preencher o vetor de tempo
49     for (int j = 0; j < N; ++j) {
50         timeVector[j] = j * dt;
51     }
52
53     std::cout << "Vetores de tempo e frequencia inicializados." << std::endl;
54
55     // Inicializar vfreq e wfreq para armazenar os resultados da FFT
56     vfreq.resize(fluids.size(), std::vector<std::complex<double>>(N, 0.0));
57     wfreq.resize(fluids.size(), std::vector<std::complex<double>>(N, 0.0));
58
59     std::cout << "Vetores vfreq e wfreq inicializados." << std::endl;
60
61     // Configurar o FFTWrapper
62     fftProcessor.setSize(N);
63     if (!fftProcessor.isInitialized()) {
64         throw std::runtime_error("Falha ao inicializar o FFTProcessor.");
65     }

```

```

66
67 // Inicializar o vetor v com zeros para todos os fluidos
68 v.resize(fluids.size(), std::vector<double>(N, 1.0));
69 x.resize(fluids.size(), std::vector<double>(N, 1.0));
70
71 std::cout << "FFTProcessor_inicializado_com_sucesso." << std::endl;
72 std::cout << "Inicializacao_do_simulador_concluida_com_" << N << "_pontos
    ." << std::endl;
73 }
74
75
76
77 void SeismicSimulator::calculateDynamicParameters() {
78     std::cout << "Calculando_parametros_dinamicos." << std::endl;
79
80     size_t numFluids = fluids.size();
81     b11.resize(numFluids);
82     b12.resize(numFluids);
83     b21.resize(numFluids);
84     b22.resize(numFluids);
85
86     double wd = 4 * M_PI * dominantFrequency; // frequencia angular dominante
87     double C = rock.getBiotCoefficient();
88     double M = rock.getBiotModulus();
89
90     for (size_t i = 0; i < numFluids; ++i) {
91         const auto& fluid = fluids[i];
92
93         // Propriedades fisicas necessarias
94         double phi = rock.getPorosity();
95         double eta = fluid.getViscosities()[0];
96         double rhof = fluid.getDensities()[0];
97         double F = rock.getTortuosity() / phi;
98         double kk = rock.getAbsolutePermeability();
99         double rho = (1.0 - phi) * rock.getDensity() + phi * rhof;
100        double lamb = rock.getBulkModulus();
101        double G = rock.getShearModulus();
102        double beta = 1.0 / (C * C - M * (lamb + 2 * G));
103
104        // Calculo de kappa conforme o MATLAB
105        double kappa = kk * (1.0 - std::sqrt(eta / (F * kk * rhof) * 4.0 / wd
            ));
106

```



```

107     // Construir a matriz para calculo dos autovalores
108     Eigen::MatrixXd matrix(2, 2);
109     matrix(0, 0) = beta * (2.0 * C * rhof - kappa * (lamb + 2 * G));
110     matrix(1, 1) = beta * (2.0 * C * rhof + kappa * (lamb + 2 * G));
111     matrix(0, 1) = kappa * M;
112     matrix(1, 0) = kappa * M;
113
114     // Calcular autovalores e autovetores
115     Eigen::EigenSolver<Eigen::MatrixXd> eigenSolver(matrix);
116     if (eigenSolver.info() != Eigen::Success) {
117         std::cerr << "Erro ao calcular autovalores para o fluido" << i +
118             1 << "." << std::endl;
119         continue;
120     }
121
122     // Obter os autovalores e autovetores
123     Eigen::VectorXd eigenvalues = eigenSolver.eigenvalues().real();
124     Eigen::MatrixXd eigenvectors = eigenSolver.eigenvectors().real();
125
126     // Armazenar os valores nos vetores bij
127     b11[i] = eigenvalues[0];
128     b12[i] = eigenvalues[1];
129     b21[i] = eigenvectors(0, 0);
130     b22[i] = eigenvectors(1, 0);
131
132     std::cout << "Parametros dinamicos calculados com sucesso." << std::endl;
133 }
134
135 void SeismicSimulator::calculateFrequencies() {
136     // Configuracao das frequencias conforme o MATLAB
137     std::cout << "Calculando as frequencias." << std::endl;
138
139     // Número de pontos
140     size_t N = numPoints;
141
142     // Tempo total e intervalo de tempo
143     double T = systemParams.getTotalTime();
144     double dt = T / N;
145
146     // frequencia de Nyquist
147     double fNyq = 1.0 / (2.0 * dt);
148

```

```

149 // Passo de discretizacao da frequencia
150 double df = 1.0 / T;
151
152 // Definindo o vetor de frequencias de acordo com o MATLAB
153 frequencyVector.resize(N);
154 // Calcular o vetor de frequencia conforme o MATLAB
155 for (int j = 0.0001; j < N; ++j) {
156     frequencyVector[j] = j * df;
157 }
158
159
160 std::cout << "frequencias calculadas com sucesso." << std::endl;
161 }
162
163 void SeismicSimulator::performFFT() {
164     // Atualiza o vetor de frequencias
165     calculateFrequencies();
166
167     // Realiza a FFT para cada fluido
168     std::cout << "Realizando a FFT para cada fluido." << std::endl;
169     for (size_t i = 0; i < fluids.size(); ++i) {
170         std::cout << "Processando FFT para o fluido " << i + 1 << "." << std
171             ::endl;
172
173         // Geracao do sinal de entrada (Ricker Wavelet)
174         double dt = systemParams.getTotalTime() / (numPoints - 1);
175         double dominantFrequency = systemParams.getDominantFrequency();
176         std::vector<double> inputSignal = generateRickerWavelet(numPoints, dt
177             , dominantFrequency);
178
179         // Garantir que o tamanho do sinal de entrada seja exatamente igual a
180             numPoints
181         if (inputSignal.size() != numPoints) {
182             std::cerr << "Erro: tamanho do sinal de entrada nao corresponde a
183                 numPoints." << std::endl;
184             inputSignal.resize(numPoints, 0.0); // Redimensiona preenchendo
185                 com zeros se necessario
186         }
187
188         // Verificar se o sinal gerado nao esta zerado
189         double maxInputSignal = *std::max_element(inputSignal.begin(),
190             inputSignal.end());
191         std::cout << "Maxima amplitude do sinal de entrada: " <<

```

```

maxInputSignal << std::endl;
186 if (maxInputSignal < 1e-6) {
187     std::cerr << "Aviso: o sinal de entrada para o fluido" << i + 1
        << " tem amplitude muito baixa." << std::endl;
188     continue; // Pular este fluido, pois o sinal esta muito baixo
189 }
190
191 // Preencher o vetor complexInput com os dados do sinal de entrada
192 std::vector<std::complex<double>> complexInput(numPoints);
193 for (int j = 0; j < numPoints; ++j) {
194     complexInput[j] = std::complex<double>(inputSignal[j], 0.0);
195 }
196
197 try {
198     // Verificar se o FFTWrapper foi inicializado corretamente
199     if (!fftProcessor.isInitialized()) {
200         throw std::runtime_error("FFTWrapper nao foi inicializado
        corretamente.");
201     }
202
203     // Realizar a FFT para todos os pontos e armazenar o resultado em
        fftResult
204     std::cout << "Executando a FFT para o fluido" << i + 1 << "." <<
        std::endl;
205     std::vector<std::complex<double>> fftResult = fftProcessor.
        forward(complexInput);
206     std::cout << "FFT para o fluido" << i + 1 << " concluida." <<
        std::endl;
207
208     // Normalizar o resultado da FFT dividindo pelo número de pontos
209     for (auto& val : fftResult) {
210         val /= static_cast<double>(numPoints);
211     }
212
213     // Atualizar o vetor vfreq com os resultados para cada ponto
214     if (vfreq[i].size() != numPoints) {
215         vfreq[i].resize(numPoints);
216     }
217     for (int j = 0; j < numPoints; ++j) {
218         vfreq[i][j] = fftResult[j]; // Armazena o resultado da FFT no
        vetor de frequencia
219     }
220

```

```

221 // Verificar se o resultado da FFT nao esta zerado
222 double maxFFT = std::abs(*std::max_element(vfreq[i].begin(),
223     vfreq[i].end(),
224     [](const std::complex<double>& a, const std::complex<double>&
225         b) {
226         return std::abs(a) < std::abs(b);
227     }));
228 std::cout << "Maxima amplitude da FFT para o fluido" << i + 1 <<
229     ":" << maxFFT << std::endl;
230 if (maxFFT < 1e-6) {
231     std::cerr << "Aviso: o resultado da FFT para o fluido" << i
232         + 1 << " tem amplitude muito baixa." << std::endl;
233 }
234
235 std::cout << "FFT para o fluido" << i + 1 << " concluida com
236     sucesso." << std::endl;
237
238 } catch (const std::exception& e) {
239     std::cerr << "Erro ao realizar a FFT para o fluido" << i + 1 <<
240         ":" << e.what() << std::endl;
241 }
242
243 // Mensagem de depuracao para verificar se o loop continua
244 std::cout << "Continuando para o proximo fluido." << std::endl;
245 }
246 std::cout << "FFT para todos os fluidos concluida." << std::endl;
247 }
248
249 void SeismicSimulator::performInverseFFT() {
250     std::cout << "Realizando a IFFT para cada fluido." << std::endl;
251
252     for (size_t i = 0; i < fluids.size(); ++i) {
253         if (vfreq[i].empty() || wfreq[i].empty()) {
254             std::cerr << "Aviso: vfreq ou wfreq vazio para o fluido" << i +
255                 1 << ", pulando IFFT." << std::endl;
256             continue;
257         }
258
259         // Realiza a IFFT usando a classe FFTWrapper para vfreq e wfreq
260         std::vector<std::complex<double>> inverseVResult = fftProcessor.
261             inverse(vfreq[i]);
262         std::vector<std::complex<double>> inverseWResult = fftProcessor.

```

```

        inverse(wfreq[i]);
256
257 // Multiplica por (4 * pi) / dt para normalizar
258 double dt = systemParams.getTotalTime() / (numPoints - 1);
259 double normalizationFactor = (4.0 * M_PI) / dt;
260 for (size_t j = 0; j < inverseVResult.size(); ++j) {
261     inverseVResult[j] *= normalizationFactor / static_cast<double>(
        numPoints);
262     inverseWResult[j] *= normalizationFactor / static_cast<double>(
        numPoints);
263 }
264
265 // Centraliza os dados em torno de zero (subtraindo a média)
266 double meanV = std::accumulate(inverseVResult.begin(), inverseVResult
    .end(), 0.0,
267
        [](double sum, const std::complex<
            double>& val) {
268             return sum + val.real();
269         }) / numPoints;
270 double meanW = std::accumulate(inverseWResult.begin(), inverseWResult
    .end(), 0.0,
271
        [](double sum, const std::complex<
            double>& val) {
272             return sum + val.real();
273         }) / numPoints;
274 for (auto& val : inverseVResult) {
275     val -= meanV;
276 }
277 for (auto& val : inverseWResult) {
278     val -= meanW;
279 }
280
281 // Atualiza os vetores v e x com os valores reais da transformada
    inversa e inverte a ordem
282 for (int j = 0; j < numPoints; ++j) {
283     v[i][j] = inverseVResult[numPoints - 1 - j].real();
284     x[i][j] = inverseWResult[numPoints - 1 - j].real();
285 }
286
287 std::cout << "IFFT para o fluido " << i + 1 << " concluída com
    sucesso." << std::endl;
288 }
289 std::cout << "IFFT para todos os fluidos concluída." << std::endl;

```

```

290 }
291
292
293
294 void SeismicSimulator::normalizeResults() {
295     // Calcular o valor maximo de vmax2 e xmax2
296     double vmax2 = 0.0;
297     double xmax2 = 0.0;
298
299     std::cout << "fluidos:_" << fluids.size() << "_e_" << vmax2 << std::endl;
300
301     // Encontrar o maximo valor de v e x para normalizacao
302     for (size_t i = 0; i < fluids.size(); ++i) {
303         double currentVMax = *std::max_element(v[i].begin(), v[i].end(),
304             [](double a, double b) { return std::abs(a) < std::abs(b); });
305         double currentXMax = *std::max_element(x[i].begin(), x[i].end(),
306             [](double a, double b) { return std::abs(a) < std::abs(b); });
307
308         vmax2 = std::max(vmax2, currentVMax);
309         xmax2 = std::max(xmax2, currentXMax);
310         std::cout << "maxX_e_maxY:_" << xmax2 << "_e_" << vmax2 << std::endl;
311     }
312
313     // Normaliza os resultados para cada fluido
314     for (size_t i = 0; i < fluids.size(); ++i) {
315         if (vmax2 > 0) { // Evitar divisao por zero
316             for (size_t j = 0; j < v[i].size(); ++j) {
317                 v[i][j] /= vmax2; // Normalizar v
318             }
319         }
320
321         if (xmax2 > 0) { // Evitar divisao por zero
322             for (size_t j = 0; j < x[i].size(); ++j) {
323                 x[i][j] /= xmax2; // Normalizar x
324             }
325         }
326     }
327 }
328
329 void SeismicSimulator::constructFrequencySolution() {
330     std::cout << "Construindo_a_solucao_no_dominio_da_frequencia." << std::
        endl;
331

```

```

332     double ezero = 8.85e-12;
333     double F = rock.getTortuosity() / rock.getPorosity();
334     double wd = 2 * M_PI * dominantFrequency;
335     double z = systemParams.getDepth();
336     double ni = sqrt(8 * rock.getAbsolutePermeability() * F);
337     double sigmab = fluids[0].getConductivity();
338     double m = 8.0;
339
340     // Preparar variaveis para calculo
341     std::vector<double> rho(fluids.size());
342     std::vector<double> omegat(fluids.size());
343     std::vector<double> Lzero(fluids.size());
344
345     for (size_t k = 0; k < fluids.size(); ++k) {
346         rho[k] = (1.0 - rock.getPorosity()) * rock.getDensity() + rock.
            getPorosity() * fluids[k].getDensities()[0];
347         omegat[k] = fluids[k].getViscosities()[0] / (F * rock.
            getAbsolutePermeability() * fluids[k].getDensities()[0]);
348         Lzero[k] = -(1.0 / F) * (ezero * fluids[k].getDielectricConstant() *
            fluids[k].getDielectricConstant() / fluids[k].getViscosities()[0])
            ;
349     }
350
351     // Calcular o vetor de frequencias conforme o MATLAB
352     double T = systemParams.getTotalTime();
353     double df = 1.0 / T;
354     std::vector<double> f(numPoints);
355     for (int j = 0; j < numPoints; ++j) {
356         f[j] = (0.0001 + j * 0.5) * df;
357     }
358
359     // Vetor de frequencias angulares
360     std::vector<std::complex<double>> w(numPoints);
361     std::complex<double> delta = std::complex<double>(0, M_PI / T);
362     for (int j = 0; j < numPoints; ++j) {
363         w[j] = 2.0 * M_PI * f[j] + delta;
364     }
365
366     for (size_t i = 0; i < fluids.size(); ++i) {
367         std::cout << "Construindo soluc o para o fluido " << i + 1 << "..."
            << std::endl;
368
369         for (int j = 0; j < numPoints; ++j) {

```

```

370         std::complex<double> wj = w[j];
371
372         // Calcular o valor de h[j] com base nas condicoes de wj
373         if (wj == wd) {
374             h[j] = -1.0i * M_PI / wd
375                 - (21.0 / 16.0) * wd * ((std::exp(-2.0i * M_PI * wj / wd)
376                     - 1.0) / (wj * wj - 4.0 * wd * wd))
377                 + (21.0 / 64.0) * wd * ((std::exp(-2.0i * M_PI * wj / wd)
378                     - 1.0) / (wj * wj - 16.0 * wd * wd))
379                 - (1.0 / 64.0) * wd * ((std::exp(-2.0i * M_PI * wj / wd)
380                     - 1.0) / (wj * wj - 64.0 * wd * wd));
381         } else if (wj == 2.0 * wd) {
382             h[j] = wd * ((std::exp(-2.0i * M_PI * wj / wd) - 1.0) / (wj *
383                 wj - wd * wd))
384                 + 21.0i * M_PI / (32.0 * wd)
385                 + (21.0 / 64.0) * wd * ((std::exp(-2.0i * M_PI * wj / wd)
386                     - 1.0) / (wj * wj - 16.0 * wd * wd))
387                 - (1.0 / 64.0) * wd * ((std::exp(-2.0i * M_PI * wj / wd)
388                     - 1.0) / (wj * wj - 64.0 * wd * wd));
389         } else if (wj == 4.0 * wd) {
390             h[j] = wd * ((std::exp(-2.0i * M_PI * wj / wd) - 1.0) / (wj *
391                 wj - wd * wd))
392                 - (21.0 / 16.0) * wd * ((std::exp(-2.0i * M_PI * wj / wd)
393                     - 1.0) / (wj * wj - 4.0 * wd * wd))
394                 - 21.0i * M_PI / (256.0 * wd)
395                 - (1.0 / 64.0) * wd * ((std::exp(-2.0i * M_PI * wj / wd)
396                     - 1.0) / (wj * wj - 64.0 * wd * wd));
397         } else if (wj == 8.0 * wd) {
398             h[j] = wd * ((std::exp(-2.0i * M_PI * wj / wd) - 1.0) / (wj *
399                 wj - wd * wd))
400                 - (21.0 / 16.0) * wd * ((std::exp(-2.0i * M_PI * wj / wd)
401                     - 1.0) / (wj * wj - 4.0 * wd * wd))
402                 + (21.0 / 64.0) * wd * ((std::exp(-2.0i * M_PI * wj / wd)
403                     - 1.0) / (wj * wj - 16.0 * wd * wd))
404                 + 1.0i * M_PI / (512.0 * wd);
405         } else {
406             h[j] = wd * ((std::exp(-2.0i * M_PI * wj / wd) - 1.0) / (wj *
407                 wj - wd * wd))
408                 - (21.0 / 16.0) * wd * ((std::exp(-2.0i * M_PI * wj / wd)
409                     - 1.0) / (wj * wj - 4.0 * wd * wd))
410                 + (21.0 / 64.0) * wd * ((std::exp(-2.0i * M_PI * wj / wd)
411                     - 1.0) / (wj * wj - 16.0 * wd * wd))
412                 - (1.0 / 64.0) * wd * ((std::exp(-2.0i * M_PI * wj / wd)

```



```

- 1.0) / (wj * wj - 64.0 * wd * wd));
398     }
399
400     // Calculo de kappa (permeabilidade dinamica)
401     std::complex<double> kappa = rock.getAbsolutePermeability() * std
        ::pow(1.0 - (1.0i * wj / omegat[i]) * 4.0 / m, 0.5) - (1.0i *
        wj / omegat[i]);
402
403     // Calculo de L (coeficiente de acoplamento)
404     std::complex<double> L = Lzero[i] * std::pow(1.0 - (1.0i * (wj /
        omegat[i]) * (m / 4.0)) * std::pow(1.0 - 2.0 * z / ni, 2.0) *
        std::pow(1.0 - 1.0i * std::sqrt(3.0 / 2.0) * z * std::sqrt(wj
        * fluids[i].getDensities()[0] / fluids[i].getViscosities()[0])
        , 2.0), -0.5);
405
406     // Calculo de Q (termo simplificador)
407     std::complex<double> Q = -fluids[i].getViscosities()[0] / (1.0i *
        wj * kappa * (1.0 - (std::pow(L, 2.0) * fluids[i].
        getViscosities()[0]) / (kappa * sigmab)));
408
409     // Parametros para o calculo dos autovalores
410     double C = rock.getBiotCoefficient();
411     double M = rock.getBiotModulus();
412     double lamb = rock.getBulkModulus();
413     double G = rock.getShearModulus();
414     double beta = 1.0 / (C * C - M * (lamb + 2.0 * G));
415
416     // Calculo dos autovalores q1 e q2
417     std::complex<double> q1quad = (beta / 2.0) * ((2.0 * C * fluids[i]
        ].getDensities()[0] - M * rho[i] - Q * (lamb + 2.0 * G)) +
418         std::sqrt(std::pow(Q * (lamb + 2.0
        * G) - M * rho[i], 2.0) -
419         4.0 * ((M * fluids[i].
        getDensities()[0] - C
        * Q) * (C * rho[i] -
        (lamb + 2.0 * G) *
        fluids[i].
        getDensities()[0]))))
        ;
420     std::complex<double> q2quad = (beta / 2.0) * ((2.0 * C * fluids[i]
        ].getDensities()[0] - M * rho[i] - Q * (lamb + 2.0 * G)) -
421         std::sqrt(std::pow(Q * (lamb + 2.0
        * G) - M * rho[i], 2.0) -

```

```

422                                     4.0 * ((M * fluids[i].
                                     getDensities()[0] - C
                                     * Q) * (C * rho[i] -
                                     (lamb + 2.0 * G) *
                                     fluids[i].
                                     getDensities()[0]))))
                                     ;
423 std::complex<double> q1 = std::sqrt(q1quad);
424 std::complex<double> q2 = std::sqrt(q2quad);
425
426 // Calculo dos autovalores e autovetores
427 std::complex<double> a1 = sqrt((M * fluids[i].getDensities()[0] *
    M * fluids[i].getDensities()[0] - 2.0 * M * fluids[i].
    getDensities()[0] * C * Q + C * C * Q * Q) /
428 (M * M * (fluids[i].getDensities()[0] * fluids[i].
    getDensities()[0] + rho[i] * rho[i]) + C * C * (fluids[i].
    getDensities()[0] * fluids[i].getDensities()[0] + Q * Q) -
429 2.0 * M * fluids[i].getDensities()[0] * C * (rho[i] + Q) -
430 2.0 * (fluids[i].getDensities()[0] * C - M * rho[i]) * ((q1 *
    q1) / beta) +
431 pow((q1 * q1) / beta, 2.0)));
432
433 std::complex<double> a2 = sqrt((M * fluids[i].getDensities()[0] *
    M * fluids[i].getDensities()[0] - 2.0 * M * fluids[i].
    getDensities()[0] * C * Q + C * C * Q * Q) /
434 (M * M * (fluids[i].getDensities()[0] * fluids[i].
    getDensities()[0] + rho[i] * rho[i]) + C * C * (fluids[i].
    getDensities()[0] * fluids[i].getDensities()[0] + Q * Q) -
435 2.0 * M * fluids[i].getDensities()[0] * C * (rho[i] + Q) -
436 2.0 * (fluids[i].getDensities()[0] * C - M * rho[i]) * ((q2 *
    q2) / beta) +
437 pow((q2 * q2) / beta, 2.0)));
438
439 // Calculo dos parametros xi1 e xi2
440 std::complex<double> xi1 = (C * fluids[i].getDensities()[0] - M *
    rho[i] - (q1 * q1) / beta) / (M * fluids[i].getDensities()[0]
    - C * Q);
441 std::complex<double> xi2 = (C * fluids[i].getDensities()[0] - M *
    rho[i] - (q2 * q2) / beta) / (M * fluids[i].getDensities()[0]
    - C * Q);
442
443 // Calculo dos parametros b11, b12, b21, b22
444 b11[i] = -a1;

```

```

445         b21[i] = a1 * xi1;
446         b12[i] = -a2;
447         b22[i] = a2 * xi2;
448
449         // Calcular os autovetores y11, y12, y21, y22
450         std::complex<double> y11 = (a1 / q1) * (-rho[i] - fluids[i].
            getDensities()[0] * xi1);
451         std::complex<double> y21 = (a1 / q1) * (fluids[i].getDensities()
            [0] + Q * xi1);
452         std::complex<double> y12 = (a2 / q2) * (-rho[i] - fluids[i].
            getDensities()[0] * xi2);
453         std::complex<double> y22 = (a2 / q2) * (fluids[i].getDensities()
            [0] + Q * xi2);
454
455         // Calculo de vfreq e wfreq utilizando os autovalores e
            autovetores
456         vfreq[i][j] = h[j] * (y11 * (b11[i] - b21[i]) * exp(1i * wj * q1
            * z) + y12 * (b12[i] - b22[i]) * exp(1i * wj * q2 * z));
457         wfreq[i][j] = -h[j] * (y21 * (b11[i] - b21[i]) * exp(1i * wj * q1
            * z) + y22 * (b12[i] - b22[i]) * exp(1i * wj * q2 * z));
458     }
459
460     std::cout << "Solucao para fluido " << i + 1 << " construida com
        sucesso." << std::endl;
461 }
462 }
463
464
465 void SeismicSimulator::plotResults(const std::string& gnuplotPath) {
466     GnuplotPlotter plotter(gnuplotPath);
467     plotter.plotResults(v, x, timeVector);
468     std::cout << "Resultados plotados com sucesso." << std::endl;
469 }
470
471 std::vector<double> SeismicSimulator::generateRickerWavelet(int numPoints,
    double dt, double dominantFrequency) {
472     // Parametros para a onda Ricker
473     double piSquared = M_PI * M_PI;
474     double fSquared = dominantFrequency * dominantFrequency;
475     double t0 = (numPoints - 1) * dt / 2.0; // Centralizar a onda Ricker no
        meio do intervalo de tempo
476
477     std::vector<double> wavelet(numPoints);

```

```

478
479     for (int i = 0; i < numPoints; ++i) {
480         double t = i * dt - t0; // Centraliza o tempo em torno de zero
481         double term = piSquared * fSquared * t * t;
482         wavelet[i] = (1.0 - 2.0 * term) * std::exp(-term);
483     }
484
485     return wavelet;
486 }

```

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

Listing 6.4: Arquivo de cabeçalho da classe eigenValueSolver.

```

1 // EigenvalueSolver.h
2 #ifndef EIGENVALUE_SOLVER_H
3 #define EIGENVALUE_SOLVER_H
4
5 #include <Eigen/Dense>
6 #include <vector>
7
8 class EigenvalueSolver {
9 public:
10     // Construtor
11     EigenvalueSolver();
12
13     // Define a matriz para calcular os autovalores e autovetores
14     void setMatrix(const Eigen::MatrixXd& matrix);
15
16     // Calcula os autovalores e autovetores
17     bool compute();
18
19     // Obtém os autovalores
20     Eigen::VectorXd getEigenvalues() const;
21
22     // Obtém os autovetores
23     Eigen::MatrixXd getEigenvectors() const;
24
25 private:
26     Eigen::MatrixXd m_matrix; // Matriz de entrada
27     Eigen::VectorXd m_eigenvalues; // Autovalores calculados
28     Eigen::MatrixXd m_eigenvectors; // Autovetores calculados
29     bool m_isComputed; // Indica se os autovalores e autovetores foram
        calculados
30 };

```

31

32 `#endif // EIGENVALUE_SOLVER_H`

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

Listing 6.5: Arquivo de implementação da classe `eigenValueSolver`.

```

1 // EigenvalueSolver.cpp
2 #include "eigenValueSolver.h"
3 #include <Eigen/Eigenvalues>
4 #include <iostream>
5
6
7 // Construtor
8 EigenvalueSolver::EigenvalueSolver() : m_isComputed(false) {}
9
10 // Define a matriz
11 void EigenvalueSolver::setMatrix(const Eigen::MatrixXd& matrix) {
12     m_matrix = matrix;
13     m_isComputed = false; // Marca como não computado
14 }
15
16 // Calcula os autovalores e autovetores
17 bool EigenvalueSolver::compute() {
18     if (m_matrix.size() == 0) {
19         std::cerr << "A matriz não foi definida." << std::endl;
20         return false;
21     }
22
23     // Calcula os autovalores e autovetores
24     Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> solver(m_matrix);
25     if (solver.info() != Eigen::Success) {
26         std::cerr << "Falha ao calcular os autovalores." << std::endl;
27         return false;
28     }
29
30     // Armazena os resultados
31     m_eigenvalues = solver.eigenvalues();
32     m_eigenvectors = solver.eigenvectors();
33     m_isComputed = true;
34
35     return true;
36 }
37
38 // Retorna os autovalores

```

```

39 Eigen::VectorXd EigenvalueSolver::getEigenvalues() const {
40     if (!m_isComputed) {
41         std::cerr << "Os autovalores não foram calculados." << std::endl;
42     }
43     return m_eigenvalues;
44 }
45
46 // Retorna os autovetores
47 Eigen::MatrixXd EigenvalueSolver::getEigenvectors() const {
48     if (!m_isComputed) {
49         std::cerr << "Os autovetores não foram calculados." << std::endl;
50     }
51     return m_eigenvectors;
52 }

```

Apresenta-se na listagem 6.6 o arquivo com o código da classe fft.

Listing 6.6: Arquivo de cabeçalho da classe fft.

```

1 // FFTWrapper.h
2 #ifndef FFTWRAPPER_H
3 #define FFTWRAPPER_H
4
5 #include <vector>
6 #include <complex>
7 #include <algorithm>
8 #include <iostream>
9 #include <iterator>
10 #include <stdexcept>
11
12 #include <fftw3/fftw3.h>
13
14 class FFTWrapper {
15 public:
16     FFTWrapper();
17     FFTWrapper(int size);
18     ~FFTWrapper();
19
20     // Configura o tamanho da FFT
21     void setSize(int n);
22
23     // Executa a transformada de Fourier direta
24     std::vector<std::complex<double>> forward(const std::vector<std::complex<
        double>>& input);
25

```

```

26 // Executa a transformada de Fourier inversa
27 std::vector<std::complex<double>> inverse(const std::vector<std::complex<
    double>>& input);
28
29 // Verifica se a FFT foi inicializada corretamente
30 bool isInitialized() const;
31
32 private:
33     int size; // Tamanho da transformada
34     fftw_complex *in, *out;
35     fftw_plan plan_forward, plan_backward;
36
37     // Funcoes internas para inicializar e liberar memoria
38     void initialize();
39     void cleanup();
40
41     // Funcoes auxiliares para simular o comportamento do MATLAB
42     std::vector<std::complex<double>> fftshift(const std::vector<std::complex<
        double>>& input);
43     std::vector<std::complex<double>> ifftshift(const std::vector<std::
        complex<double>>& input);
44 };
45
46 #endif // FFTWRAPPER_H

```

Apresenta-se na listagem 6.7 o arquivo com o código da implementação da classe `fft`.

Listing 6.7: Arquivo de implementação da classe `fft`.

```

1 // FFTWrapper.cpp
2 #include "fft.h"
3 #include <stdexcept>
4
5 FFTWrapper::FFTWrapper()
6 {
7     initialize();
8 }
9
10 FFTWrapper::FFTWrapper(int size) : size(size), in(nullptr), out(nullptr)
11 {
12     initialize();
13 }
14
15 FFTWrapper::~FFTWrapper() {
16     cleanup();

```

```

17 }
18
19 void FFTWrapper::setSize(int n) {
20     size = n;
21 }
22
23 void FFTWrapper::initialize() {
24     if (size <= 0) {
25         throw std::runtime_error("Tamanho inválido para a FFT.");
26     }
27     std::cout << "Iniciando FFTWrapper com tamanho: " << size << std::
        endl;
28
29     // Não chame cleanup no início. Em vez disso, verifique se os recursos
        estão alocados.
30     // Alocação de memória para FFT
31     in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * size);
32     out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * size);
33
34     if (!in || !out) {
35         std::cerr << "Falha ao alocar memória para FFTW." << std::endl;
36         throw std::runtime_error("Falha ao alocar memória para FFTW.");
37     }
38
39     // Criação dos planos FFT
40     std::cout << "Criando planos FFT..." << std::endl;
41     plan_forward = fftw_plan_dft_1d(size, in, out, FFTW_FORWARD,
        FFTW_ESTIMATE);
42     plan_backward = fftw_plan_dft_1d(size, in, out, FFTW_BACKWARD,
        FFTW_ESTIMATE);
43
44     if (!plan_forward) {
45         std::cerr << "Falha ao criar plano FFTW_FORWARD." << std::endl;
46         cleanup(); // Libere os recursos já alocados
47         throw std::runtime_error("Falha ao criar plano FFTW_FORWARD.");
48     }
49
50     if (!plan_backward) {
51         std::cerr << "Falha ao criar plano FFTW_BACKWARD." << std::endl;
52         cleanup(); // Libere os recursos já alocados
53         throw std::runtime_error("Falha ao criar plano FFTW_BACKWARD.");
54     }
55

```



```

56     std::cout << "FFTWrapper_inicializado_com_sucesso." << std::endl;
57 }
58 void FFTWrapper::cleanup() {
59     std::cout << "Iniciando_cleanup." << std::endl;
60     if (plan_forward) {
61         fftw_destroy_plan(plan_forward);
62         plan_forward = nullptr; // Evitar uso posterior
63         std::cout << "Plano_FFTW_forward_destruído." << std::endl;
64     }
65     if (plan_backward) {
66         fftw_destroy_plan(plan_backward);
67         plan_backward = nullptr; // Evitar uso posterior
68         std::cout << "Plano_FFTW_backward_destruído." << std::endl;
69     }
70     if (in) {
71         fftw_free(in);
72         in = nullptr; // Evitar uso posterior
73         std::cout << "Memoria_de_entrada_FFTW_liberada." << std::endl;
74     }
75     if (out) {
76         fftw_free(out);
77         out = nullptr; // Evitar uso posterior
78         std::cout << "Memoria_de_saída_FFTW_liberada." << std::endl;
79     }
80     std::cout << "Cleanup_concluído." << std::endl;
81 }
82
83 std::vector<std::complex<double>> FFTWrapper::forward(const std::vector<std::
    complex<double>>& input) {
84     if (input.size() != size) {
85         throw std::invalid_argument("O_tamanho_do_vetor_de_entrada_nao_
            corresponde_ao_tamanho_da_FFT.");
86     }
87
88     // Copia os dados de entrada para o array 'in'
89     for (int i = 0; i < size; ++i) {
90         in[i][0] = input[i].real();
91         in[i][1] = input[i].imag();
92     }
93
94     // Executa a transformada direta
95     fftw_execute(plan_forward);
96

```

```

97     // Copia os resultados para um vetor de saída
98     std::vector<std::complex<double>> output(size);
99     for (int i = 0; i < size; ++i) {
100         output[i] = std::complex<double>(out[i][0], out[i][1]);
101     }
102
103     // Aplica fftshift para reorganizar as frequências como no MATLAB
104     output = fftshift(output);
105
106     // Normaliza o resultado como é feito no MATLAB, dividindo pelo tamanho
107     for (auto& val : output) {
108         val /= static_cast<double>(size);
109     }
110
111     return output;
112 }
113
114
115 std::vector<std::complex<double>> FFTWrapper::inverse(const std::vector<std::
    complex<double>>& input) {
116     if (!isInitialized()) {
117         throw std::runtime_error("FFTWrapper não foi inicializado
            corretamente.");
118     }
119
120     if (input.size() != size) {
121         throw std::invalid_argument("O tamanho do vetor de entrada não
            corresponde ao tamanho da FFT.");
122     }
123
124     // Aplica ifftshift ao vetor de entrada antes da transformada inversa
125     std::vector<std::complex<double>> shiftedInput = ifftshift(input);
126
127     // Copia os dados de entrada (shiftedInput) para o array 'in'
128     for (int i = 0; i < size; ++i) {
129         in[i][0] = shiftedInput[i].real();
130         in[i][1] = shiftedInput[i].imag();
131     }
132
133     // Executa a transformada inversa
134     fftw_execute(plan_backward);
135
136     // Copia os resultados para um vetor de saída

```

```

137     std::vector<std::complex<double>> output(size);
138     for (int i = 0; i < size; ++i) {
139         output[i] = std::complex<double>(out[i][0], out[i][1]);
140     }
141
142     // Aplica ifftshift ao vetor de saída para reorganizar as frequências
143     output = ifftshift(output);
144
145     return output;
146 }
147
148
149 // Funcoes auxiliares fftshift e ifftshift
150 std::vector<std::complex<double>> FFTWrapper::fftshift(const std::vector<std
::complex<double>>& input) {
151     std::vector<std::complex<double>> output(size);
152     int mid = size / 2;
153     for (int i = 0; i < size; ++i) {
154         int shiftedIndex = (i + mid) % size;
155         output[i] = input[shiftedIndex];
156     }
157     return output;
158 }
159
160 std::vector<std::complex<double>> FFTWrapper::ifftshift(const std::vector<std
::complex<double>>& input) {
161     std::vector<std::complex<double>> output(size);
162     int mid = size / 2;
163     // Desloca os elementos para a esquerda, para que o centro seja movido
    para a primeira posicao
164     for (int i = 0; i < size; ++i) {
165         int shiftedIndex = (i + mid) % size;
166         output[i] = input[shiftedIndex];
167     }
168     return output;
169 }
170
171 bool FFTWrapper::isInitialized() const {
172     // Verifica se os planos foram criados com sucesso
173     return plan_forward != nullptr && plan_backward != nullptr && in !=
    nullptr && out != nullptr;
174 }

```

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

Listing 6.8: Arquivo de cabeçalho da classe fluidProperties.

```

1 #ifndef FLUID_PROPERTIES_H
2 #define FLUID_PROPERTIES_H
3
4 #include <vector>
5
6 class FluidProperties {
7 public:
8     FluidProperties(const std::vector<double>& densities, const std::vector<
9         double>& viscosities,
10         double dielectricConstant, double zetaPotential, double
11             conductivity)
12         : densities(densities), viscosities(viscosities), dielectricConstant(
13             dielectricConstant),
14             zetaPotential(zetaPotential), conductivity(conductivity) {}
15
16     std::vector<double> getDensities() const { return densities; }
17     std::vector<double> getViscosities() const { return viscosities; }
18     double getDielectricConstant() const { return dielectricConstant; }
19     double getZetaPotential() const { return zetaPotential; }
20     double getConductivity() const { return conductivity; }
21
22 private:
23     std::vector<double> densities;
24     std::vector<double> viscosities;
25     double dielectricConstant;
26     double zetaPotential;
27     double conductivity;
28 };
29
30 #endif // FLUID_PROPERTIES_H

```

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

Listing 6.9: Arquivo de cabeçalho da classe rockProperties.

```

1 #ifndef ROCK_PROPERTIES_H
2 #define ROCK_PROPERTIES_H
3
4 class RockProperties {
5 public:
6     RockProperties(double density, double shearModulus, double porosity,
7         double bulkModulus,
8         double tortuosity, double biotCoefficient, double
9             biotModulus, double permeability)

```

```

8      : density(density), shearModulus(shearModulus), porosity(porosity),
      bulkModulus(bulkModulus),
9      tortuosity(tortuosity), biotCoefficient(biotCoefficient),
      biotModulus(biotModulus),
10     permeability(permeability) {}
11
12     double getDensity() const { return density; }
13     double getShearModulus() const { return shearModulus; }
14     double getPorosity() const { return porosity; }
15     double getBulkModulus() const { return bulkModulus; }
16     double getTortuosity() const { return tortuosity; }
17     double getBiotCoefficient() const { return biotCoefficient; }
18     double getBiotModulus() const { return biotModulus; }
19     double getAbsolutePermeability() const { return permeability; }
20
21 private:
22     double density;
23     double shearModulus;
24     double porosity;
25     double bulkModulus;
26     double tortuosity;
27     double biotCoefficient;
28     double biotModulus;
29     double permeability;
30 };
31
32 #endif // ROCK_PROPERTIES_H

```

Apresenta-se na listagem 6.10 o arquivo com o código da classe systemParameters.

Listing 6.10: Arquivo de cabeçalho da classe systemParameters.

```

1 // SystemParameters.h
2 #ifndef SYSTEM_PARAMETERS_H
3 #define SYSTEM_PARAMETERS_H
4
5 class SystemParameters {
6 public:
7
8     SystemParameters(double dominantFrequency, double depth, double totalTime
9         , int numPoints,
10         double debyeLength, double electricalPermittivity)
11     : dominantFrequency(dominantFrequency), depth(depth), totalTime(
12         totalTime), numPoints(numPoints),
13         debyeLength(debyeLength), electricalPermittivity(

```

```

    electricalPermittivity) {}

12
13 // Getters
14 double getDominantFrequency() const { return dominantFrequency; }
15 double getDepth() const { return depth; }
16 double getTotalTime() const { return totalTime; }
17 int getNumPoints() const { return numPoints; }
18 double getDebyeLength() const { return debyeLength; }
19 double getElectricalPermittivity() const { return electricalPermittivity;
    }
20
21 // Setters
22 void setDominantFrequency(double value) { dominantFrequency = value; }
23 void setDepth(double value) { depth = value; }
24 void setTotalTime(double value) { totalTime = value; }
25 void setNumPoints(int value) { numPoints = value; }
26 void setDebyeLength(double value) { debyeLength = value; }
27 void setElectricalPermittivity(double value) { electricalPermittivity =
    value; }
28
29 private:
30 double dominantFrequency; // Frequência dominante [Hz]
31 double depth; // Profundidade da fonte [m]
32 double totalTime; // Tempo total da simulação [s]
33 int numPoints; // Número de pontos de discretização
34 double debyeLength; // Comprimento de Debye
35 double electricalPermittivity; // Permissividade elétrica do meio
36 };
37
38 #endif // SYSTEM_PARAMETERS_H

```

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

Listing 6.11: Arquivo de cabeçalho da classe gnuPlotter.

```

1 // GnuplotPlotter.h
2 #ifndef GNUPLOT_PLOTTER_H
3 #define GNUPLOT_PLOTTER_H
4
5 #include <string>
6 #include <vector>
7 #include <iostream>
8 #include <fstream>
9 #include <cstdlib>
10

```

```

11 class GnuplotPlotter {
12 public:
13     GnuplotPlotter(const std::string& gnuplotPath = "c:/gnuplot");
14     void plotResults(const std::vector<std::vector<double>>& resultsV,
15                     const std::vector<std::vector<double>>& resultsX,
16                     const std::vector<double>& timeVector);
17
18 private:
19     std::string gnuplotPath;
20     std::string tempDataFileX;
21     void writeDataToFile(const std::vector<std::vector<double>>& results,
22                         const std::vector<double>& timeVector,
23                         const std::string& fileName) ;
24 };
25
26 #endif // GNUPLOT_PLOTTER_H

```

Apresenta-se na listagem 6.12 o arquivo com o código da implementação da classe `gnuPlotter`.

Listing 6.12: Arquivo de implementação da classe `gnuPlotter`.

```

1 // GnuplotPlotter.cpp
2 #include "gnuPlotter.h"
3
4 // Para portabilidade entre sistemas (Windows vs. Linux/macOS)
5 #ifdef _WIN32
6 #define popen _popen
7 #define pclose _pclose
8 #endif
9
10 GnuplotPlotter::GnuplotPlotter(const std::string& gnuplotPath)
11     : gnuplotPath(gnuplotPath) {
12     // A variável tempDataFileX membro não é mais usada,
13     // pois os nomes dos arquivos temporários são definidos localmente
14     // dentro de plotResults para 'v' e 'x'.
15     // Portanto, não há necessidade de inicializá-la aqui.
16 }
17
18 void GnuplotPlotter::plotResults(const std::vector<std::vector<double>>&
19                                 resultsV,
20
21                                 const std::vector<std::vector<double>>&
22                                 resultsX,
23
24                                 const std::vector<double>& timeVector) {
25     // Escreve os dados em arquivos temporários separados para 'v' e 'x'

```

```

22     std::string tempDataFileV = "temp_data_v.txt";
23     std::string tempDataFileX = "temp_data_x.txt";
24     writeDataToFile(resultsV, timeVector, tempDataFileV);
25     writeDataToFile(resultsX, timeVector, tempDataFileX);
26
27     // Configura o comando para o Gnuplot
28     // Usando popen/pclose para compatibilidade
29     std::string command = "\"" + gnuplotPath + "/bin/gnuplot.exe\" -persist";
30     FILE* gnuplotPipe = popen(command.c_str(), "w");
31     if (!gnuplotPipe) {
32         std::cerr << "Erro ao abrir o pipe para o Gnuplot." << std::endl;
33         return;
34     }
35
36     // Configurações comuns para Gnuplot
37     const char* gnuplotSettings = R"(
38     set key outside
39     unset ytics
40     set grid
41     set encoding utf8
42     set xlabel 'Tempo (s)'
43     set ylabel 'Amplitude Normalizada'
44     set tics font ',16'
45     set title font ',18'
46     set term wxt size 800,600
47     )";
48     fprintf(gnuplotPipe, "%s\n", gnuplotSettings);
49
50     // Plot para 'x' (Velocidade do fluido) na primeira janela
51     fprintf(gnuplotPipe, "set term wxt 0\n"); // Abre a primeira janela
52     fprintf(gnuplotPipe, "set title 'Velocidade da Fase Fluida'\n");
53     fprintf(gnuplotPipe, "plot '%s' using 1:2 with lines title 'Agua' lc rgb \
        'blue',\n", tempDataFileX.c_str());
54     fprintf(gnuplotPipe, "'' using 1:($3+2) with lines title 'Oleo \
        Leve' lc rgb 'green',\n", tempDataFileX.c_str());
55     fprintf(gnuplotPipe, "'' using 1:($4+4) with lines title 'Oleo \
        Medio' lc rgb 'red',\n", tempDataFileX.c_str());
56     fprintf(gnuplotPipe, "'' using 1:($5+6) with lines title 'Oleo \
        Pesado' lc rgb 'black'\n", tempDataFileX.c_str());
57     fflush(gnuplotPipe); // Garante que os comandos para a primeira janela
        são enviados
58
59

```



```

60
61 // Fecha o pipe do Gnuplot
62 fflush(gnuplotPipe); // Garante que os comandos para a segunda janela
    são enviados
63 fclose(gnuplotPipe); // Usa fclose para compatibilidade
64
65 // Remover os arquivos temporários
66 //std::remove(tempDataFileV.c_str());
67 //std::remove(tempDataFileX.c_str());
68 }
69
70 void GnuplotPlotter::writeDataToFile(const std::vector<std::vector<double>>&
    results,
71                                     const std::vector<double>& timeVector,
72                                     const std::string& fileName) {
73     std::ofstream file(fileName);
74     if (!file.is_open()) {
75         std::cerr << "Erro: Não foi possível abrir o arquivo " << fileName
            << " para escrita." << std::endl;
76         return;
77     }
78
79     // Assumimos que results[0] corresponde à coluna de dados para
        timeVector
80     // e as colunas subsequentes são os dados para cada fluido.
81     // O formato esperado pelo gnuplot é: Tempo Fluido1 Fluido2 ...
82     // timeVector.size() deve ser igual a results[i].size() para todos i.
83     if (timeVector.size() != results[0].size()) {
84         std::cerr << "Erro: Tamanho do vetor de tempo não corresponde ao
            tamanho dos resultados." << std::endl;
85         return;
86     }
87
88     for (size_t i = 0; i < timeVector.size(); ++i) {
89         file << timeVector[i]; // Coluna de tempo
90         for (size_t j = 0; j < results.size(); ++j) {
91             file << " " << results[j][i]; // Colunas de dados para cada
                fluido
92         }
93         file << std::endl;
94     }
95
96     file.close();

```


Capítulo 7

Teste

Todo projeto de engenharia passa por uma etapa de testes. Neste capítulo apresentamos alguns testes do software desenvolvido.

Para a realização dos testes, foram consideradas determinadas propriedades com o objetivo de possibilitar a simulação de um ambiente geológico que se aproxime das condições reais.

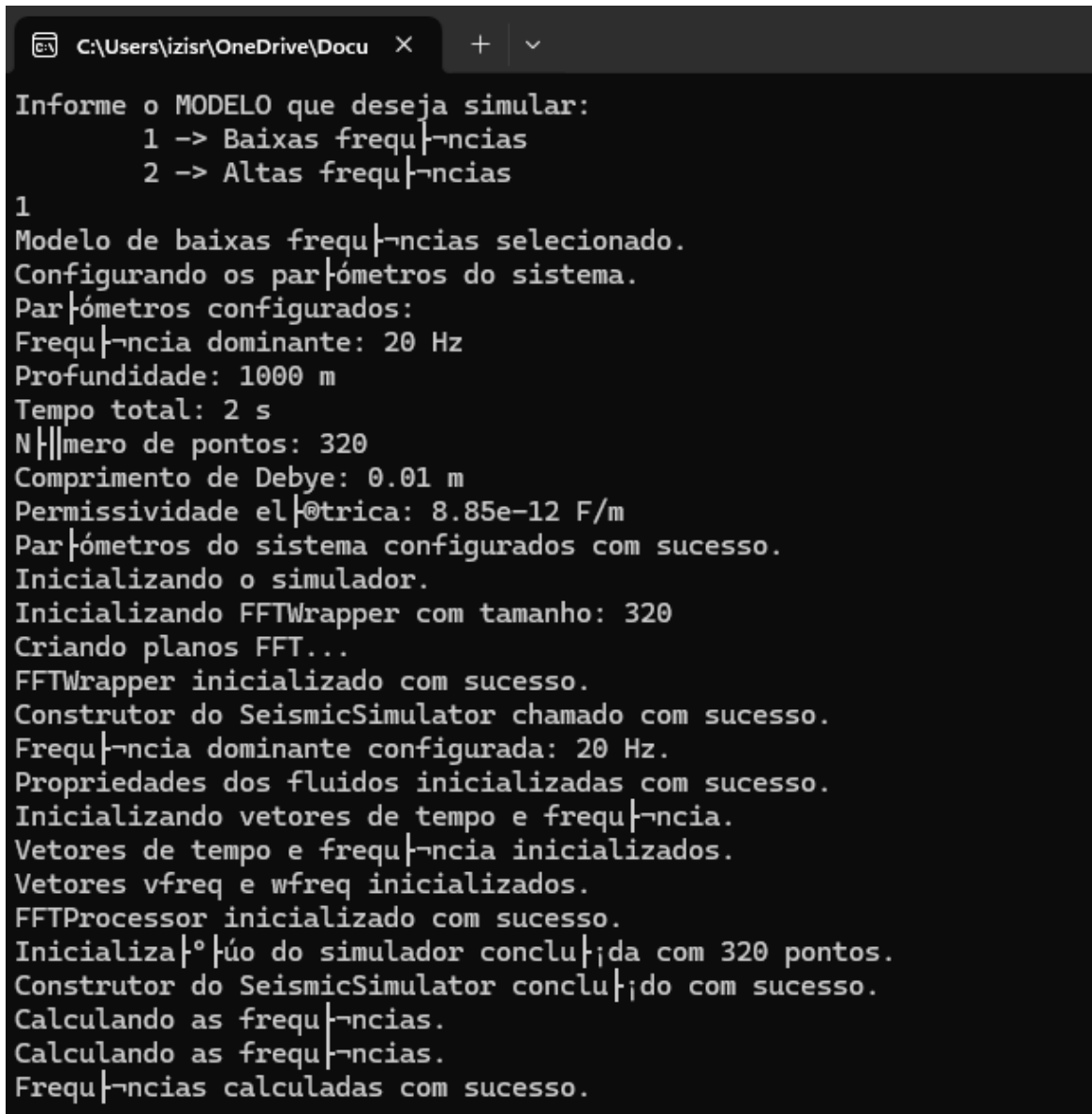
Tabela 7.1: As propriedades petrofísicas e físicas do meio poroso e dos fluidos.

Propriedade	Símbolo	Unidade	Valor
Densidade da água	ρ_1	kg/m^3	1040
Densidade do óleo leve	ρ_2	kg/m^3	844,8
Densidade do óleo médio	ρ_3	kg/m^3	905,1
Densidade do óleo pesado	ρ_4	kg/m^3	967,2
Viscosidade da água	η_1	$Pa.s$	$1,5 \cdot 10^{-3}$
Viscosidade do óleo leve	η_2	$Pa.s$	$8,989 \cdot 10^{-3}$
Viscosidade do óleo médio	η_3	$Pa.s$	$7,379 \cdot 10^{-3}$
Viscosidade do óleo pesado	η_4	$Pa.s$	$8,5378 \cdot 10^{-3}$
Densidade do meio poroso	ρ_s	kg/m^3	2619
Porosidade	ϕ	-	0,4
Tortuosidade	α	-	2,4
Permeabilidade	k	m^2	$1,974 \cdot 10^{-13}$

7.1 Teste 1

Primeiramente, iremos analisar o efeito sismoelétrico num modelo de arenito homogêneo considerando os parâmetros utilizados na Tabela 1. Este modelo foi gerado para verificar-se o tempo de chegada da onda nos receptores está coerente com a velocidade de propagação da onda sísmica no meio poroso e verificar se as amplitudes dos sinais sísmicos estão de acordo com os parâmetros físicos do modelo.

A primeira simulação será considerada uma frequência de 20Hz.



```
C:\Users\izisr\OneDrive\Docu X + v
Informe o MODELO que deseja simular:
  1 -> Baixas frequências
  2 -> Altas frequências
1
Modelo de baixas frequências selecionado.
Configurando os parâmetros do sistema.
Parâmetros configurados:
Frequência dominante: 20 Hz
Profundidade: 1000 m
Tempo total: 2 s
Número de pontos: 320
Comprimento de Debye: 0.01 m
Permissividade elétrica: 8.85e-12 F/m
Parâmetros do sistema configurados com sucesso.
Iniciando o simulador.
Iniciando FFTWrapper com tamanho: 320
Criando planos FFT...
FFTWrapper inicializado com sucesso.
Construtor do SeismicSimulator chamado com sucesso.
Frequência dominante configurada: 20 Hz.
Propriedades dos fluidos inicializadas com sucesso.
Iniciando vetores de tempo e frequência.
Vetores de tempo e frequência inicializados.
Vetores vfreq e wfreq inicializados.
FFTProcessor inicializado com sucesso.
Inicialização do simulador concluída com 320 pontos.
Construtor do SeismicSimulator concluído com sucesso.
Calculando as frequências.
Calculando as frequências.
Frequências calculadas com sucesso.
```

Figura 7.1: Telas de execução do programa escolhendo baixas frequências

A solução obtida em baixas frequências foi:

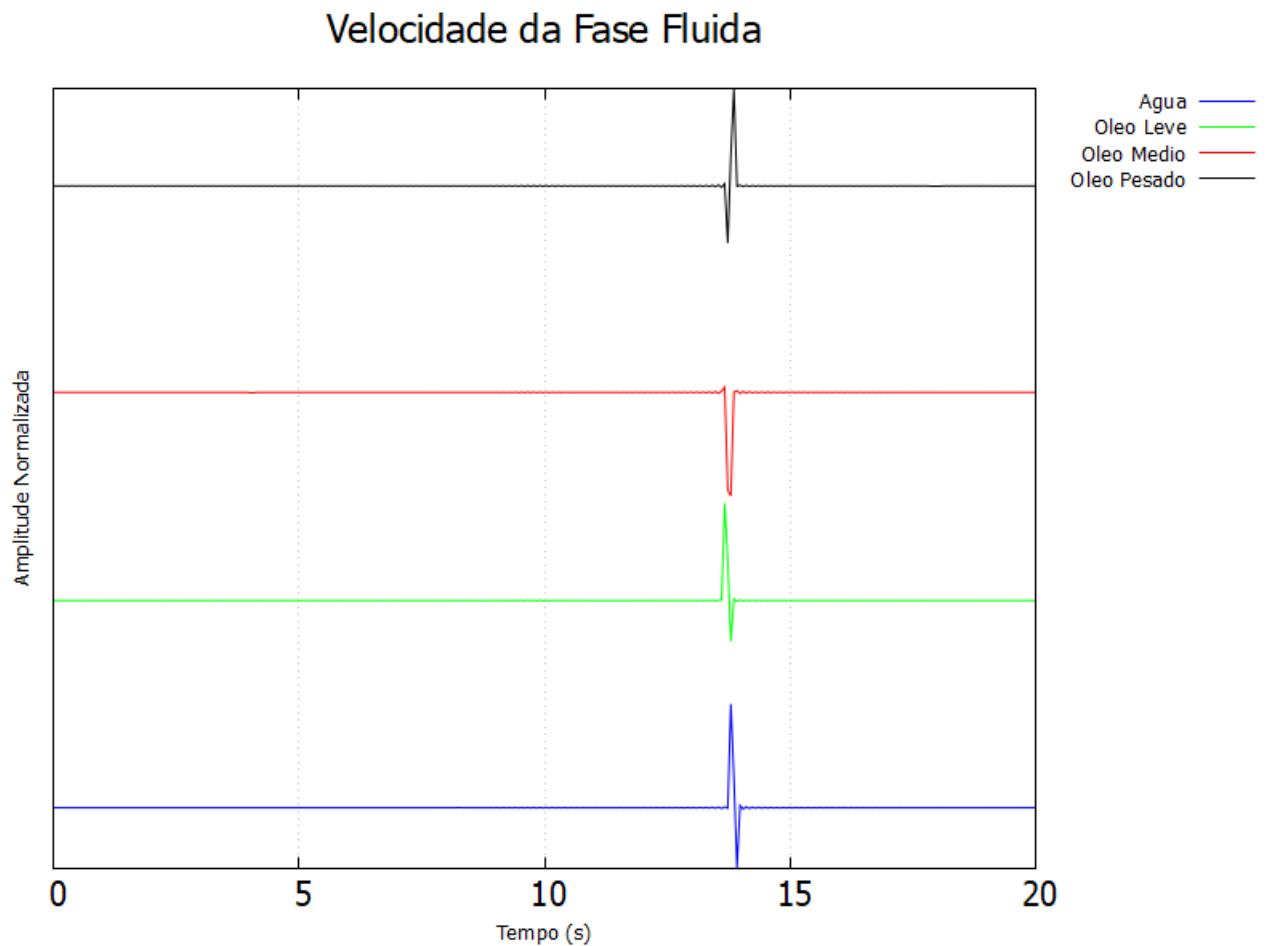


Figura 7.2: Velocidade da fase fluida em baixas frequências

Para baixas frequências, o meio não suporta a onda P lenta, que se torna difusiva, dessa forma, todos os eventos observados são referentes à propagação da onda P rápida.

7.2 Teste 2

Para a segunda simulação serão analisadas as propagações das ondas em altas frequências.

```
C:\Users\izisr\OneDrive\Docu X + v
Informe o MODELO que deseja simular:
    1 -> Baixas frequências
    2 -> Altas frequências
2
Modelo de altas frequências selecionado.
Configurando os parâmetros do sistema.
Parâmetros configurados:
Frequência dominante: 200000 Hz
Profundidade: 1000 m
Tempo total: 1 s
Número de pontos: 320
Comprimento de Debye: 0.01 m
Permissividade elétrica: 8.85e-12 F/m
Parâmetros do sistema configurados com sucesso.
Iniciando o simulador.
Iniciando FFTWrapper com tamanho: 320
Criando planos FFT...
FFTWrapper inicializado com sucesso.
Construtor do SeismicSimulator chamado com sucesso.
Frequência dominante configurada: 200000 Hz.
Propriedades dos fluidos inicializadas com sucesso.
Iniciando vetores de tempo e frequência.
Vetores de tempo e frequência inicializados.
Vetores vfreq e wfreq inicializados.
FFTProcessor inicializado com sucesso.
Inicialização do simulador concluída com 320 pontos.
Construtor do SeismicSimulator concluído com sucesso.
Calculando as frequências.
Calculando as frequências.
Frequências calculadas com sucesso.
Frequências calculadas com sucesso.
Calculando os parâmetros dinâmicos.
Calculando parâmetros dinâmicos.
Parâmetros dinâmicos calculados com sucesso.
Parâmetros dinâmicos calculados com sucesso.
```

Figura 7.3: Telas de execução do programa escolhendo altas frequências

Em altas frequências as ondas P lentas não possuem o comportamento difuso, assim como em baixas frequências.

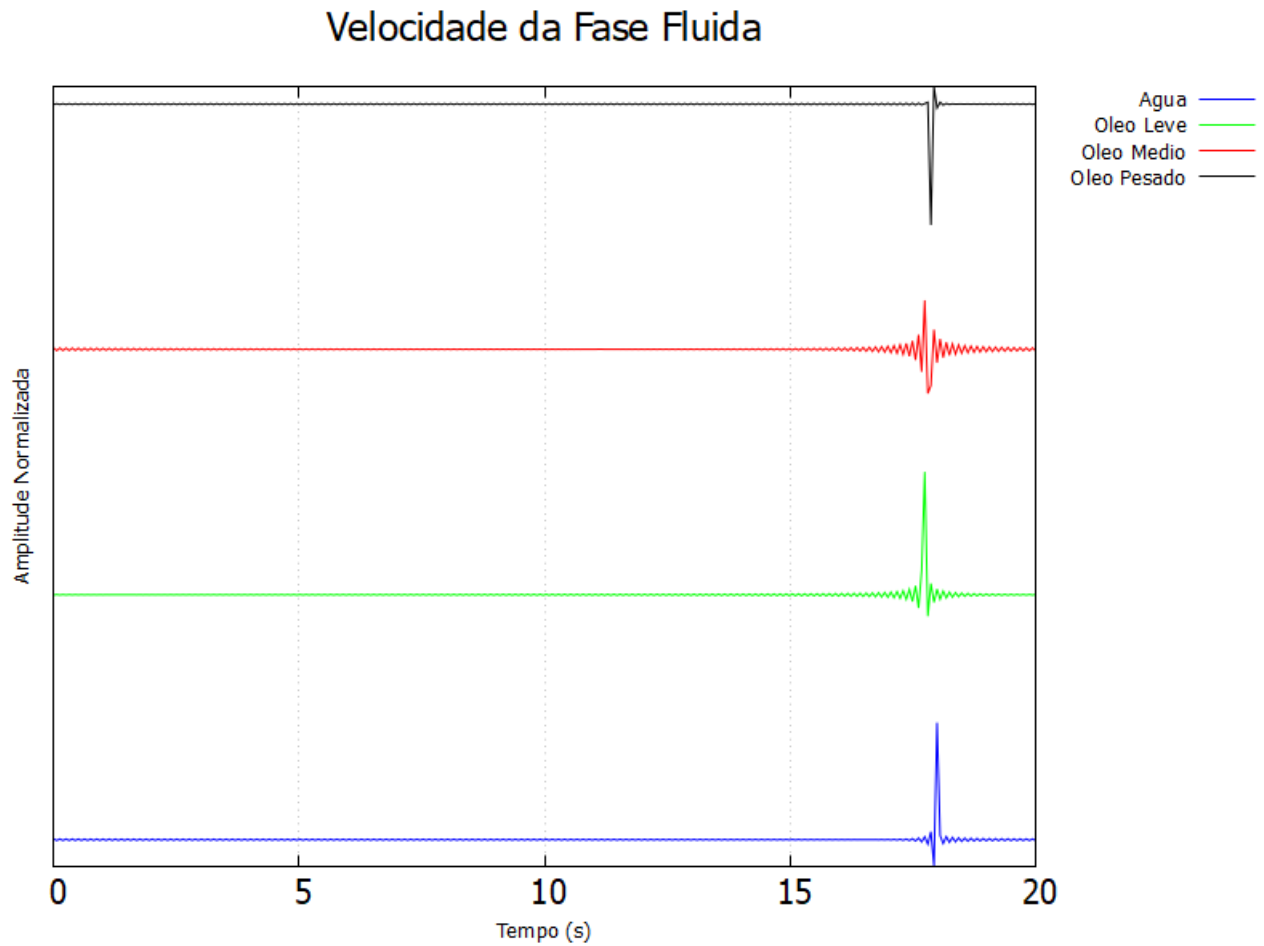


Figura 7.4: Velocidade da fase fluida em altas frequências

Com a simulação do efeito sismoelétrico para o modelo homogêneo para baixas frequências o meio não suporta a onda P lenta, ela se torna difusiva. Quando analisamos a figura percebemos somente a chegada da onda P rápida, pois a onda P lenta é dissipada e tem a variação de amplitude desprezível. Em altas frequências a onda P lenta está relacionada ao movimento relativo, entre a fase sólida e a fase fluida que permanece estática, por isso, ela possui características intrínsecas do fluido, ou seja, a viscosidade do fluido é um mecanismo fundamental de atenuação da onda P lenta.

Capítulo 8

Documentação

Todo projeto de engenharia precisa ser bem documentado. Neste sentido, apresenta-se neste capítulo a documentação sobre o uso do "Simulador do Efeito Sismoelétrico". Esta documentação trás um passo a passo de como usar o software.

8.1 Documentação do usuário

Descreve-se aqui o manual do usuário, um guia que para instalação e uso do software Simulador do Efeito Sismoelétrico.

8.1.1 Como rodar o software

Para rodar o software é necessário:

- Executar o arquivo `main.cpp` via terminal ou abrí-lo diretamente de seu compilador C++ de preferência.
- Seguir as instruções auto-explicativas do programa enquanto ele é executado.

Aqui cabe uma ressalva para modificação das propriedades físicas, propriedades do sistema e as dos fluidos será necessária a alteração diretamente no código-fonte.

8.2 Documentação para desenvolvedor

Apresenta-se nesta seção a documentação para desenvolvedor, isto é, informações para usuários que queiram modificar, aperfeiçoar ou ampliar este software.

8.2.1 Dependências

Para compilar o software é necessário atender as seguintes dependências:

- No sistema operacional GNU/Linux: instalar o compilador `g++` da GNU disponível em <http://gcc.gnu.org>. Para instalar no GNU/Linux use o comando `yum install gcc`.

- No sistema operacional Windows: instalar um compilador apropriado, tal como Dev C++ ou Microsoft Visual Basic for Applications.
- O software `Gnuplot`, disponível no endereço <http://www.gnuplot.info/>, deve estar instalado. É possível que haja necessidade de setar o caminho para execução do `gnuplot`.
- O programa depende ou não da existência de um arquivo de dados (formato `.txt`).

Referências Bibliográficas

- [Blanc 2013]BLANC, E. *Time-domain numerical modeling of poroelastic waves: the Biot-JKD model with fractional derivatives*. Tese (Theses) — Aix-Marseille Université, dez. 2013.
- [Butler et al. 1996]BUTLER, K. E. et al. Measurement of the seismoelectric response from a shallow boundary. *Geophysics*, v. 61, p. 1769–1778, 1996.
- [Chen e Mu 2005]CHEN, B.; MU, Y. Experimental studies of seismoelectric effects in fluid-saturated porous media. *Journal of Geophysics and Engineering*, v. 2, p. 222–230, 2005.
- [Chiavassa, Lombard e Piraux 2010]CHIAVASSA, G.; LOMBARD, B.; PIRAUX, J. Numerical modeling of 1d transient poroelastic waves in the low-frequency range. *J. Computational Applied Mathematics*, v. 234, p. 1757–1765, 2010.
- [Chiavassa, Lombard e Piraux 2010]CHIAVASSA, G.; LOMBARD, B.; PIRAUX, J. Numerical modeling of 1d transient poroelastic waves in the low-frequency range. *J. Computational Applied Mathematics*, v. 234, p. 1757–1765, 2010.
- [Djuraev et al. 2018]DJURAEV, U. et al. Numerical simulation of seismoelectric effect for monitoring foam propagation through a reservoir. *Journal of Petroleum Science and Engineering*, v. 171, p. 618–635, 2018.
- [DJURAEV, U., JUFAR, S. R. e VASANT, P. 2017]DJURAEV, U.; JUFAR, S. R.; VASANT, P. Numerical Study of Frequency-dependent Seismoelectric Coupling in Partially-saturated Porous Media. *Matec Web of Conferences*, v. 87, p. 02001, 2017.
- [Frenkel 2005]FRENKEL, J. On the Theory of Seismic and Seismoelectric Phenomena in a Moist Soil. *Journal of Engineering Mechanics-asce*, v. 131, p. 879–887, 2005.
- [Garambois e Dietrich 2001]GARAMBOIS, S.; DIETRICH, M. Seismoelectric Wave Conversions in Porous Media: Field Measurements and Transfer Function Analysis. *Geophysics*, v. 66, p. 1417–1430, 2001.
- [Haartsen e Pride 1997]HAARTSEN, M. W.; PRIDE, S. R. Electrostatic waves from point sources in layered media. *Journal of Geophysical Research*, v. 102, p. 24745–24769, 1997.
- [Haines e Pride 2006]HAINES, S. S.; PRIDE, S. R. Seismoelectric numerical modeling on a grid. *Geophysics*, v. 71, p. 57–65, 2006.

- [Haines et al. 2007]HAINES, S. S. et al. Seismoelectric imaging of shallow targets. *Geophysics*, v. 72, p. 9–20, 2007.
- [Jouniaux e Bordes 2012]JOUNIAUX, L.; BORDES, C. Frequency-Dependent Streaming Potentials: A review. *International Journal of Geophysics*, v. 2012, p. 1–11, 2012.
- [Jouniaux e Zyserman 2016]JOUNIAUX, L.; ZYSERMAN, F. A review on electrokinetically induced seismo-electrics, electro-seismics, and seismo-magnetics for Earth sciences. *Solid Earth*, v. 7, p. 249–284, 2016.
- [Kroger, Yaramanci e Kemna 2009]KROGER, B.; YARAMANCI, U.; KEMNA, A. Modelling of Seismoelectric Effects. Excerpt from the Proceedings of the COMSOL Conference, Hanôver, 2009.
- [Murthy 1985]MURTHY, Y. S. First results on the direct detection of groundwater by seismoelectric effect - A field experiment. *Exploration Geophysics*, v. 16, p. 254–256, 1985.
- [Oliveira, Azeredo e Priimenko 2018]OLIVEIRA, I.; AZEREDO, M.; PRIIMENKO, V. Modelagem de propagação das ondas elásticas em meios porosos 1D - modelos de Biot vs. Biot-JKD. *Revista Brasileira de Geofísica*, p. 1–5, 2018.
- [Pride 1994]PRIDE, S. Governing equations for the coupled electromagnetics and acoustics of porous media. *Physical review. B, Condensed matter*, v. 50, p. 15678–15696, 1994.
- [Pride, Berryman e Harris 2004]PRIDE, S. R.; BERRYMAN, J. G.; HARRIS, J. M. Seismic attenuation due to wave-induced flow. *Journal of Geophysical Research: Solid Earth*, v. 109, 2004.
- [Pride e Haartsen 1996]PRIDE, S. R.; HAARTSEN, M. W. Electrostatic wave properties. *The Journal of the Acoustical Society of America*, v. 100, p. 1301–1315, 1996.
- [PRIIMENKO V.; VISHNEVSKII 2007]PRIIMENKO V.; VISHNEVSKII, M. Mathematical Problems of Electromagnetoelastic Interactions. *Boletim da Sociedade Paranaense de Matemática*, v. 25, p. 55–66, 2007.
- [Ren, Huang e Chen 2015]REN, H.; HUANG, Q.; CHEN, X. Existence of evanescent electromagnetic waves resulting from seismoelectric conversion at a solid-porous interface. *Geophysical Journal International*, v. 204, p. 147–166, 2015.
- [Ricker 1943]RICKER, N. Further developments in the wavelet theory of seismogram structure. *Bulletin of the Seismological Society of America*, v. 33, p. 197–228, 1943.
- [Ricker 1944]RICKER, N. Wavelet functions and their polynomials. *Geophysics*, v. 9, n. 3, p. 314–323, 1944.
- [Ricker 1953]RICKER, N. The form and laws of propagation of seismic wavelets. *Geophysics*, v. 18, n. 1, p. 10–40, 1953.

- [Ricker 1953]RICKER, N. Wavelet contraction, wavelet expansion, and the control of seismic resolution. *Geophysics*, v. 18, p. 769–792, 1953.
- [Russell et al. 1997]RUSSELL, R. D. et al. Seismoelectric exploration. *The Leading Edge*, v. 16, p. 1611–1615, 1997.
- [Thompson e Gist 1993]THOMPSON, A. H.; GIST, G. A. Geophysical applications of electrokinetic conversion. *The Leading Edge*, v. 12, p. 1169–1173, 1993.
- [Thompson 1936]THOMPSON, R. R. The seismic electric effect. *Geophysics*, v. 1, p. 327–335, 1936.
- [Ursin 1983]URSIN, B. Review of elastic and electromagnetic wave propagation in horizontally layered media. *Geophysics*, v. 48, p. 1063–1081, 1983.
- [Vanzeler F.; Priimenko 2009]VANZELER F.; PRIIMENKO, V. Modelagem Numérica do Efeito Seismoeletrico em Meios 2D. *Revista Brasileira de Geofísica [online]*, v. 27, p. 63–80, 2009.
- [Zhu e McMechan 1991]ZHU, X.; MCMECHAN, G. A. Numerical simulation of seismic responses of poroelastic reservoirs using biot theory. *Geophysics*, v. 56, p. 328–339, 1991.
- [Zhu, Haartsen e Toksoz 2000]ZHU, Z.; HAARTSEN, M. W.; TOKSOZ, M. N. Experimental studies of seismoelectric conversions in fluid-saturated porous media. *Journal of Geophysical Research: Solid Earth*, v. 105, p. 28055–28064, 2000.