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

PROJETO ENGENHARIA
DESENVOLVIMENTO DO SOFTWARE
SIMULADOR DE RESERVATÓRIO MONOFÁSICO 2D
TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA

Versão 1:
NICHOLAS DE ALMEIDA PINTO
KEVIN ALVES BARTELEGA
Prof. André Duarte Bueno

MACAÉ - RJ
Novembro - 2021

Sumário

Capítulo 1

Introdução

No presente projeto de engenharia desenvolve-se o software SIMULADOR DE RESERVATÓRIO MONOFÁSICO 2D, um código em linguagem orientada a objeto que tem como principal objetivo implementar as equações vistas nas disciplinas de Avaliação de Formações e Engenharia de Reservatórios ([Pico, 2018]).

Dessa forma, a principal finalidade do programa é fornecer o cálculo do campo de pressões em um dado poço de um reservatório de óleo ou de gás. Para isso, utilizou-se uma simulação numérica computacional baseada no método dos volumes finitos. Esta é uma ferramenta poderosa de resolução de equações diferenciais parciais a um determinado volume de meio contínuo baseado, por exemplo, na resolução de balanços de massa.

1.1 Escopo do problema

Em se tratando da Engenharia de Reservatórios, o foco do estudo é o próprio reservatório de óleo ou de gás. Os engenheiros lutam por mais entendimento do comportamento de um reservatório, para que se possa fazer predições cada vez mais condizentes com as medidas de campo e aumentar a segurança em dizer se um campo é viável ou não à exploração e por quanto tempo esse campo será viável. Dada uma aplicação de injeção ou produção em poços no reservatório, se faz necessário um conhecimento sólido e completo de como ele se comportará e influenciará a dinâmica de pressões no poço ([ROSA, 2006]).

Portanto, o problema que se propõe a resolver é a simulação de poços com propriedades distintas, para otimizar a produção no reservatório. De posse dela - ou de, pelo menos, um valor próximo estimado pelo software, seria possível, por exemplo, dimensionar equipamentos de fundo do poço, prever tempo produtivo e quantificação o volume de fluido de completação, por exemplo.

1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:
  - Desenvolver projeto de engenharia de software baseado em simulação numérica implícita computacional para determinar a evolução da pressão em um poço dentro de um reservatório estratigráfico de óleo ou gás.
- Objetivos específicos:
  - Modelar física e matematicamente o problema.
  - Modelagem estática por meio de diagramas com interface amigável.
  - Calcular Permeabilidade.
  - Calcular Transmissibilidade.
  - Calcular Matriz de Coeficientes.
  - Resolver Sistemas de equações.
  - Calcular Pressões.
  - Simular para diferentes fluidos dentro do reservatório (óleo ou gás).
  - Simular para diferentes camadas estratigráficas rochosas.
  - Gerar gráficos externos a partir do software externo Gnuplot..

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	SIMULADOR DE RESERVATÓRIO MONOFÁSICO 2D
Componentes principais	Sistema para cálculos da distribuição de pressão em um poço/reservatório em função das coordenadas espaço-temporais, utilizando método numérico implícito
Missão	Calcular pressão no poço ao longo do tempo

2.2 Especificação

Deseja-se desenvolver um software com interface em modo texto que seja capaz de determinar o comportamento das pressões dentro de um poço. O processo é governado pela Equação da Equação da Difusividade Hidráulica. Será utilizada a modelagem numérica pela discretização em volumes finitos e método implícito de Newton para resolução.

Na dinâmica de execução do software, o usuário deverá entrar com os dados relativos ao fluido, à matriz da rocha, ao meio poroso, ao grid-2D, ao simulador, os valores das permeabilidade das camadas estrátigraficas, inserir espessuras delas, dizer ao software quais camadas abertas à produção, bem como o tipo de fluido presente no reservatório, se óleo ou gás. Poderá optar-se também pela inserção dos dados em um documento de texto *.txt. Dada a primeira ou segunda escolha, o software calcula suas propriedades termofísicas, e por fim, apresenta a pressão no poço e no reservatório.

Os dados com as suas respectivas unidades estão listados abaixo:

- **Dados relativos ao fluido:**

k permeabilidade $[md]$;

ρ_f massa específica do fluido $[kg/m^3]$;

c_{pf} calor especíco à pressão constante do fluido $[J/KgK]$.

- **Dados relativos à matriz da rocha:**

  ϕ porosidade absoluta $[m^3/m^3]$;

- **Dados relativos ao meio poroso:**

  T temperatura absoluta $[K]$;

  P pressão $[Pa]$;

- **Dados relativos ao grid bidimensional:**

  dx intervalo de discretização na direção x $[m]$;

  dy intervalo de discretização na direção y $[m]$;

- **Dados relativos ao simulador:**

  dt intervalo de tempo $[s]$;

Após a entrada de dados pelo usuário, o programa irá calcular as propriedades do fluido escolhido e irá resolver a EDP discretizada com um método numérico, obtendo uma solução numérica implícita para cada passo de tempo, isto é, uma distribuição da pressões P(r, z, t), como função das coordenadas espaciais e temporais.

O software então irá plotar gráficos que serão gerados com um programa externo (gnuplot).

Por fim, vale destacar que o software cuja interface será em modo texto, será escrito na linguagem C++ com o paradigma de orientação ao objeto, uma linguagem reconhecida por sua grande eciência, abrangência e facilidade no reaproveitamento de códigos desenvolvidos previamente.

2.2.1 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

RF-01	O usuário tem a liberdade de escolher todos os dados de entrada, mencionados na seção 2.2.

RF-02	O usuário pode obter a distribuição de pressão (r,z) para qualquer tempo (t).

RF-03	O usuário pode modelar o processo de simulação escolhendo qual tipo de fluido, bem como as camadas estratigráficas nas quais haverá fluxo.

RF-04	Deve mostrar os resultados na tela.

RF-05	O usuário poderá plotar seus resultados de simulação em gráficos. O gráfico poderá ser salvo como imagem ou ter seus dados exportados como texto.

2.2.2 Requisitos não funcionais

RNF-01	Os cálculos devem ser feitos utilizando-se o método numérico implícito para cada passo de tempo.

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

2.3 Casos de uso

Tabela 2.1: Exemplo de caso de uso

Nome do caso de uso:	Cálculo da pressão
Resumo/descrição:	Cálculo da pressão em poço e reservatório em determinadas condições
Etapas:	1. Entrada de dados. 2. Executar o software 3. Gerar gráficos. 4. Analisar resultados.
Cenários alternativos:	Um cenário alternativo envolve um poço com penetração parcial e liquído no reservatório.

2.3.1 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, calculando a pressão ou analisando resultados. Este diagrama de caso de uso ilustra as etapas a serem executadas pelo usuario ou sistema, ou seja, a iteração do usuário com o sistema.

2.3.2 Diagrama de caso de uso específico

Os diagramas de casos de uso específicos estão descritos nas Figuras 2.1 e 2.3 e na Tabela 2.1 Ele mostra a interação usuário-software para calcular a pressão no reservatório e no poço usando o método numérico implícito.

No primeiro caso de uso específico, ou sub caso, mostra-se as possibilidades de se simular o software com fluido ora líquido ora gás, e penetração do poço parcial ou total. Entende-se por penetração as áreas abertas ao fluxo adjacentes ao poço.

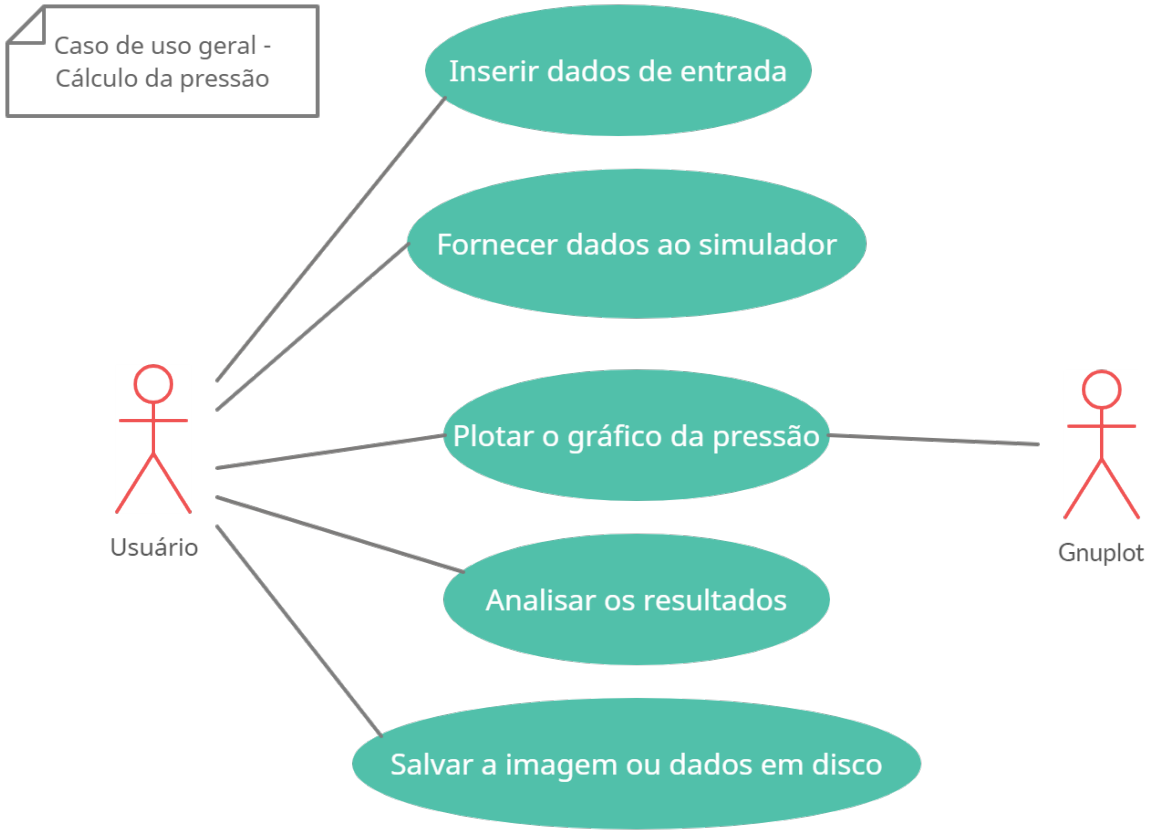

Figura 2.1: Diagrama de caso de uso – Caso de uso geral

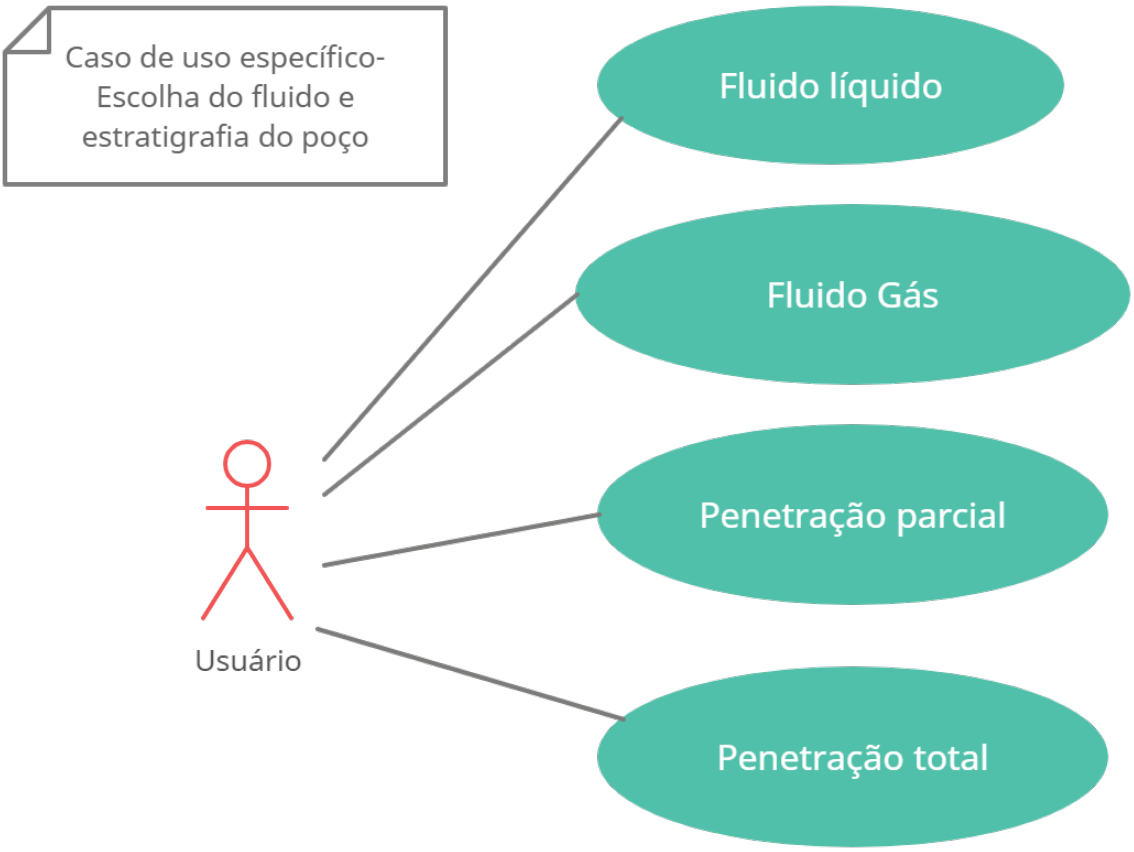

Figura 2.2: Diagrama de caso de uso específico – Título

Já no segundo caso de uso especíco (4 etapas), o usuário optou por simular um líquido com penetração parcial. Assim, insere os dados de entrada, define zonas de fluxo no poço, executa a simulação. Depois disso, o software gera gráficos e o usuário analisa os resultados.

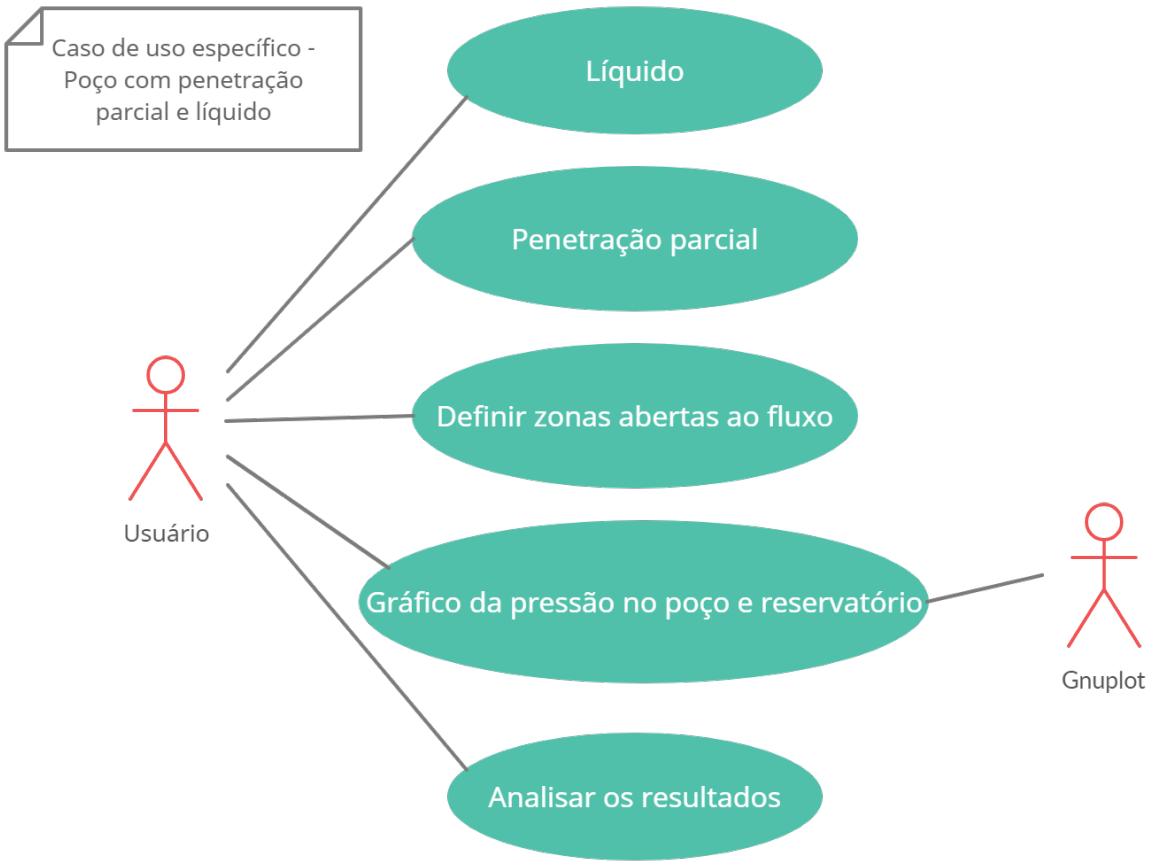

Figura 2.3: Diagrama de caso de uso específico – Título

Capítulo 3

Elaboração

Depois da definição dos objetivos, da especificação do software e da montagem dos primeiros diagramas, neste capítulo será apresentada a elaboração, que envolve o estudo de conceitos relacionados ao sistema a ser desenvolvido, a análise de domínio e a identificação de pacotes. Esse processo é feito através de pesquisas bibliográficas e entrevistas, que nos mostram o que é necessário para a formulação do programa.

Uma analise dos requisitos para o funcionamento do programa será feita para se avaliar as condições necessárias para o desenvolvimento de um sistema útil, que satisfaça as necessidades requeridas e que permita extensão futura.

3.1 Análise de domínio

Após estudo dos requisitos/especificações do sistema, leitura de artigos recomendados e disciplinas do curso foi possível identificar nosso domínio de trabalho no desenvolvimento do simulador.

- Engenharia de Reservatórios: parte fundamental na qual esse projeto se sustenta. O software desenvolvido, utiliza conceitos tais como de propriedades dos fluídos , propriedades de rochas, a Equação de Balanço de Materiais (EBM). Ele então aplicará todos esses conceitos na caracterização adicional do reservatório e do poço o que permite a predição do comportamento de ambos ao longo da produção.

- A Simulação de Reservatórios é um ramo da na engenharia de reservatório. Trata da utilização do desenvolvimento de simuladores, que por meio de modelos matemáticos buscam prever o comportamento de um reservatório de petróleo e de seus poços associados. Os simuladores podem ser do tipo black oil ou composicionais, no primeiro o óleo é considerado uma substância só, e no segundo uma mistura de diversas substâncias.

- Modelagem Numérica Computacional que desenvolve modelos matemáticos para a solução de um determinado problema físico e então parte para um o modelo

computacional por meio de algoritmos a fim de encontrar a solução do problema. Utilizou-se conceitos matemáticos de Cálculo Numérico, vistos na primeira parte do curso e aprimorados no ciclo profissionalizantes. Neste software foi utilizados, por exemplo, o método numérico de Newton-Raphson.

- A Termodinâmica é uma área da física que estuda os efeitos de mudanças na temperatura, pressão, volume e outras propriedades termodinâmicas de um sistema. Ela é extremamente importante no desenvolvimento de um simulador de reservatório pois os fluidos de um reservatório sofrem diversas alterações físicas durante sua produção, sendo necessária uma boa modelagem termodinâmica para entender como eles reagirão a estas alterações.

- Álgebra linear e Cálculo Integral e Diferencial na resolução de sistemas de matrizes e em cálculos de derivadas parciais, Jacobianos, por exemplo.

- Pacote Gráfico: usar-se-á um pacote gráfico para plotar o comportamento da pressão, por exemplo, ao longo do poço e do reservatório para que haja uma melhor compreeensão e vizualização.

- Software: Serão utilizadas métodos e funções já existentes para a resolução de sistemas de matrizes.

- Elaboração 3.1. Analise de dominio 3.2. Formulação teorica 3.3. Identificacao de pacotes - assuntos 3.4. Diagrama de Pacotes - assuntos

3.2 Formulação teórica

O petróleo é uma das matérias-primas mais importantes utilizadas pelo homem. Infelizmente, os reservatórios rasos estão quase todos esgotados ou possuem óleos de baixa qualidade, sendo necessária a extração em altas profundidades e em reservatórios de geometria e propriedades complexas.

Nesse contexto, os métodos de recuperação secundária e avançada são as ferramentas mais empregadas para otimizar a produção. As jazidas de petróleo possuem uma quantidade de energia, denominada energia primária, na época de sua descoberta, determinada pelas condições de pressão e temperatura e pela natureza dos fluidos existentes. Porém, à medida que os fluidos são produzidos, parte dessa energia primária é dissipada e o efeito reflete-se principalmente no decréscimo da pressão do reservatório e consequente redução da produtividade dos poços.

Para minorar os efeitos do decréscimo da pressão e obter ótimas porcentagens de recuperação, são utilizados métodos de recuperação avançados, como injeção de água, gases, solventes, etc. Mas somente injetar fluidos em poços próximos ao produtor não é suficiente para maximizar a extração, é necessário também saber onde, quando, quanto e

quais devem ser as propriedades do fluido a ser injetado. Para isto, são realizados, entre outros, testes de pressão, que permitem identificar ou caracterizar o sistema fluido/rocha de cada reservatório. Para a interpretação destes testes é necessário o desenvolvimento de um modelo teórico que descreva o escoamento dos fluidos no reservatório.

Em determinadas situações é factível resolver analiticamente as equações do modelo, porém, estes casos se limitam com frequência ao escoamento monofásico, regido por equações diferenciais lineares. Em casos mais complexos, como a injeção de fluidos alheios ao reservatório, possivelmente com diferentes temperaturas, a complexidade matemática do modelo não permite a sua solução analítica. Nestas situações, as equações diferenciais do modelo são resolvidas utilizando métodos numéricos.

3.2.1 Fluxo monofásico

A equação do escoamento monofásico em meios porosos e em coordenadas cilíndricas (r, z) é:

$$\alpha_c \frac{\partial}{\partial t}(\phi b) = \beta_c \frac{1}{r} \frac{\partial}{\partial r}\left(r \frac{k_r b}{\mu} \frac{\partial p}{\partial r}\right) + \beta_c \frac{\partial}{\partial z}\left(\frac{k_z b}{\mu} \frac{\partial p}{\partial z}\right) + q_{sc} \tag{3.1}$$

onde:

- α é a constante de conversão das unidades de acúmulo;
- ϕ é a porosidade;
- b é o inverso do fator volume formação (Volume do óleo nas condições padrão / volume do óleo nas condições de reservatório);
- β é a constante de conversão das unidades de fluxo;
- k_r é a permeabilidade radial;
- k_z é a permeabilidade vertical;
- μ é a viscosidade do óleo;
- q_{sc} é a vazão do poço.

Considere o seguinte arranjo de um elemento de volume em coordenadas cilíndricas (Fig. 3.1) e a malha (Fig. 3.2) no sistema radial abaixo:

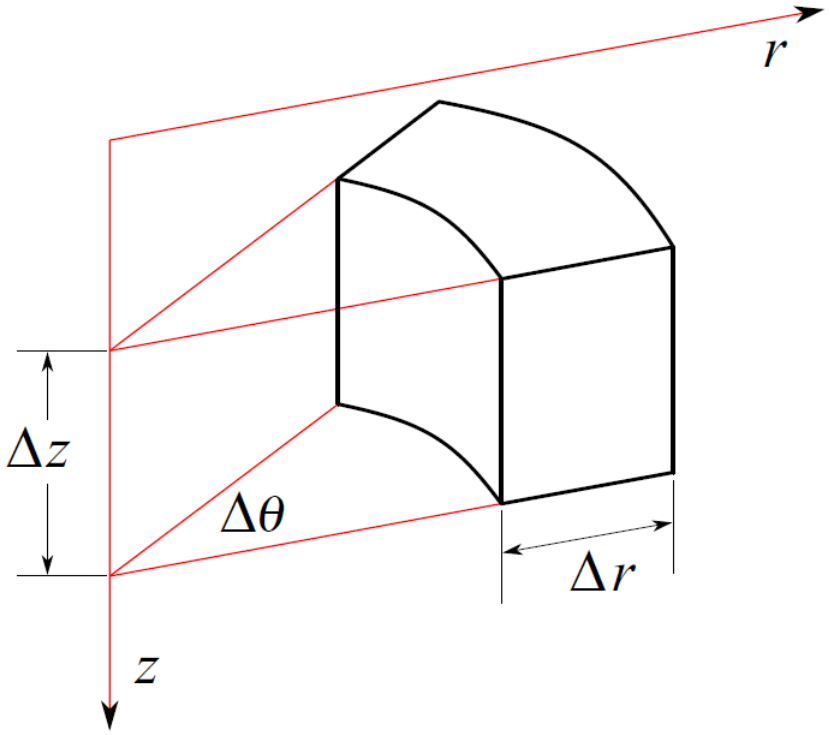

Figura 3.1: Elemento de volume em coordenadas cilndricas.

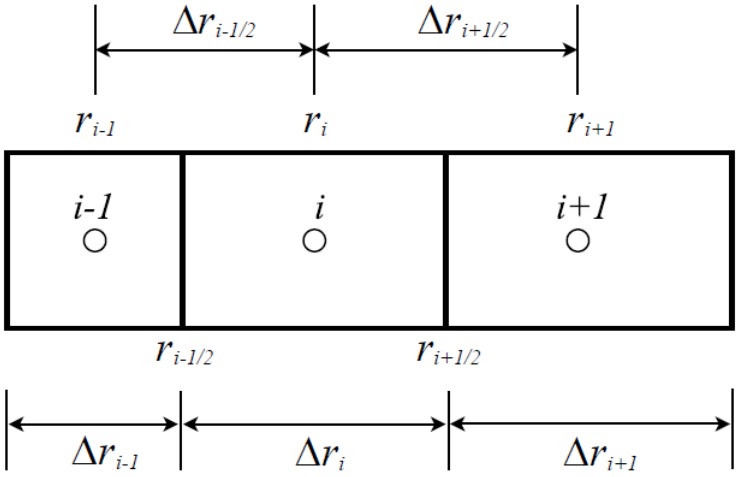

Figura 3.2: Malha radial não-homogênea.

Aplicando-se a dicretização por volumes finitos, podemos reecrever a Eq. 3.1 como:

$$\frac{\alpha_c V_{i,j}}{\Delta t}\left[(\phi b)_{i,j}^{n+1}-(\phi b)_{i,j}^{n}\right]= \begin{array}{l} T_{i+\frac{1}{2},j}^{n+1}\left(p_{i+1,j}^{n+1}-p_{i,j}^{n+1}\right)-T_{i-\frac{1}{2},j}^{n+1}\left(p_{i,j}^{n+1}-p_{i-1,j}^{n+1}\right) \quad + \\ T_{i,j+\frac{1}{2}}^{n+1}\left(p_{i,j+1}^{n+1}-p_{i,j}^{n+1}\right)-T_{i,j-\frac{1}{2}}^{n+1}\left(p_{i,j}^{n+1}-p_{i,j-1}^{n+1}\right) \quad +q_{gsc,i,j} \end{array} \tag{3.2}$$

onde a transmissibilidade T é definida por:

$$T_{i\pm\frac{1}{2},j} = G_{i\pm\frac{1}{2},j}\left(\frac{b}{\mu}\right)_{i\pm\frac{1}{2},j} \tag{3.3}$$

$$T_{i,j\pm\frac{1}{2}} = G_{i,j\pm\frac{1}{2}}\left(\frac{b}{\mu}\right)_{i,j\pm\frac{1}{2}} \tag{3.4}$$

As propriedades dos fluidos nas interfaces são:

$$\psi_{i+\frac{1}{2},j} = (1-\Omega)\,\psi_{i,j} + \Omega\psi_{i+1,j} \quad ,\psi = \mu, b, \phi \tag{3.5}$$

$$\psi_{i,j+\frac{1}{2}} = \frac{\psi_{i,j}+\psi_{i,j+1}}{2} \quad ,\psi = \mu, b, \phi \tag{3.6}$$

As permeabilidades nas interfaces são dadas pelo conjeuntop de 4 equações abaixo:

$$k_{i+\frac{1}{2},j} = \frac{k_i k_{i+1} ln\left(\frac{r_{i+1}}{r_i}\right)}{k_i ln\left(\frac{r_{i+1}}{r_{i+\frac{1}{2}}}\right) + k_{i+1} ln\left(\frac{r_{i+\frac{1}{2}}}{r_i}\right)} \tag{3.7}$$

$$k_{i-\frac{1}{2},j} = \frac{k_{i-1} k_i ln\left(\frac{r_i}{r_{i-1}}\right)}{k_{i-1} ln\left(\frac{r_i}{r_{i-\frac{1}{2}}}\right) + k_i ln\left(\frac{r_{i-\frac{1}{2}}}{r_{i-1}}\right)} \tag{3.8}$$

$$k_{i,j+\frac{1}{2}} = \frac{z_{i,j+1} - z_{i,j}}{\frac{z_{i,j+1}-z_{i,j+\frac{1}{2}}}{k_{i,j+1}} + \frac{z_{i,j+\frac{1}{2}}-z_{i,j}}{k_{i,j}}} \tag{3.9}$$

$$k_{i,j-\frac{1}{2}} = \frac{z_{i,j} - z_{i,j-1}}{\frac{z_{i,j}-z_{i,j-\frac{1}{2}}}{k_{i,j}} + \frac{z_{i,j-\frac{1}{2}}-z_{i,j-1}}{k_{i,j-1}}} \tag{3.10}$$

Seguem abaixo outras fórmulas utilizadas no desenvolvimento anterior colocadas aqui para não quebra alinha de raciocínio:

$$G_{i\pm\frac{1}{2},j} = \beta_c \frac{r_{i\pm\frac{1}{2}} k_{i\pm\frac{1}{2}}}{\Delta r_{i\pm\frac{1}{2}}} \Delta\theta\Delta z \tag{3.11}$$

$$r_{i+\frac{1}{2}} = \frac{r_{i+1}-r_i}{ln\left(\frac{r_{i+1}}{r_i}\right)}, \quad r_{i-\frac{1}{2}} = \frac{r_i - r_{i-1}}{ln\left(\frac{r_i}{r_{i-1}}\right)} \tag{3.12}$$

$$V_{bi} = \frac{1}{2}\left(r^2_{i+\frac{1}{2}} - r^2_{i-\frac{1}{2}}\right)\Delta\theta\Delta z \tag{3.13}$$

As duas próximas subseções trazem propriedades dos gases e dos líquidos, caso opte-se pela escolha de um dos dois em questão.

3.2.2 Propriedades dos gases

O comportamento de gás está definido pela equação de estado de gás real abaixo:

$$pV = ZnRT \tag{3.14}$$

O fator de compressibilidade Z está dado pela correlação apresentada por [Kareem et al., 2015]. Esta correlação permite calcular explicitamente fator de compressibilidade nas faixas $0.2 \leq p_{pr} \leq 15$ e $1.15 \leq T_{pr} \leq 3$ de forma simples.

O inverso do fator volume formação do gás é calculado usando a equação de estado:

$$b = \frac{T_o p}{T p_o Z} \tag{3.15}$$

A viscosidade do gás é calculada pela correlação de [Lee and Eakin, 1966], como função da temperatura, a massa molecular aparente Ma e a massa específica:

$$\rho = \frac{pM_a}{ZRT} \tag{3.16}$$

Vale destacar que, para efeitos de simplificação, desconsiderou-se os efeitos não-darcianos que podem ocorrer nas proximidades do poço. A título de curiosiadade, a equação de Forchheimer é o modelo empregado para representar tais eeitos ou feitos de inercia, causados pela a alta velocidade. Segundo [K. Aziz, 1979], em um sistema consistente de unidades, a Eq. é:

$$-\frac{\partial p}{\partial x} = \frac{\mu}{k} u + \beta \rho u \left| u \right| \tag{3.17}$$

onde β é o coeciente de Forchheimer, cuja dimensão e o inverso do comprimento, u é a vazão por unidade de área e x a direção paralela ao escoamento.

Em termos práticos, um fator de correção $\delta_{i\pm\frac{1}{2}}$ deve ser calculado em conjunto com a solução nas interfaces ([MacDonald and Coats, 1970];[MacDonald and Coats, 1970, Pico, 2018]).

3.2.3 Propriedades dos líquidos

No caso de um líquido, as equações foram:

Para inverso do fator volume formação:

$$b_l = b_l^0 \left(1 + c_l \left(p_l - p_l^0\right)\right) \tag{3.18}$$

Para viscosidade:

$$\mu_l = \mu_l^0 \left(1 + c_l \left(p_l - p_l^0\right)\right) \tag{3.19}$$

3.2.4 Equação geral

De posse das equações anteriores, é possível reescrever a equação da discretização por volumes finitos (Eq. 3.1) como:

$$R = TP + Q - H \tag{3.20}$$

onde:

- R é o vetor de resíduos;
- T é a matriz de transmissibilidade;
- Q é o vetor de vazões;
- H é o vetor de acúmulo;
- P é o vetor de pressões.

O formato das matrizes com termo do poço fica:

$$T = \begin{bmatrix}
W & WR & & & WR & & & WR & & \\
RW & C & N & & T & & & & & \\
 & S & C & N & & T & & & & \\
 & & S & C & & & T & & & \\
RW & B & & & C & N & & T & & \\
 & & B & & S & C & N & & T & \\
 & & & B & & S & C & & & T \\
RW & & & & B & & & C & N & \\
 & & & & & B & & S & C & N \\
 & & & & & & B & & S & C
\end{bmatrix} \tag{3.21}$$

onde T é uma matriz não homogênea.

$$Q = [q_{sc}, 0, 0, ...0]^T \tag{3.22}$$

.

$$H = \left[0, H_{\{1,1\}}, H_{\{2,1\}}, ..., H_{\{n_r,n_z\}}\right]^T \tag{3.23}$$

$$P = \left[P_w, P_{1,1}, P_{2,1}, ..., P_{n_r,n_z}\right]^T \tag{3.24}$$

Sendo:

$$H_{i,j} = \frac{\alpha_c V_{i,j}}{\Delta t}\left[(\phi b)^{n+1}_{i,j} - (\phi b)^{n}_{i,j}\right] \tag{3.25}$$

3.2.5 Jacobiano

Para resolver esse sistema linear, foi utilizado o método de Newton-Raphson. Esse método requer uma solução iterativa, por meio da equação abaixo:

$$J^{(\nu)} P^{\nu+1} = -R^{\{(\nu)\}} \tag{3.26}$$

$$J = \left[\frac{\partial R_{i,j}}{\partial p_{i,j}} \right]_{nr*nz\, x\, nr*nz} \tag{3.27}$$

onde J é a matriz Jacobiana, com as derivadas das equações de resíduo, com relação às incógnitas ([K. Aziz, 1979];[T. Ertekin, 2001]).

O J também pode ser calculado como:

$$J = T + \tau - \eta \tag{3.28}$$

sendo T a matriz de transmissibilidades, τ a derivada dos termos de fluxo e η as derivadas do termo de acúmulo. Abaixo a equação para η:

$$\eta_{i,j} = \frac{\alpha_c V_{b_{i,j}}}{\Delta t} \left(\phi_{i,j} \frac{\partial b_{i,j}}{\partial p_{i,j}} + b_i \frac{\partial \phi_{i,j}}{\partial p_{i,j}} \right) \tag{3.29}$$

$$\eta = \begin{bmatrix} 0 & & & & & & & & \\ & \eta_1 & & & & & & & \\ & & \eta_2 & & & & & & \\ & & & \eta_3 & & & & & \\ & & & & \eta_4 & & & & \\ & & & & & \eta_5 & & & \\ & & & & & & \eta_6 & & \\ & & & & & & & \eta_7 & \\ & & & & & & & & \eta_8 \\ & & & & & & & & & \eta_9 \end{bmatrix} \tag{3.30}$$

$$\tau_{i\pm\frac{1}{2},j} = \frac{\partial T_{i\pm\frac{1}{2},j}}{\partial p_{i\pm1,j}} \left(p_{i\pm1,j} - p_{i,j} \right) \tag{3.31}$$

$$\tau_{i,j\pm\frac{1}{2}} = \frac{\partial T_{i,j\pm\frac{1}{2}}}{\partial p_{i,j\pm1}} \left(p_{i,j\pm1} - p_{i,j\pm1} \right) \tag{3.32}$$

$$\begin{aligned}
\tau_{i,j} &= G_{i-1,j}\frac{(1-\Omega)}{\mu_{i-\frac{1}{2},j}}\left[\frac{\partial b_{i,j}}{\partial p_{i,j}} - \left(\frac{b}{\mu}\right)_{i-\frac{1}{2},j}\frac{\partial \mu_{i,j}}{\partial p_{i,j}}\right](p_{i-1,j} - p_{i,j}) \quad + \\
&+ G_{i+\frac{1}{2},j}\frac{\Omega}{\mu_{i+\frac{1}{2},j}}\left[\frac{\partial b_i}{\partial p_i} - \left(\frac{b}{\mu}\right)_{i+\frac{1}{2},j}\frac{\partial \mu_{i,j}}{\partial p_{i,j}}\right](p_{i+1,j} - p_{i,j}) \quad + \\
&+ G_{i,j-\frac{1}{2}}\left[\frac{(\mu_{i,j-1}+\mu_{i,j})\frac{\partial b_{i,j}}{\partial p_{i,j}} + (b_{i,j-1}+b_{i,j})\frac{\partial \mu_{i,j}}{\partial p_{i,j}}}{(\mu_{i,j-1}+\mu_{i,j})^2}\right](p_{i,j-1} - p_{i,j}) \quad + \\
&+ G_{i,j+\frac{1}{2}}\left[\frac{(\mu_{i,j+1}+\mu_{i,j})\frac{\partial b_{i,j}}{\partial p_{i,j}} + (b_{i,j+1}+b_{i,j})\frac{\partial \mu_{i,j}}{\partial p_{i,j}}}{(\mu_{i,j+1}+\mu_{i,j})^2}\right](p_{i,j+1} - p_{i,j})
\end{aligned} \tag{3.33}$$

As derivadas das transmissibilidades são:

$$\frac{\partial T_{i-\frac{1}{2},j}}{\partial p_{i-1,j}} = G_{i-1,j}\frac{(1-\Omega)}{\mu_{i-\frac{1}{2},j}}\left[\frac{\partial b_{i-1,j}}{\partial p_{i-1,j}} - \left(\frac{b}{\mu}\right)_{i-\frac{1}{2},j}\frac{\partial \mu_{i-1,j}}{\partial p_{i-1,j}}\right] \tag{3.34}$$

$$\frac{\partial T_{i+\frac{1}{2},j}}{\partial p_{i+1,j}} = G_{i+\frac{1}{2},j}\frac{\Omega}{\mu_{i+\frac{1}{2},j}}\left[\frac{\partial b_{i+1,j}}{\partial p_{i+1,j}} - \left(\frac{b}{\mu}\right)_{i+\frac{1}{2},j}\frac{\partial \mu_{i+1,j}}{\partial p_{i+1,j}}\right] \tag{3.35}$$

$$\frac{\partial T_{i,j\pm\frac{1}{2}}}{\partial p_{i,j\pm1}} = G_{i,j\pm\frac{1}{2}}\left[\frac{(\mu_{i,j+1}+\mu_{i,j})\frac{\partial b_{i,j\pm1}}{\partial p_{i,j\pm1}} + (b_{i,j+1}+b_{i,j})\frac{\partial \mu_{i,j\pm1}}{\partial p_{i,j\pm1}}}{(\mu_{i,j+1}+\mu_{i,j})^2}\right] \tag{3.36}$$

Resultando em uma matriz com aparência igual ao da Transmissibilidade.

3.3 Identificação de pacotes – assuntos

A partir da analise dos modelos apresentados, pode-se identicar os seguintes assuntos/ pacotes:

- Engenharia de Reservatórios: este pacote recebe arquivos digitados pelo usuário ou os lê de um arquivo de extensão .txt. Nele, os dados se separam, de acordo com suas característica: rocha, fluido, aquífero, dados de produção, dados de injeção. Quando juntas fornecem uma caracterização do reservatório como um todo e servem de base para os cálculos da simulação.

- Simulador: Relaciona os pacotes, sendo responsavel pela criação e destruição de objetos, assim como interagir com o usuário através de um interface via texto para denir todas ações a serem tomadas.

- Modelagem Numérica Computacional: contém os algoritmos matemáticos necessários para a solução do modelo do simulador, como por exemplo, o Método de Newton-Raphson. Este pacote está separado do simulador, pois um dos objetivos da AOO é ter uma maior reusabilidade do código, assim, estando separados, é possível aplicar este mesmo pacote para outros problemas de engenharia, como por exemplo o de análise de teste de pressão.

- Termodinâmica: pacote que envolve todos os conceitos físicos (efeitos de mudanças na temperatura, pressão, volume e outras propriedades termodinâmicas de um sistema) sendo necessário no desenvolvimento de um simulador de reservatório devido ao dinamismo do comportamento dos fluidos.

- Álgebra linear e Cálculo Integral e Diferencial: pacote com deduções matématicas, teoremas, e e deduções que são a base de todo o processo.

- Pacote Gráfico: é um pacote que utiliza o gnuplot para plotar as soluções numéricas obtidas, isto é, as distribuições de pressão. Em outras palavras, é o software gnuplot que implementa a saída gráfica dos dados calculados.

- Biblioteca: Serão utilizadas métodos e funções já existentes para a resolução de sistemas de matrizes, bibliotecas padrão de C++ tais como (STL) e bibliotecas como a iostream, iomanip, etc.

3.4 Diagrama de pacotes – assuntos

O diagrama de pacotes da Figura 3.3 mostra as relações existentes entre os pacotes deste software.

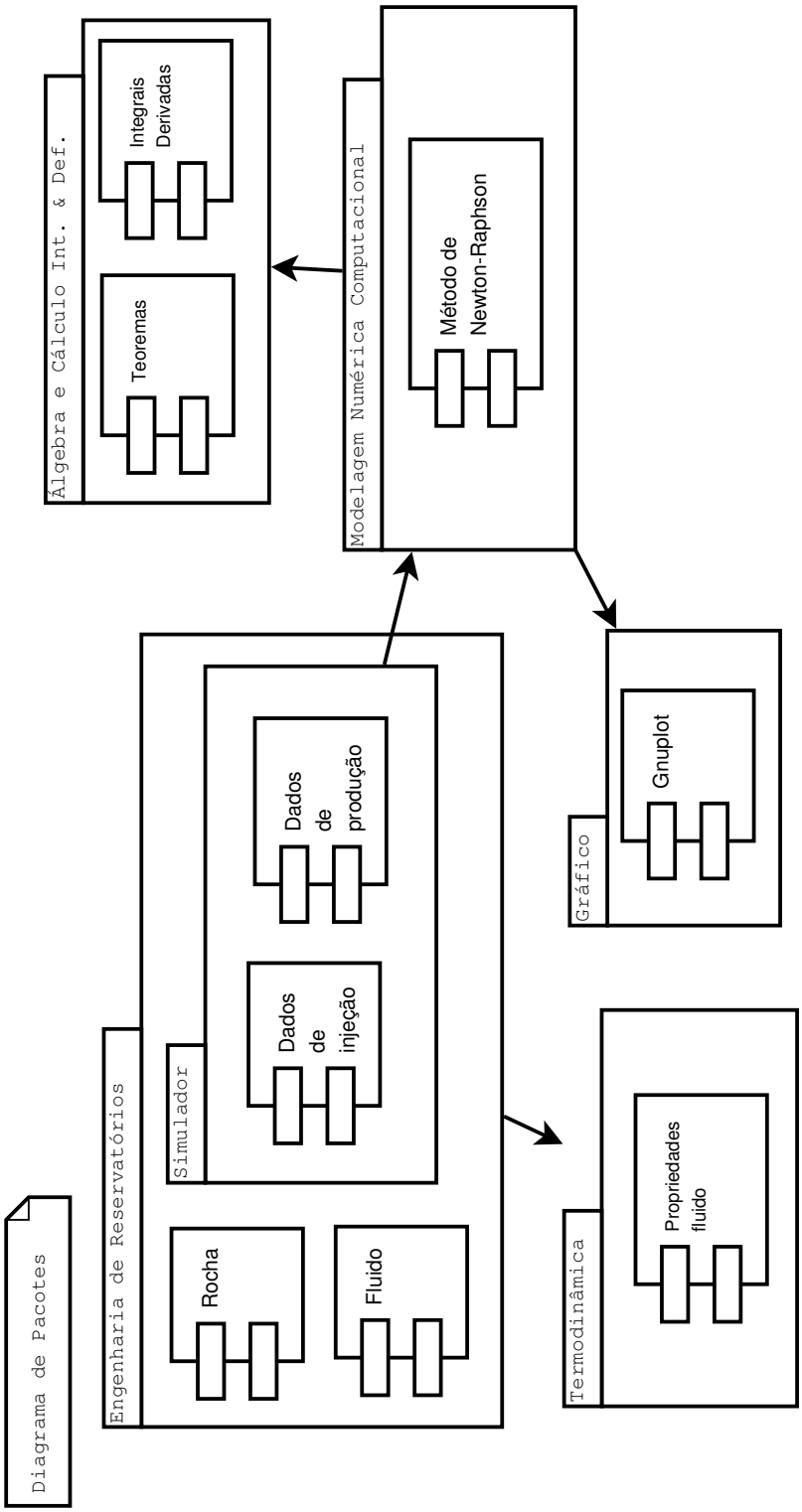

Figura 3.3: Diagrama de Pacotes.

Capítulo 4

AOO – Análise Orientada a Objeto

A AOO – Análise Orientada a Objeto é a terceira etapa do desenvolvimento de um projeto de engenharia, neste caso um software aplicado a engenharia de petróleo. Ela utiliza algumas regras para identificar os objetos de interesse, as relações entre os pacotes, as classes, os atributos, os métodos, as heranças, as associações, as agregações, as composições e as dependências. O modelo de análise enfatiza o que deve ser feito e não como foi realizado.

Nas próximas seções, serão apresentados um conjunto de cinco diagramas (de classes, de sequência, de comunicação, de máquina de estado e de atividades) com o objetivo de identificar os objetos e seus relacionamentos e assim visualizar o software de várias formas.

4.1 Diagramas de classes

O diagrama de classes é essencial para a montagem da versão inicial do código do software. Ele é constituído pelas classes, seus métodos e atributos, além das diversas relações entre elas (herança, dependência, nível de acesso). Então, o diagrama aqui desenvolvido é composto por 10 classes e é apresentado na Figura 4.1 que se segue.

/Users/Usuario/Desktop/Programaca

4.1.1 Dicionário de classes

- Classe **CGas**: gás que satura o meio poroso, contendo as propriedades físicas características. Sua função é fornecer informações para compor um meio poroso saturado. Classe filha de CFluido, ou seja, herda propriedades e métodos da classe mãe, e contém métodos e propriedades próprias. Cabe destacar, fator de compressibilidade e constante universal dos gases.

- Classe **CLiquido**: líquido que satura o meio poroso, contendo as propriedades físicas características. Sua função é fornecer informações para compor um meio poroso saturado. Classe filha de CFluido.Classe filha de CFluido, ou seja, herda propriedades e métodos da classe mãe, e contém métodos e propriedades próprias.

- Classe **CFluido**: classe virtual que representa o fluido que satura o meio poroso. Basicamente, por meio de uma classe virtual, ao se criar uma subclasse é possível torná-la mais específica não se precisando reimplementar toda a classe, pois é possível alterar o comportamento pontualmente.

- Classe **CResevoir:** representa uma rocha reservatório com atributos específicos do reservatório, como pressão inicial, raio externo, compressibilidade, porosidade, temperatura, permeabilidade.

- Classe **CProps**: classe que recebe características do fluido e do reservatório e calcula propriedades de interesse. Tudo que foi armazenado é acessado dinamicamente. É a base de cálculo do método numérico por implementa as derivadas e possui métodos de atualização do conteúdo das células discretizadas.

- Classe **CGrid**: classe que representa o meio poroso como um domínio discretizado, ou seja, fornece o dimensionamento do poço e reservatório no espaço, uma grade propriamente dita. Sua função é identificar os pontos no espaço em que a solução em volumes finitos será calculada.

- Classe **CDiscretization:** armazena propriedades puras da simulação. Dito de outra forma, propriedades da malha que não dependem da tempo e que são estáticas.

- Classe **CWell**: classe que representa o poço.

- Classe **CGnuplot**: classe que fornece os métodos necessários para a geração de gráficos. Sucintamente, o programa externo Gnuplot é uma ferramenta utilizada para a geração dos gráficos da distribuição de pressão, em função do tempo obtida pelo simulador para o poço e reservatório.

- Classe **CSimulator**: uma classe que representa o simulador, "cérebro do software". Ela quem dita as regras e comanda quais ações serão tomadas e em qual ordem. Em destaque, possui o método Run, que dá o ponta pé inicial na resolução pelo

método numérico. Como já foi dito, é um método cuja discretização da EBM pode ser resolvida implicitamente, obtendo a distribuição de pressões. Constantemente acessa a Classe CProps com valores em atualização a cada passo de tempo.

4.2 Diagrama de sequência – eventos e mensagens

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

4.2.1 Diagrama de sequência geral

O diagrama de sequência geral do software é mostrado na Figura 4.2 que se segue:

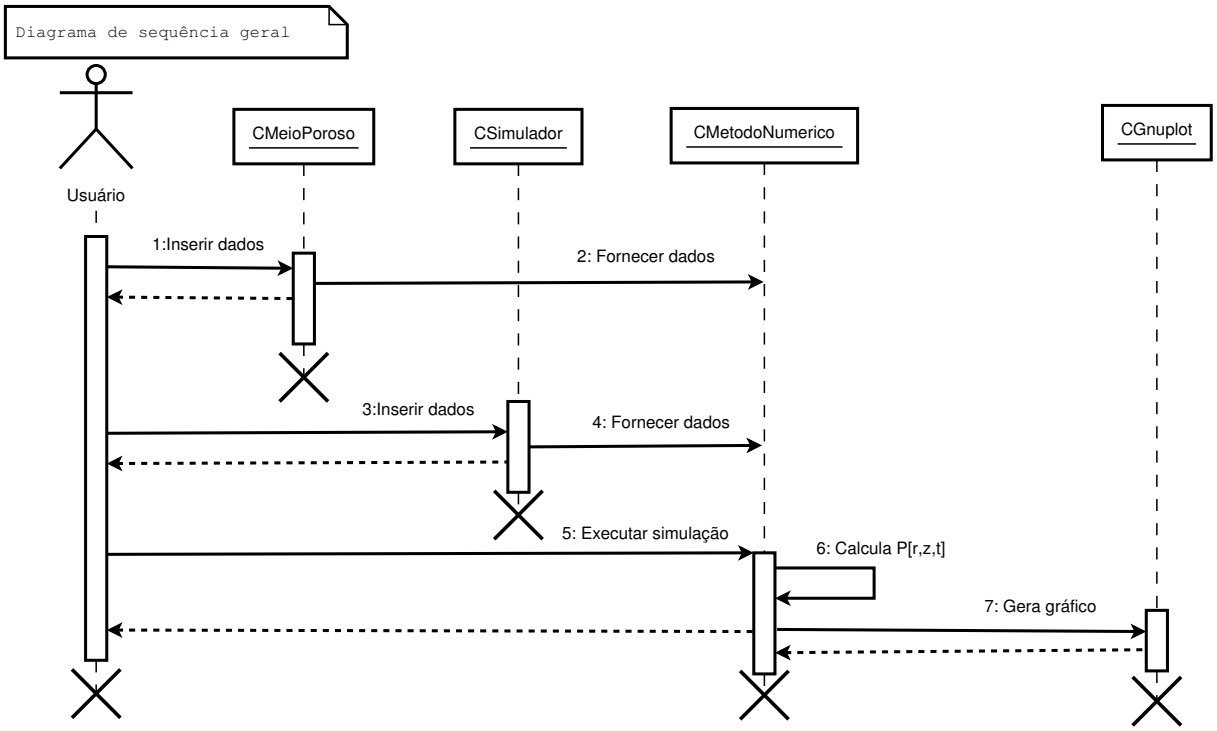

Figura 4.2: Diagrama de sequência

4.2.2 Diagrama de sequência específico

O diagrama de sequência específico é mostrado na Figura 4.3 abaixo.

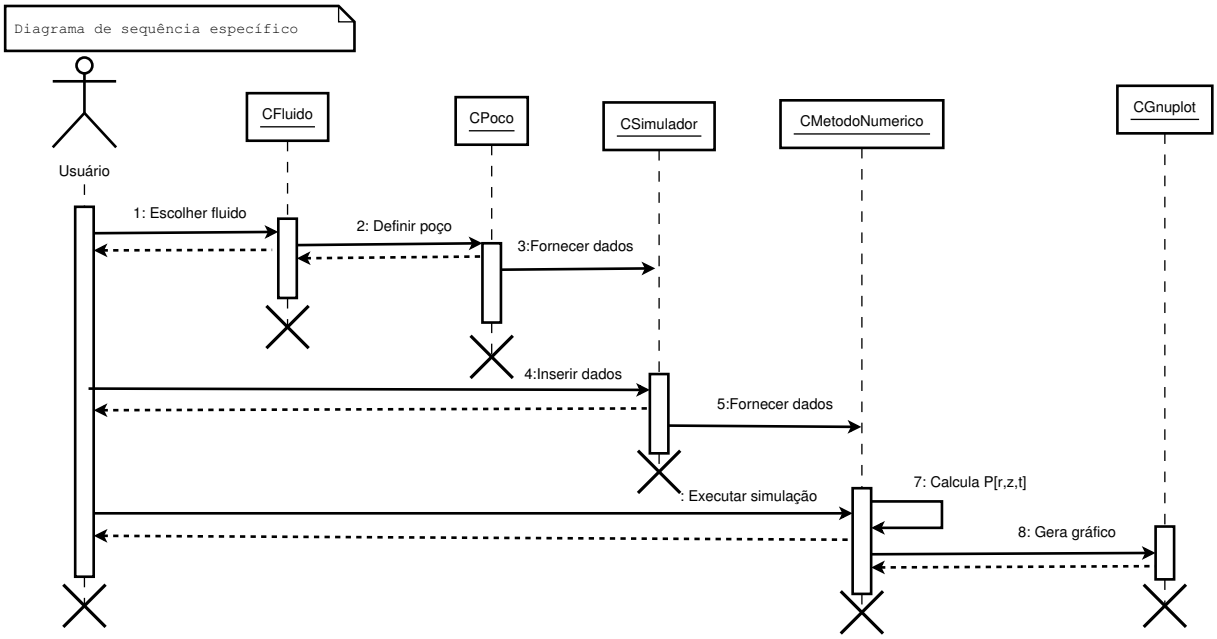

Figura 4.3: Diagrama de sequência

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. O usuário está sempre informando ao computador dados que são necessários para o processamento da simulação. Aqui a ênfase é o entendimento das mensagens que chegam e saem de cada objeto.

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

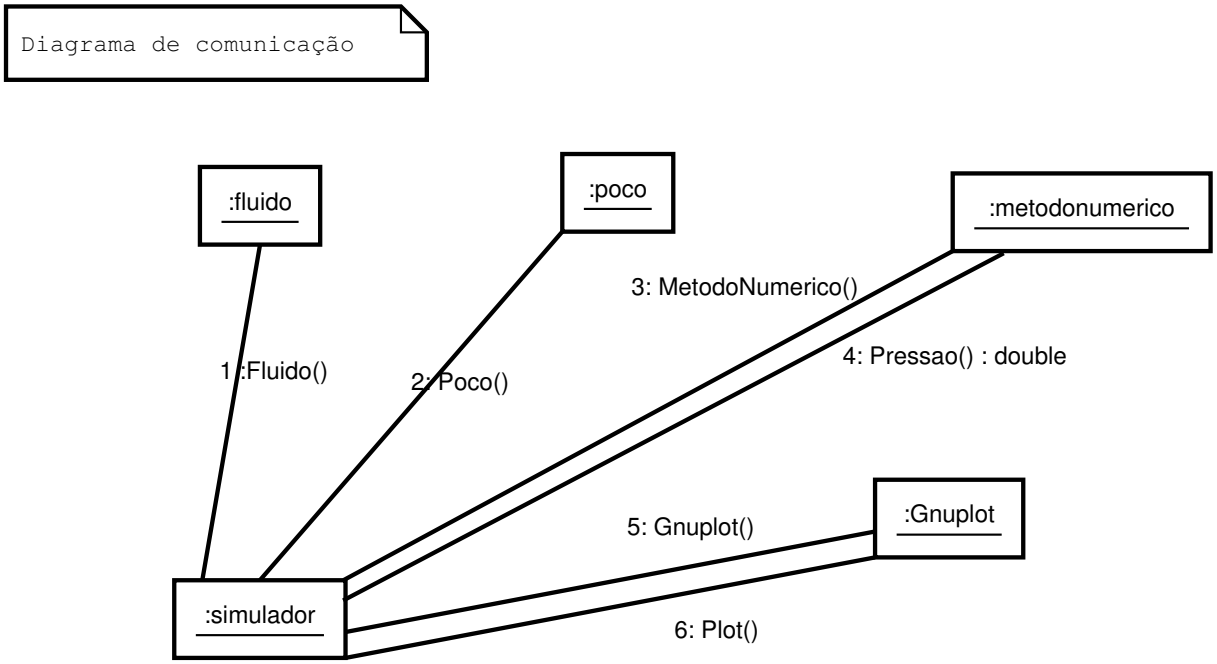

Figura 4.4: Diagrama de comunicação

Por sua vez, nesse segundo diagrama apresentado na Figura 4.3, tem-se a completa comunicação entre os sistemas envolvidos. Observa-se que o primeiro passo é escolher o tipo de fluido, depois construir o poço com zonas abertas ao fluxo ou não. A seguir, fornecer dados ao simulador. Com todos os parâmetro já definidos e alocados na memória o computador processa-os a partir do objeto CMetodoNumerico. A comunicação continua com a exibição dos resultados para o usuário e com o fornecimento deles para o Gnuplot gerar os respectivos gráficos e fornecê-los ao usuário.

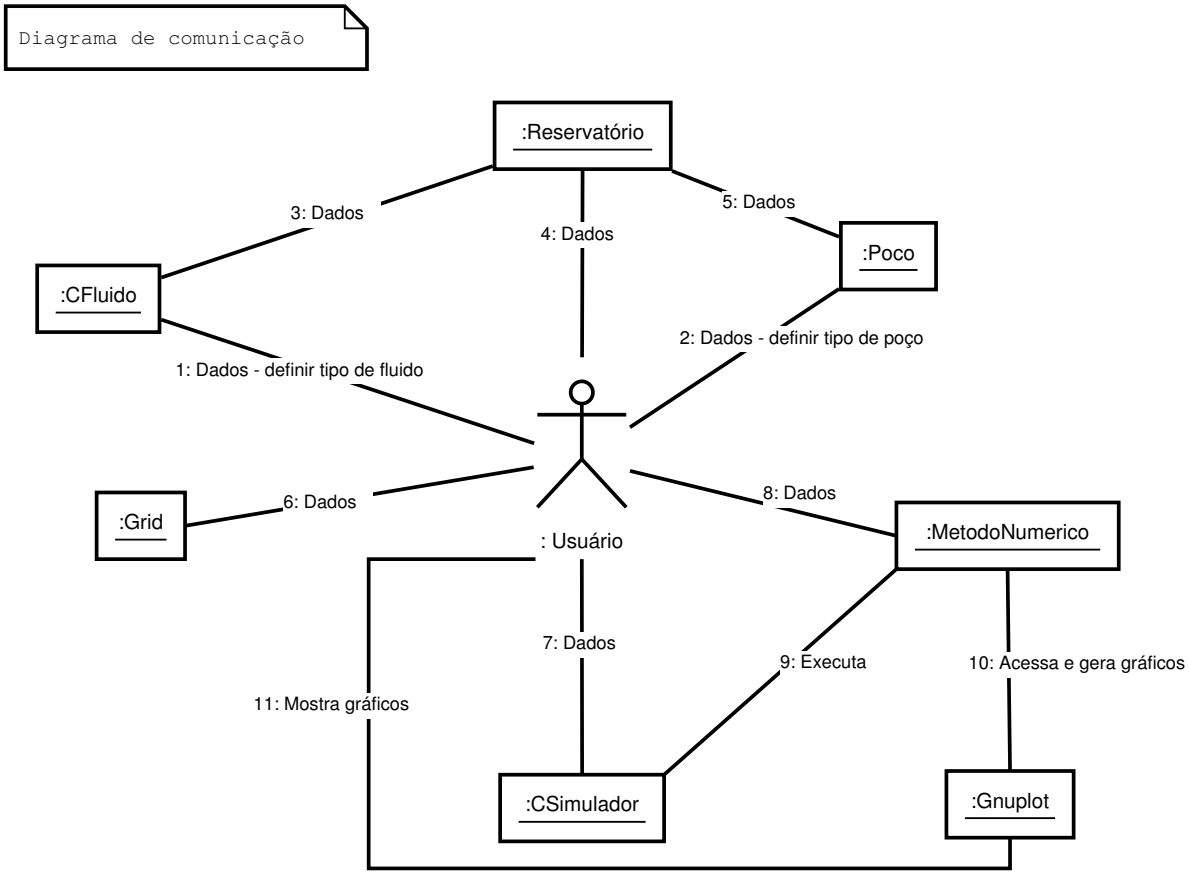

Figura 4.5: Diagrama de comunicação

4.4 Diagrama de máquina de estado

Um diagrama de máquina de estado representa os diversos estados que o objeto assume e os eventos que ocorrem ao longo de sua vida ou mesmo ao longo de um processo (histórico do objeto). É usado para modelar aspectos dinâmicos do objeto. Veja na Figura 4.6 o diagrama de máquina de estado para o objeto CSimulador.

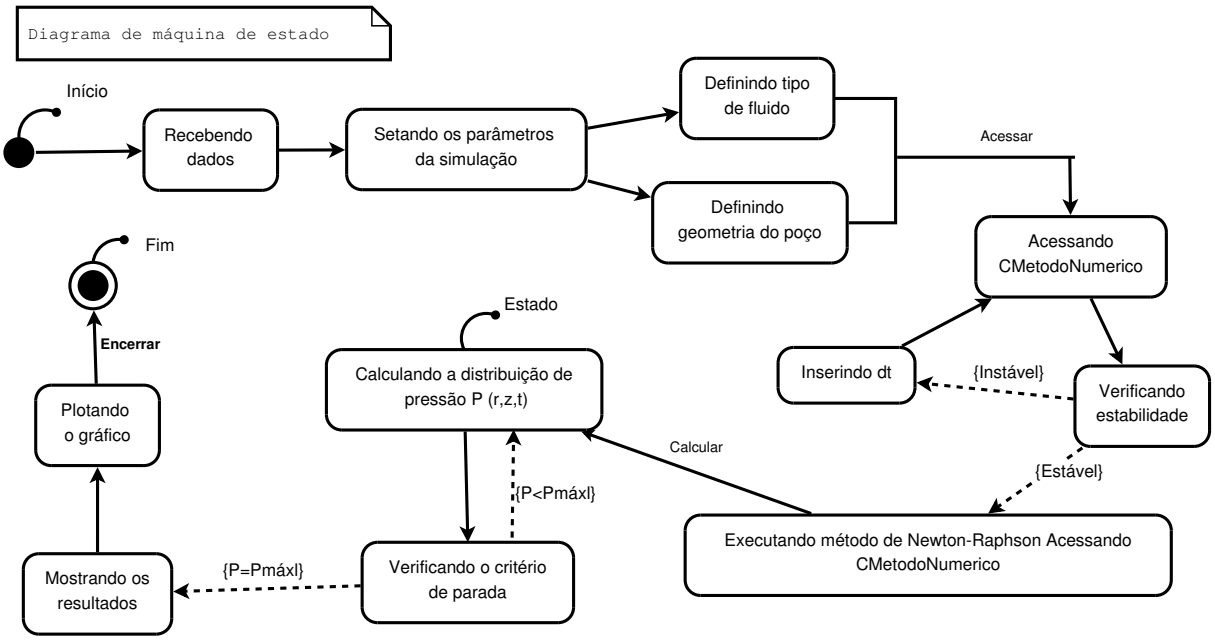

Figura 4.6: Diagrama de máquina de estado

4.5 Diagrama de atividades

Veja na Figura 4.7 que o diagrama de atividades correspondente a uma atividade específica do diagrama de máquina de estado. Nesse caso, "calculando a distribuição de pressão $P\ (r, z, t)$".

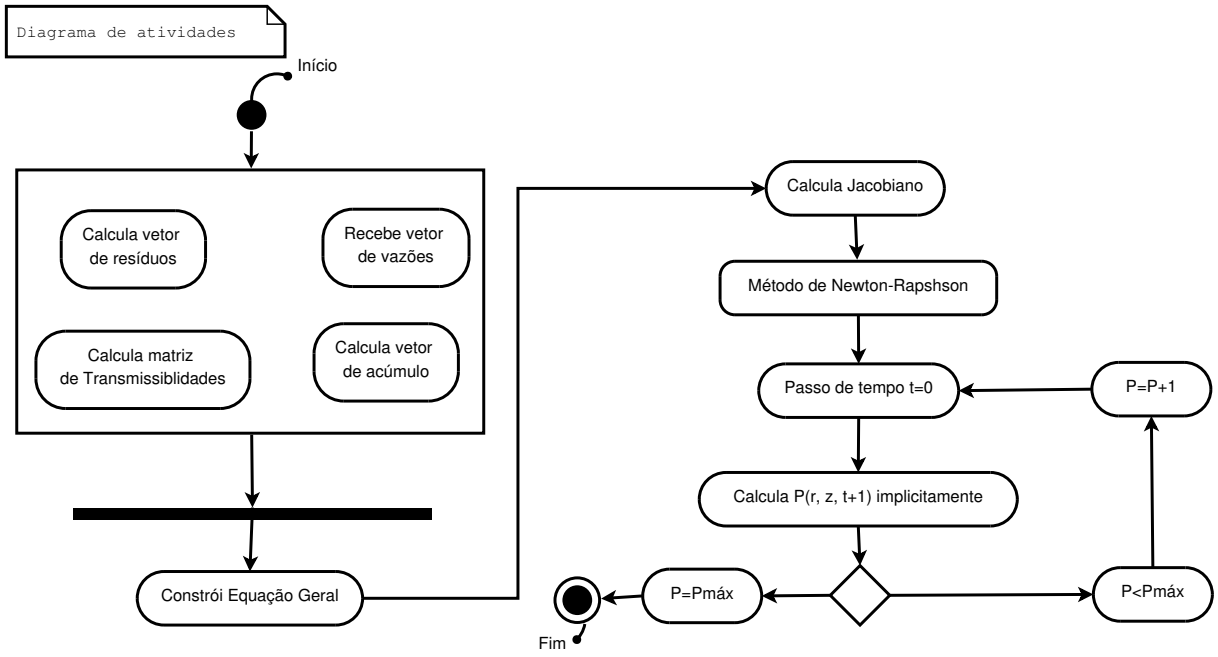

Figura 4.7: Diagrama de atividades

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.

A importância de se definir padrões específicos de documentação, nome das classes, padrões de retorno, interface do usuário e características de desempenho constitui-se como uma ferramente estratégica para resolver o problema, elaborar uma solução e garantir repetibilidade e expansão para problemas similares.

Nessa etapa será avaliado algumas características do *software* tais como:

1. Protocolos

    - O programa permite salvar dados em disco no formato aberto, como .txt.
    - Será efetuada a entrada de dados via arquivo de texto .txt.
    - Neste projeto o software irá se comunicar com o componente externo Gnuplot que gera gráficos.

2. Recursos

    - O programa utilizará uma máquina computacional com HD, CPU, RAM, periféricos, processador, teclado para a entrada de dados e o monitor para a saída

de dados.

- O simulador utiliza o programa externo Gnuplot. Que por sua vez plota os objetos da imagem rotulados e ajustados as formas geométricas.

3. Controle

   - Este software requer um controle sequencial.

4. Plataformas

   - O programa é multiplataforma, o que permite executá-lo em Windows e Mac OS X , mas será desenvolvido na plataforma Windows. A linguagem de software utilizada é a C++ orientada a objeto.
   - Ambiente de desenvolvimento Microsoft Visual C ++ (MSVC). Um editor de códigos e compilador de diversas linguagens de programação como python, C, C++, C#, desenvolvido pela Microsoft . MSVC é um software proprietário. Originalmente um produto autônomo que mais tarde tornou-se parte do Visual Studio e foi disponibilizado em versões de teste e *freeware*. Ele apresenta ferramentas para desenvolver e depurar código C ++, especialmente código escrito para a API do Windows , DirectX e .NET
   - O software utilizara a biblioteca externa CGnuplot que permite acesso ao programa Gnuplot. É um utilitário portátil de gráfico baseado em linha de comando para Linux, OS / 2, MS Windows, OSX, VMS e muitas outras plataformas. O código-fonte é protegido por direitos autorais, mas é distribuído gratuitamente.
   - Características do hardware:

5. Padrões de projeto

   - Não se aplica para esse caso já que o software foi feito para cunho acadêmico e não empresarial.

5.2 Projeto orientado a objeto – POO

O projeto orientado a objeto é a etapa posterior ao projeto do sistema. Baseia-se na análise, mas considera as decisões do projeto do sistema. Acrescenta a análise desenvolvida e as características da plataforma escolhida (hardware, sistema operacional e linguagem de software). Passa pelo maior detalhamento do funcionamento do software, acrescentando atributos e métodos que envolvem a solução de problemas específicos não identificados durante a análise.Além disso, 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.
- O código possui comentários com explicações dos algoritmos a serem executados.
- Neste projeto foi feita uma associação entre a biblioteca CGnuplot com as classes CSimulator, que por sua vez associa-se com as classes CReservoir, CDiscretization, CGrid, CWell, CProps, CFluido, que se associação com CGas e CLiquido

Efeitos do projeto no modelo dinâmico

- Não foi realizada nessa etapa do projeto uma vez que os diagramas de sequência, de comunicação, máquina de estado e de atividades serão modicados durante o desenvolvimento do código caso seja necessário.

Efeitos do projeto nos atributos

- Como alguns atributos necessitavam de constante atualização, foi implementado uma função chamada update, que a cada passa de tempo, recalculava as propriedades do sistema reservatório e poço.
- As relações entre classes foram melhoradas e adaptadas em relação à herança entre CFluido com CGas e CLiquido. Basicamente, foi necessário melhor especificar com base nos atributos inerentes de cada uma delas.

Efeitos do projeto nos métodos

- Em virtude de usar leitura de disco, um método de inserção de dados através do teclado foi adicionado.

- A razão da existência do método Run() presente em CSimulador se explica pela intenção em deixar o código mais enxuto e pela intenção de agrupar algoritmos de mesma natureza em um só método. Ele é chamado e governa toda a execução do código.

Efeitos do projeto nas heranças

- Sempre que um ou mais atributos e métodos se repetiam, foi necessário reavaliar o código. Desse modo, o item de ação voltou-se para o diagrama de classes que foi reformulado algumas vezes em subdivisões de classes e criação de novas classes.
- Algumas heranças puderam ser excluídas do diagrama, uma vez que alguns atributos necessários inicialmente puderam ser passados através da chamada das funções.

Efeitos do projeto nas associações

- Algumas heranças foram trocadas por associações e novas associações foram criadas para relacionamento com novas classes

Efeitos do projeto nas otimizações

- Logo no início optou-se por pedir todas as informações ao usuário juntas.
- O software tem opção de parada para mudança de valores e depois retomada.
- Pode ser otimizado pela implementação de processamento paralelo a fim de utilizar melhor a capacidade de processamento da máquina.
- Possibilidade de inclusão de bibliotecas otimizadas para resolução do Método de Newton-Raphson e sistema linear.
- A Classe CProps foi criada para reunir alguns atributos que estavam presentes na maioria dos cálculos e que necessitavam de atualizações constantes. Por conta disso eles foram retirados das respectivas classes e implementados em outra.

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. De posse do diagrama de componentes, temos a lista de todos os arquivos necessários para compilar e rodar o software. Podemos perceber as dependências de cada componente. Por exemplo, o componente CSimulator depende de todos outros para funcionar. E por sua vez, como exemplo específico, o CFluido para funcionar, depende do CGas ou CLiquido.

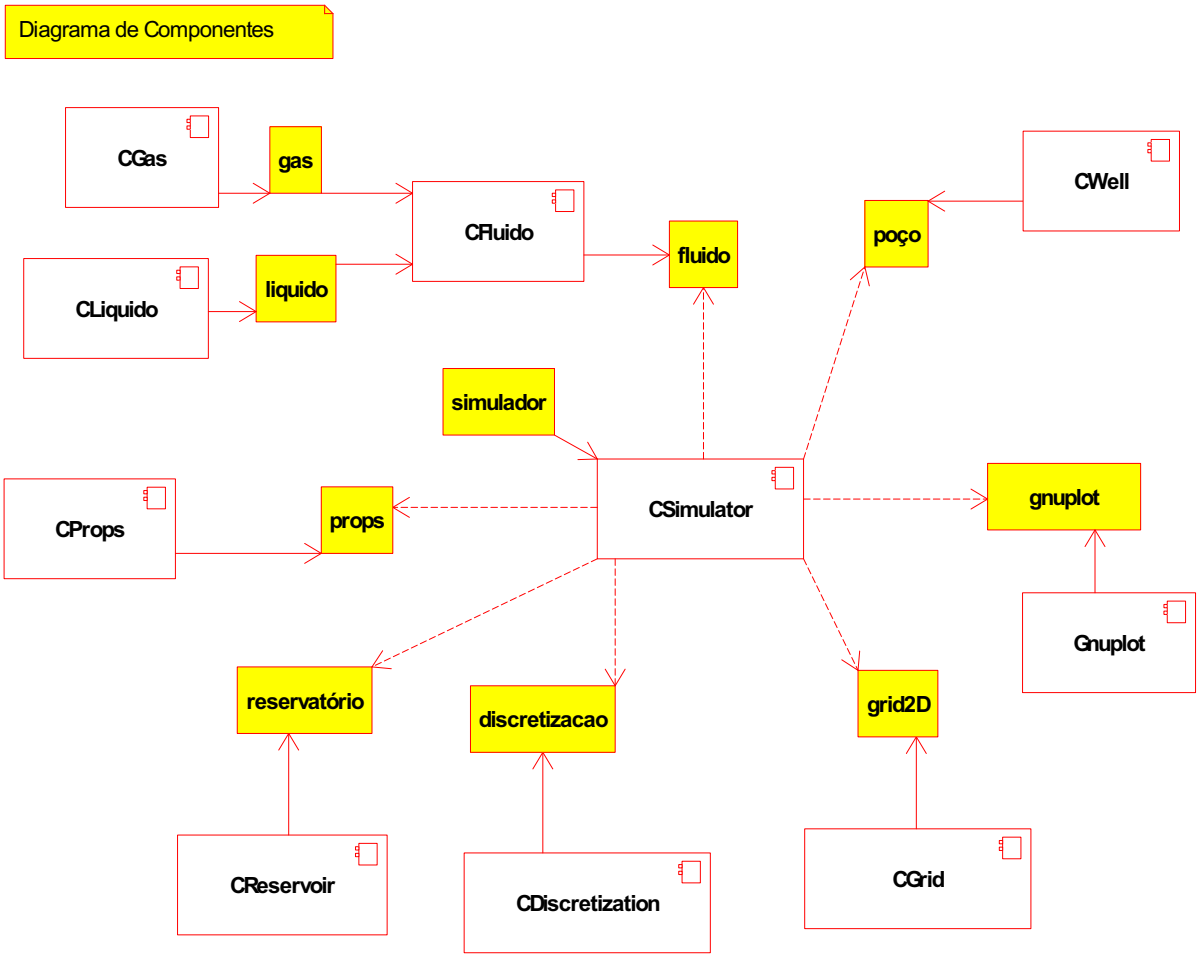

Figura 5.1: Diagrama de componentes

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. Este 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 (CPU, RAM, HD) e uma aquisição de dados, sendo requeridos dados do poço e dados de reservatório.

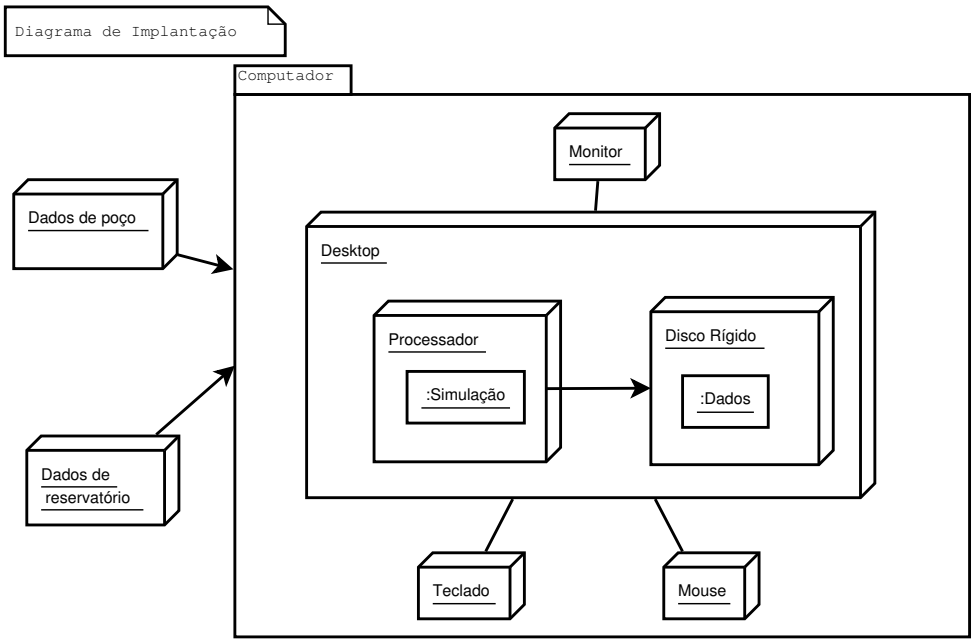

Figura 5.2: Diagrama de implantação

Capítulo 6

Implementação

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

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

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

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

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: Arquivo de implementação da função main().

```
1 #include "CSimulador.hpp"
2
3 int main() {
4         CSimulador* simulator = new CSimulador;
5
6         simulator->run();
7         return 0;
```

```
8 }
```

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

Listing 6.2: Arquivo de cabeçalho da classe CWell.

```
1 #ifndef CWELL_HPP
2 #define CWELL_HPP
3
4 #include <vector>
5 #include<iostream>
6
7 class CWell {
8
9 private:
10         std::vector<double> tp{0, 300};                    /// tempos
              de mudanca na vazao na superficie [h]
11         std::vector<double> qsc{0, -300 };                 /// vazoes
              nos tempos de mudanca [m^3 std / dia]
12         std::vector<double> dz{ 1,1 };                     /// altura
              de cada delta z
13         std::vector<double> partial{1,1};                  ///
              porcentagem de abertura de cada delta z
14         double rw{ 0.09486 };                                      ///
               raio do poco [m]
15
16 public:
17         CWell() {}
18         CWell(std::vector<double> _tp, std::vector<double> _qsc,
              std::vector<double> _dz, std::vector<double> _partial,
              double _rw) : tp{ _tp }, qsc{ _qsc }, dz{ _dz }, partial
              { _partial }, rw{ _rw }{}
19         std::vector<double> get_tp() { return tp; }
20         std::vector<double> get_qsc() { return qsc; }
21         std::vector<double> get_dz() { return dz; }
22         std::vector<double> get_partial() { return partial; }
23         double get_rw() { return rw; }
24         double get_qsc(double time);
25         double get_partial(int i) { return partial[i]; }
26         void print();
27 };
28 #endif
```

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

CWell.

Listing 6.3: Arquivo de implementação da classe CWell.

```
#include "CWell.hpp"

double CWell::get_qsc(double t) {
        bool end = true;
        double q;
        for (int i = 0; i < tp.size(); i++) {
                if (t <= tp[i]) { // esse 1 eh so para gerar uma
                   margem, eh muito literal o igual
                        q = qsc[i];
                        end = false;
                }
        }
        return end ? qsc[qsc.size() - 1] : q;
}

void CWell::print() {
        std::cout << "\nObjeto␣Well" << std::endl;
        std::cout << "tp:␣";
        for (int i = 0; i < tp.size(); i++) {
                std::cout << tp[i] << "␣␣";
        }

        std::cout << "\nqsc:␣";
        for (int i = 0; i < qsc.size(); i++) {
                std::cout << qsc[i] << "␣␣";
        }
        std::cout << "\ndz␣|␣partial:␣\n";
        for (int i = 0; i < dz.size(); i++) {
                std::cout << dz[i] << "␣|␣" << partial[i] << std::
                   endl;
        }
        std::cout << "rw:␣" << rw << std::endl;
}
```

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

Listing 6.4: Arquivo de cabeçalho da classe CSimulador.

```
#ifndef CSIMULADOR_HPP
#define CSIMULADOR_HPP

#include <vector>
```

```
5#include <string>
6#include <iostream>
7#include <iomanip>
8#include <ctime>
9#include <fstream>
10
11#include "CGas.hpp"
12#include "CWell.hpp"
13#include "CGrid.hpp"
14#include "CProps.hpp"
15#include "CMatrix.hpp"
16#include "CFluido.hpp"
17#include "CGnuplot.hpp"
18#include "CLiquido.hpp"
19#include "CReservoir.hpp"
20#include "CDiscretization.hpp"
21
22class CSimulador {
23public:
24        CSimulador();
25
26        void run();
27private:
28        const double PI = 3.141592;
29        CWell* well;
30        CGrid* grid;
31        CFluido* fluido;
32        CReservoir* reservoir;
33        CDiscretization* discretization;
34
35        std::vector<std::vector<double>> Pressure;      /// todas
            as pressoes em todo o tempo
36        std::vector<double> WellPressure;                    ///
             todas as pressoes do poco
37
38private:
39        std::vector<double> calc_H(CGrid* grid, CProps* props_n,
            CProps* props_nu, double dt);
40        std::vector<double> calc_Q(CGrid* grid, CWell* well, double
             time);
41        std::vector<double> calc_X(double pw, std::vector<double> p
            );
```

```
42          std::vector<std::vector<double>> calc_T(CGrid* grid, CProps
               * props_n);
43
44          std::vector<std::vector<double>> calc_eta(CGrid* grid,
               CProps* props_nu, double dt);
45          std::vector<std::vector<double>> calc_tau(CGrid* grid,
               CWell* well, CProps* props_nu, std::vector<double> p_nu,
                double pw_nu);
46
47          void plot(std::vector<double> time, std::vector<double> Pw)
               ;
48
49          bool isErrorNotAcceptable(double dt, std::vector<double> R,
                CGrid* grid, CProps* props_nu, double q,
               CDiscretization* discretization);
50
51          void read_data_and_start_objects(std::string nameFile);
52 };
53 #endif
```

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

Listing 6.5: Arquivo de implementação da classe CSimulador.

```
1 #include "CSimulador.hpp"
2
3 CSimulador::CSimulador() {
4          std::cout << "Nome␣do␣arquivo:␣";
5          std::string nameFile;
6          nameFile = "input.dat";
7          std::cout << nameFile << std::endl;
8          //std::cin >> nameFile;
9          read_data_and_start_objects(nameFile);
10 }
11
12 void CSimulador::run() {
13          CProps* props_n  = new CProps(grid, fluido, reservoir,
               discretization);
14          CProps* props_nu = new CProps(grid, fluido, reservoir,
               discretization);
15          std::vector<double> time = grid->get_time();
16
17          /// var de ajuda
```

```
18          int nr = discretization->get_nr();
19          int nz = discretization->get_nz();
20          int iteracoes = 0;
21          double erro_MB = 1;
22          double erro_NR = 1;
23          double dt;
24
25          /// variaveis das pressoes - pressao no tempo anterior /
                pressao iteracao n / pressao iteracao n+1
26          double pw_n              = reservoir->get_p_i();
27          double pw_nu    = reservoir->get_p_i();
28          std::vector<double> Pw(time.size());
29          std::vector<double> p_n(nr * nz, reservoir->get_p_i());
30          std::vector<double> p_nu(nr * nz, reservoir->get_p_i());
31
32          /// comeco a preparar o loop
33          std::vector<double> H;
34          std::vector<double> Q;
35          std::vector<double> X;
36          std::vector<std::vector<double>> T;
37          std::vector<double> R(nr*nz+1);
38
39          std::vector<std::vector<double>> eta(nr*nz+1, std::vector<
                double>(nr*nz+1));
40          std::vector<std::vector<double>> tau(nr * nz + 1, std::
                vector<double>(nr * nz + 1));
41          std::vector<std::vector<double>> J(nr * nz + 1, std::vector
                <double>(nr * nz + 1));
42          std::vector<double> dX;
43
44          Pw[0] = reservoir->get_p_i();
45
46          clock_t begin_time;
47          /// loop do tempo
48          for (unsigned int t = 1; t < time.size(); t++) {
49                  dt = time[t] - time[t - 1];
50
51                  props_n->update(pw_n, p_n);
52
53                  begin_time = clock();
54
55                  do {
```

```
56                      props_nu->update(pw_nu, p_nu);
57
58                      H = calc_H(grid, props_n, props_nu, dt);
59                      Q = calc_Q(grid, well, time[t]);
60                      X = calc_X(pw_nu, p_nu);
61                      T = calc_T(grid, props_nu);
62
63                      for (int i = 0; i < nr * nz + 1; i++) { ///
                            a linha da multiplicacao
64                          R[i] = 0.0;
65                          for (int j = 0; j < nr * nz + 1; j
                                ++) /// os termos da
                                multiplicacao
66                              R[i] += T[i][j] * X[j];
67                          R[i] += Q[i] - H[i];
68                      }
69
70
71                      /// matriz jacobiana
72                      eta = calc_eta(grid, props_nu, dt);
73                      tau = calc_tau(grid, well, props_nu, p_nu,
                            pw_nu);
74
75                      for (int i = 0; i < nr * nz + 1; i++) /// a
                            linha da multiplicacao
76                          for (int j = 0; j < nr * nz + 1; j
                                ++) /// os termos da
                                multiplicacao
77                              J[i][j] = T[i][j] + tau[i][
                                    j] - eta[i][j];
78
79                      dX = CMatrix::LU_solver(J, R);
80                      //dX = CMatrix::GaussSolver(R, J); ///
                            calculo a solucao do sistema linear
81
82                      pw_nu = pw_nu + dX[0];
83                      for (int i = 0; i < nr * nz; i++)
84                          p_nu[i] = p_nu[i] + dX[i+1];
85
86                      iteracoes++;
87                  } while (iteracoes < 10 && isErrorNotAcceptable(dt,
                        R, grid, props_nu, Q[0], discretization));
```

```
88
89                  std::cout << "Tempo: " << time[t] << " - iteracoes:
                        " << iteracoes << " - duracao: " << float(clock
                        () - begin_time) / CLOCKS_PER_SEC << std::endl;
90                  p_n = p_nu;
91                  pw_n = pw_nu;
92                  iteracoes = 0;
93                  Pw[t] = pw_nu;
94                  Pressure.push_back(p_n);
95          }
96          CMatrix::mostrarMatriz(Pw, "Pw");
97          plot(time, Pw);
98 }
99
100 std::vector<double> CSimulador::calc_H(CGrid* grid, CProps* props_n
      , CProps* props_nu, double dt) {
101         std::vector<double> H(grid->get_ntotal()+1,0.0);
102         for (int i = 0; i < grid->get_ntotal(); i++)
103                 H[i+1] = grid->get_Vb_ac(i) * (props_nu->get_b(i)*
                        props_nu->get_phi(i) - props_n->get_b(i)*
                        props_n->get_phi(i)) / dt;
104         return H;
105 }
106
107 std::vector<double> CSimulador::calc_Q(CGrid* grid, CWell* well,
      double time) {
108         std::vector<double> Q(grid->get_ntotal() + 1, 0.0);
109         Q[0] = well->get_qsc(time) * (grid->get_dtheta() / (2 * PI)
              );
110         return Q;
111 }
112
113 std::vector<double> CSimulador::calc_X(double pw, std::vector<
      double> p) {
114         std::vector<double> X(p.size() + 1);
115         X[0] = pw;
116         for (unsigned int i = 1; i < X.size(); i++)
117                 X[i] = p[i-1];
118         return X;
119 }
120
121 std::vector<std::vector<double>> CSimulador::calc_T(CGrid* grid,
```

```
    CProps* props_n) {
122         int nt = grid->get_ntotal()+1;
123         int nr = grid->get_nr();
124         int nz = grid->get_nz();
125         int i;
126 
127         double Right;
128         double Left;
129         double Bottom;
130         double Top;
131         double Well;
132 
133         std::vector<std::vector<double>> T(nt, std::vector<double>(
                nt, 0.0));
134 
135         T[0].resize(nt, 0.0);
136         for (int k = 1; k < nt; k++) { // rodo sem o poco
137                 i = k - 1;
138                 //T[k].resize(nt, 0.0);
139                 Right = 0.0;
140                 Left = 0.0;
141                 Bottom = 0.0;
142                 Top = 0.0;
143                 Well = 0.0;
144 
145                 /// passo por todas as colunas da matriz
146                 if (i - nr >= 0) {
147                         Top = grid->get_Gjmh(i) * props_n->get_bjmh
                                (i) / props_n->get_mujmh(i);
148                         T[k][k - nr] = Top;
149                 }
150                 if (i + nr < nt-1) {
151                         Bottom = grid->get_Gjph(i) * props_n->
                                get_bjph(i) / props_n->get_mujph(i);
152                         T[k][k + nr] = Bottom;
153                 }
154                 if ((i - 1) % nr >= 0) {
155                         Left = grid->get_Gimh(i) * props_n->
                                get_bimh(i) / props_n->get_muimh(i);
156                         T[k][k - 1] = Left;
157                 }
158                 if ((i + 1) % nr > 0) {
```

```
159                         Right = grid->get_Giph(i) * props_n->
                               get_biph(i) / props_n->get_muiph(i);
160                         T[k][k + 1] = Right;
161                 }
162                 if (i % nr == 0) {
163                         Well = well->get_partial((int)i/nr) * grid
                               ->get_Gw((int)i/nr) * props_n->get_bw()
                               / props_n->get_muw();
164                         T[k][0] = Well;
165                         T[0][k] = Well;
166                         T[0][0] -= Well;
167                 }
168                 T[k][k] = -Top - Bottom - Right - Left - Well;
169         }
170         return T;
171 }
172
173 std::vector<std::vector<double>> CSimulador::calc_eta(CGrid* grid,
      CProps* props_nu, double dt) {
174         int nt = grid->get_ntotal() + 1;
175         int nr = grid->get_nr();
176         int nz = grid->get_nz();
177
178         std::vector<std::vector<double>> eta(nt, std::vector<double
              > (nt, 0.0));
179         eta[0].resize(nt, 0.0);
180         for (int i = 1; i < nt; i++) {
181                 eta[i][i] = grid->get_Vb_ac(i-1)*(props_nu->get_b(i
                      -1)*props_nu->get_dphidp(i-1) + props_nu->
                      get_phi(i-1) * props_nu->get_dbdp(i-1)) / dt;
182         }
183         return eta;
184 }
185
186 std::vector<std::vector<double>> CSimulador::calc_tau(CGrid* grid,
      CWell* well, CProps* props_nu, std::vector<double> p_nu, double
      pw_nu) {
187         int nt = grid->get_ntotal();
188         int nr = grid->get_nr();
189         int nz = grid->get_nz();
190
191         double omega = grid->get_omega();
```

```
192
193         std::vector<std::vector<double>> tau(nt + 1, std::vector<
                double>(nt+1, 0.0));
194
195         /// bottom
196         for (int i = 0; i < nt - nr; i++) {
197                 tau[i + 1][i + 1 + nr] = (p_nu[i] - p_nu[i + nr]) *
                        grid->get_Gjph(i)
198                         * (props_nu->get_dbdp(i) * (props_nu->
                            get_mu(i + nr) + props_nu->get_mu(i))
199                                 - props_nu->get_dmudp(i) * (
                                    props_nu->get_b(i + nr) +
                                    props_nu->get_b(i)))
200                         / pow(props_nu->get_mu(i) + props_nu->
                            get_mu(i + nr), 2);
201                 tau[i + 1][i + 1] += tau[i + 1][i + 1 + nr];
202         }
203
204         /// top
205         for (int i = nr; i < nt; i++) {
206                 tau[i + 1][i + 1 - nr] = (p_nu[i] - p_nu[i - nr]) *
                        grid->get_Gjmh(i)
207                         * (props_nu->get_dbdp(i) * (props_nu->
                            get_mu(i - nr) + props_nu->get_mu(i))
208                                 - props_nu->get_dmudp(i) * (
                                    props_nu->get_b(i - nr) +
                                    props_nu->get_b(i)))
209                         / pow(props_nu->get_mu(i) + props_nu->
                            get_mu(i - nr), 2);
210                 tau[i + 1][i + 1] += tau[i + 1][i + 1 - nr];
211         }
212
213         /// left
214         for (int i = 1; i < nt; i++) {
215                 tau[i+1][i] = grid->get_Gimh(i) * (1.0 - omega) * (
                        p_nu[i - 1] - p_nu[i])
216                         * (props_nu->get_dbdp(i) / props_nu->
                            get_muimh(i)
217                                 - (props_nu->get_bimh(i) / pow(
                                    props_nu->get_muimh(i), 2)) *
                                    props_nu->get_dmudp(i - 1));
218                 tau[i+1][i+1] = tau[i+1][i];
```

```
219         }
220
221         /// right
222         for (int i = 1; i < nt - 1; i++) {
223                 tau[i][i + 1] = grid->get_Giph(i) * omega * (p_nu[i
                        + 1] - p_nu[i])
224                         * (props_nu->get_dbdp(i) / props_nu->
                            get_muiph(i)
225                                 - (props_nu->get_biph(i) / pow(
                                    props_nu->get_muiph(i), 2)) *
                                    props_nu->get_dmudp(i - 1));
226                 tau[i][i] = tau[i][i + 1];
227         }
228
229         /// well
230         double temp_well;
231         for (int i = 0; i < nz; i++) {
232                 tau[0][0] += (grid->get_Gw(i) / props_nu->get_muw()
                        ) * (props_nu->get_dbwdpw() - (props_nu->get_bw
                        () / props_nu->get_muw()) * props_nu->
                        get_dmuwdpw())*(p_nu[i*nr] - pw_nu) * well->
                        get_partial(i);
233                 temp_well = (grid->get_Gw(i) / props_nu->get_muw())
                        * (props_nu->get_dbwdp1() - (props_nu->get_bw()
                        / props_nu->get_muw()) * props_nu->get_dmuwdp1
                        ()) * (p_nu[i * nr] - pw_nu) * well->get_partial
                        (i);
234                 tau[0][i * nr + 1] += temp_well;
235                 tau[0][0] -= temp_well;
236                 tau[i * nr + 1][i * nr + 1] -= temp_well;
237                 tau[i * nr + 1][0] += (grid->get_Gw(i) / props_nu->
                        get_muw()) * (props_nu->get_dbwdp1() - (props_nu
                        ->get_bw() / props_nu->get_muw()) * props_nu->
                        get_dmuwdp1()) * (p_nu[i * nr] - pw_nu) * well->
                        get_partial(i);
238         }
239         return tau;
240 }
241
242 void CSimulador::plot(std::vector<double> time, std::vector<double>
        Pw) {
243         std::string name = ("Pw_versus_time");
```

```
244
245         std::ofstream outdata; //save data
246         outdata.open((name + ".dat").c_str());
247         outdata << "# time Temperature " << std::endl;
248         for (int i = 0; i < Pw.size(); i++)
249                 outdata << time[i] << " " << Pw[i] << std::endl;
250
251         CGnuplot::semilogx((name + ".dat").c_str(), "time", "Pw", (
              name + ".png").c_str());
252 }
253
254 bool CSimulador::isErrorNotAcceptable(double dt, std::vector<double
    > R, CGrid* grid, CProps* props_nu, double q, CDiscretization*
    discretization) {
255         double sum_R_MB = R[0], sum_Divisor_MB = 0.0, NR = 0.0;
256         for (int i = 0; i < R.size()-1; i++) {
257                 sum_R_MB += R[i+1];
258                 sum_Divisor_MB += (grid->get_Vb_ac(i) * props_nu->
                    get_phi(i));
259                 NR += R[i + 1] / (grid->get_Vb_ac(i) * props_nu->
                    get_phi(i));
260         }
261         double MB = dt * abs(sum_R_MB) / sum_Divisor_MB;
262
263         double biggestNR = abs(R[0] / q) > NR ? abs(R[0] / q) : NR;
264         return (MB > discretization->get_eps_MB() || biggestNR >
              discretization->get_eps_NR());
265 }
266
267 void CSimulador::read_data_and_start_objects(std::string nameFile)
    {
268         std::ifstream file(nameFile);
269
270         std::string text;
271         std::getline(file, text);
272         std::getline(file, text);
273         std::getline(file, text);
274         std::getline(file, text);
275         std::getline(file, text);
276
277         /// Pegando as variaveis do Poco
278         std::getline(file, text);
```

```
279
280         std::vector<double> tp;
281         std::vector<double> qsc;
282         std::vector<double> dz;
283         std::vector<double> partial;
284
285         int periodos;
286         file >> text; file >> text;
287         periodos = std::stoi(text);
288
289         /// pego os valores dos tps
290         file >> text;
291         for (int i = 0; i < periodos; i++) {
292                 file >> text;
293                 tp.push_back(std::stof(text));
294         }
295         std::getline(file, text);
296
297         /// pego os valores dos qsc
298         file >> text;
299         for (int i = 0; i < periodos; i++) {
300                 file >> text;
301                 qsc.push_back(std::stof(text));
302         }
303         std::getline(file, text);
304
305         /// pego os valores do h e partial
306         file >> text; file >> text;
307         periodos = std::stoi(text);
308         std::getline(file, text);
309         std::getline(file, text);
310
311         for (int i = 0; i < periodos; i++) {
312                 file >> text;
313                 dz.push_back(std::stof(text));
314                 file >> text; file >> text;
315                 partial.push_back(std::stof(text));
316         }
317
318         file >> text; file >> text;
319         double rw = std::stof(text);
320         well = new CWell(tp, qsc, dz, partial, rw);
```

```
321
322         ///
323         /// RESERVOIR
324         ///
325         std::getline(file, text);       std::getline(file, text);
               std::getline(file, text);
326         file >> text; file >> text;
327         bool isLiquid = std::stoi(text);
328
329         file >> text; file >> text;
330         double re = std::stof(text); std::getline(file, text);
331
332         file >> text; file >> text;
333         double theta = std::stof(text); std::getline(file, text);
334
335         file >> text; file >> text;
336         double k0r = std::stof(text); std::getline(file, text);
337
338         file >> text; file >> text;
339         double k0z = std::stof(text); std::getline(file, text);
340
341         file >> text; file >> text;
342         double cphi = std::stof(text); std::getline(file, text);
343
344         file >> text; file >> text;
345         double phi0 = std::stof(text); std::getline(file, text);
346
347         file >> text; file >> text;
348         double p0 = std::stof(text); std::getline(file, text);
349
350         file >> text; file >> text;
351         double p_i = std::stof(text); std::getline(file, text);
352
353         file >> text; file >> text;
354         double S = std::stof(text); std::getline(file, text);
355
356         file >> text; file >> text;
357         double Temperature = std::stof(text); std::getline(file,
               text);
358
359         reservoir = new CReservoir(isLiquid, rw, re, dz, theta, k0r
               , k0z, cphi, phi0, p0, p_i, S, Temperature);
```

```
360
361         ///
362         /// DISCRETIZACAO
363         ///
364         std::getline(file, text); std::getline(file, text);
365         int nz = dz.size();
366
367         file >> text; file >> text;
368         int nr = std::stoi(text); std::getline(file, text);
369
370         file >> text; file >> text;
371         int nrs = std::stoi(text); std::getline(file, text);
372
373         file >> text; file >> text;
374         int nt = std::stoi(text); std::getline(file, text);
375
376         file >> text; file >> text;
377         int ntp = std::stoi(text); std::getline(file, text);
378
379         file >> text; file >> text;
380         int max_iter = std::stoi(text); std::getline(file, text);
381
382         file >> text; file >> text;
383         double dtmin = std::stof(text); std::getline(file, text);
384
385         file >> text; file >> text;
386         double eps_NR = std::stof(text); std::getline(file, text);
387
388         file >> text; file >> text;
389         double eps_MB = std::stof(text); std::getline(file, text);
390
391         file >> text; file >> text;
392         double Ac = std::stof(text); std::getline(file, text);
393
394         file >> text; file >> text;
395         double Bc = std::stof(text); std::getline(file, text);
396
397         discretization = new CDiscretization(nz, nr, nrs, nt, ntp,
              max_iter, dtmin, eps_NR, eps_MB, Ac, Bc);
398
399         ///
400         /// FLUID
```

```
401         ///
402         if (reservoir->isLiquid())
403                 fluido = new CLiquido;
404         else
405                 fluido = new CGas;
406 
407         grid = new CGrid(reservoir, discretization, well);
408 }
```

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

Listing 6.6: Arquivo de cabeçalho da classe CReservoir.

```
1 #ifndef CRESERVOIR_HPP
2 #define CRESERVOIR_HPP
3 
4 #include<string>
5 #include<vector>
6 
7 class CReservoir {
8 private:
9         bool _isLiquid{1};
10 
11         double rw{ 0.09486 };           /// raio interno (poco)
12         double re{ 3000.0 };                    /// raio externo
13         std::vector<double> dz{ 1, 1 };                 /// altura
              do reservatorio
14         double theta{3.141592/6};       /// angulo estudado do
              reservatorio
15         double k0r{500};                                ////
              permeabilidade horizontal
16         double k0z{ 100 };                      /// permeabilidade
              vertical
17         double cphi{1.0e-4};                    ///
              compressibilidadae da formacao
18         double phi0{ 0.2 };                     /// porosidade
              inicial
19         double p0{1.033512};                    /// pressao de
              referencia
20         double p_i{350.0};                      /// pressoa inicial
21         double S{ 0 };                          /// fator de
              pelicula
22         double Temperature{353.15};     /// temperatura do
              reservatorio
```

```
23
24          bool aquifer{false};              /// presenca de aquifero
               por fronteira de Neumann?
25          bool infVol{false};                       /// presenca de
               aquifero por volume infinito?
26
27public:
28          CReservoir() {}
29          CReservoir( bool _isLiquid, double _rw, double _re, std::
               vector<double> _dz, double _theta, double _k0r, double
               _k0z, double _cphi, double _phi0, double _p0, double
               _p_i, double _S, double _Temperature) : _isLiquid{
               _isLiquid },
30                    rw{ _rw }, re{ _re }, dz{ _dz }, theta{ _theta },
                         k0r{ _k0r }, k0z{ _k0z }, cphi{ _cphi }, phi0{
                         _phi0 }, p0{ _p0 }, p_i{ _p_i }, S{ _S },
                         Temperature{ _Temperature } {}
31
32          double calc_phi(double p);
33
34          double calc_dphidp(double p);
35
36          /// funcoes get
37          bool isLiquid() { return _isLiquid; }
38          double get_rw() { return rw; }
39          double get_re() { return re; }
40          //double get_h() { return h; }
41          std::vector<double> get_dz() { return dz; }
42          double get_theta() { return theta; }
43          double get_k0r() { return k0r; }
44          double get_k0z() { return k0z; }
45          double get_cphi() { return cphi; }
46          double get_phi0() { return phi0; }
47          double get_p0() { return p0; }
48          double get_p_i() { return p_i; }
49          double get_S(){ return S; }
50          double get_Temperature() { return Temperature; }
51          bool get_aquifer() { return aquifer; }
52          bool get_infVol() { return infVol; }
53
54          void set_rw(double _rw) { rw = _rw; }
55          void set_re(double _re) { re = _re; }
```

```
56          //void set_h(double _h) { h = _h; }
57
58 };
59 #endif
```

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

Listing 6.7: Arquivo de implementação da classe CReservoir.

```
1 #include "CReservoir.hpp"
2
3 double CReservoir::calc_phi(double p) {
4          return phi0 * (1.0 + cphi * (p - p0));
5 }
6
7 double CReservoir::calc_dphidp(double p) {
8          return (phi0 * cphi);
9 }
```

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

Listing 6.8: Arquivo de cabeçalho da classe CProps.

```
1 #ifndef CPROPS_HPP
2 #define CPROPS_HPP
3
4 #include <vector>
5 #include "CGrid.hpp"
6 #include "CFluido.hpp"
7 #include "CReservoir.hpp"
8 #include "CDiscretization.hpp"
9
10 class CProps {
11 public:
12          CProps(CGrid* _grid, CFluido* _fluido, CReservoir*
               _reservoir, CDiscretization* _discretization);
13
14 private:
15          CGrid* grid;
16          CFluido* fluido;
17          CReservoir* reservoir;
18          CDiscretization* discretization;
19
20          int nr;
21          int nz;
```

```
22
23          std::vector<double> b;
24          std::vector<double> dbdp;
25
26          std::vector<double> biph;
27          std::vector<double> bimh;
28          std::vector<double> bjph;
29          std::vector<double> bjmh;
30
31
32          std::vector<double> mu;
33          std::vector<double> dmudp;
34
35          std::vector<double> muiph;
36          std::vector<double> muimh;
37          std::vector<double> mujph;
38          std::vector<double> mujmh;
39
40
41          std::vector<double> phi;
42          std::vector<double> dphidp;
43
44          double bw = .0;
45          double dbwdp1 = .0;
46          double dbwdpw = .0;
47          double muw = .0;
48          double dmuwdp1 = .0;
49          double dmuwdpw = .0;
50
51          void update_b(std::vector<double> pressure);
52          void update_mu(std::vector<double> pressure);
53          void update_well(double pw);
54
55  public:
56          void update(double pw, std::vector<double> pressure);
57
58          /// funcoes get do vetor completo
59          std::vector<double> get_phi()           { return phi; }
60          std::vector<double> get_dphidp()        { return dphidp; }
61          std::vector<double> get_b()                     { return b;
                }
62          std::vector<double> get_dbdp()          { return dbdp; }
```

```
63
64          std::vector<double> get_biph()          { return biph; }
65          std::vector<double> get_bimh()          { return bimh; }
66          std::vector<double> get_bjph()          { return bjph; }
67          std::vector<double> get_bjmh()          { return bjmh; }
68
69
70          std::vector<double> get_mu()            { return mu; }
71          std::vector<double> get_dmudp()         { return dmudp; }
72
73          std::vector<double> get_muiph()         { return mujph; }
74          std::vector<double> get_muimh()         { return muimh; }
75          std::vector<double> get_mujph()         { return mujph; }
76          std::vector<double> get_mujmh()         { return mujmh; }
77
78          /// get com as posicoes desejadas
79          double get_phi(int i)           { return phi[i]; }
80          double get_dphidp(int i)        { return dphidp[i]; }
81
82
83          double get_b(int i)                     { return b[i]; }
84          double get_dbdp(int i)          { return dbdp[i]; }
85
86          double get_biph(int i)          { return biph[i]; }
87          double get_bimh(int i)          { return bimh[i]; }
88          double get_bjph(int i)          { return bjph[i]; }
89          double get_bjmh(int i)          { return bjmh[i]; }
90
91
92          double get_mu(int i)            { return mu[i]; }
93          double get_dmudp(int i)         { return dmudp[i]; }
94
95          double get_muiph(int i)         { return mujph[i]; }
96          double get_muimh(int i)         { return muimh[i]; }
97          double get_mujph(int i)         { return mujph[i]; }
98          double get_mujmh(int i)         { return mujmh[i]; }
99
100         /// props do poco
101         double get_bw()                                 { return bw
               ; }
102         double get_dbwdp1()                             { return
               dbwdp1; }
```

```
103         double get_dbwdpw()                                    { return
               dbwdpw; }
104         double get_muw()                                       { return
               muw; }
105         double get_dmuwdp1()                          { return dmuwdp1; }
106         double get_dmuwdpw()                          { return dmuwdpw; }
107 };
108 #endif
```

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

Listing 6.9: Arquivo de implementação da classe CProps.

```
1  #include "CProps.hpp"
2
3  CProps::CProps(CGrid* _grid, CFluido* _fluido, CReservoir*
       _reservoir, CDiscretization* _discretization) {
4          grid = _grid ;
5          fluido = _fluido;
6          reservoir = _reservoir;
7          discretization = _discretization;
8
9          nr = discretization->get_nr();
10         nz = discretization->get_nz();
11
12         b.resize(nr*nz);
13         dbdp.resize(nr * nz);
14
15         biph.resize(nr * nz);
16         bimh.resize(nr * nz);
17         bjph.resize(nr * nz);
18         bjmh.resize(nr * nz);
19
20         mu.resize(nr * nz);
21         dmudp.resize(nr * nz);
22
23         muiph.resize(nr * nz);
24         muimh.resize(nr * nz);
25         mujph.resize(nr * nz);
26         mujmh.resize(nr * nz);
27
28         phi.resize(nr * nz);
29         dphidp.resize(nr * nz);
```

```
30 }
31
32 void CProps::update(double pw, std::vector<double> pressure) {
33         for (int i = 0; i < nr*nz; i++) {
34                 b[i] = fluido->calc_b(pressure[i]);
35                 dbdp[i] = fluido->calc_dbdp(pressure[i]);
36
37                 mu[i] = fluido->calc_mu(pressure[i]);
38                 dmudp[i] = fluido->calc_dmudp(pressure[i]);
39
40                 phi[i] = reservoir->calc_phi(pressure[i]);
41                 dphidp[i] = reservoir->calc_dphidp(pressure[i]);
42         }
43
44         update_b(pressure);
45         update_mu(pressure);
46         update_well(pw);
47 }
48
49 void CProps::update_well(double pw) {
50         bw = fluido->calc_b(pw);
51         dbwdp1 = 0;
52         dbwdpw = fluido->calc_dbdp(pw);
53         muw = fluido->calc_mu(pw);
54         dmuwdp1 = 0;
55         dmuwdpw = fluido->calc_dmudp(pw);
56 }
57
58 void CProps::update_b(std::vector<double> pressure) {
59         double omega = grid->get_omega();
60         std::vector<double> z = grid->get_z();
61         int camada = -1;
62
63         for (int i = 0; i < nr*nz; i++) {
64                 /// guardo a camada para uso futuro
65                 if (i % nr == 0) camada++;
66
67                 /// fronteira externa radial
68                 if ((i + 1) % nr == 0)
69                         biph[i] = b[i];
70                 else
71                         biph[i] = (1.0 - omega) * b[i] + omega * b[
```

```
                        i + 1];
72
73                  /// fronteira interna radial
74                  if (i % nr == 0)
75                          bimh[i] = b[i];
76                  else
77                          bimh[i] = biph[i - 1];
78
79                  /// fronteira inferior vertical
80                  if (i >= (nz-1)*nr)
81                          bjph[i] = b[i];
82                  else
83                          bjph[i] = (b[i] * z[camada] + b[i+nr] * z[
                                camada+1]) / (z[camada] + z[camada+1]);
84                  /// fronteira superior vertical
85                  if (i < nr)
86                          bjmh[i] = b[i];
87                  else
88                          bjmh[i] = bjph[i-nr];
89          }
90}
91
92void CProps::update_mu(std::vector<double> pressure) {
93          double omega = grid->get_omega();
94          std::vector<double> z = grid->get_z();
95          int camada = -1;
96
97          for (int i = 0; i < nr * nz; i++) {
98                  /// guardo a camada para uso futuro
99                  if (i % nr == 0) camada++;
100
101                 /// fronteira externa radial
102                 if ((i + 1) % nr == 0)
103                         muiph[i] = mu[i];
104                 else
105                         muiph[i] = (1.0 - omega) * mu[i] + omega *
                                mu[i + 1];
106
107                 /// fronteira interna radial
108                 if (i % nr == 0)
109                         muimh[i] = mu[i];
110                 else
```

```
111                         muimh[i] = muiph[i - 1];
112
113                 /// fronteira inferior vertical
114                 if (i >= (nz - 1) * nr)
115                         mujph[i] = mu[i];
116                 else
117                         mujph[i] = (mu[i] * z[camada] + mu[i + nr]
                               * z[camada + 1]) / (z[camada] + z[camada
                                + 1]);
118                 /// fronteira superior vertical
119                 if (i < nr)
120                         mujmh[i] = mu[i];
121                 else
122                         mujmh[i] = mujph[i - nr];
123         }
124}
```

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

Listing 6.10: Arquivo de cabeçalho da classe CLiquido.

```
1#ifndef CLIQUIDO_HPP
2#define CLIQUIDO_HPP
3
4#include <string>
5#include "CFluido.hpp"
6
7class CLiquido : public CFluido{
8public:
9         CLiquido() {}
10        CLiquido (float _cf, float _b0, float _p0, float _mu, float
              _cmu) :cf{ _cf }, b0{ _b0 }, p0{ _p0 }, mu{ _mu }, cmu{
              _cmu } {}
11
12        double calc_b(double p);
13        double calc_dbdp(double p);
14
15        double calc_mu(double p);
16        double calc_dmudp(double p);
17
18        std::string get_type() { return type; }
19
20private:
21        std::string type = "liquid";
```

```
22
23          double cf{ 14.7e-5 };            /// compressibilidade do
               fluido [cm^2/kgf]
24          double b0{ 1.0 };                        /// inverso do
               fator volume formacao na pressao p0 [m^3 std / m^3]
25          double p0{ 1.0335123 };          /// pressao de referencia [
               kgf/cm^2]
26          double mu{ 1.0 };                        /// viscosidade [cp
               ]
27          double cmu{ 0.0 };
28};
29#endif
```

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

Listing 6.11: Arquivo de implementação da classe CLiquido.

```
1#include "CLiquido.hpp"
2
3double CLiquido::calc_b(double p) {     return b0 * (1.0 + cf * (p
    - p0)); }
4
5double CLiquido::calc_dbdp(double p) { return b0 * cf; }
6
7double CLiquido::calc_mu(double p) { return mu * (1.0 + cmu * (p -
    p0)); }
8
9double CLiquido::calc_dmudp(double p) { return mu * cmu; }
```

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

Listing 6.12: Arquivo de cabeçalho da classe CGrid.

```
1#ifndef CGRID_HPP
2#define CGRID_HPP
3
4#include  <vector>
5#include <math.h>
6
7#include "CReservoir.hpp"
8#include "CDiscretization.hpp"
9#include "CWell.hpp"
10
11class CGrid {
12public:
```

```
13          CGrid(CReservoir* reservoir, CDiscretization*
               discretization, CWell* well);
14
15  private:
16          int nr, nz;
17          double alpha;
18          double omega;
19          double dtheta;
20
21          std::vector<double> time;        /// vetor dos tempos
22
23          std::vector<double> r;           /// posicao do centro do
               bloco em relacao a x
24          std::vector<double> rph;         /// posicao em x + 1/2
25          std::vector<double> rmh;         /// posicao em x - 1/2
26
27          std::vector<double> z;           /// posicao do centro do
               bloco em relacao a z
28          std::vector<double> zph;         /// posicao em z + 1/2
29          std::vector<double> zmh;         /// posicao em z - 1/2
30
31          std::vector<double> Vb;          /// volume
32          std::vector<double> Vb_ac;       /// volume corrigido
33
34          std::vector<double> kr;          /// permeabilidade
               horizontal
35          std::vector<double> kz;          /// permeabilidade vertical
36
37          std::vector<double> kiph;        /// permeabilidade em i +
               1/2
38          std::vector<double> kimh;        /// permeabilidade em i -
               1/2
39          std::vector<double> kjph;        /// permeabilidade em j +
               1/2
40          std::vector<double> kjmh;        /// permeabilidade em j -
               1/2
41
42          std::vector<double> Giph;        /// fator geometrico da
               transmissibilidade em i + 1/2
43          std::vector<double> Gimh;        /// fator geometrico da
               transmissibilidade em i - 1/2
44          std::vector<double> Gjph;        /// fator geometrico da
```

```
            transmissibilidade em j + 1/2
45          std::vector<double> Gjmh;       /// fator geometrico da
            transmissibilidade em j - 1/2
46
47          std::vector<double> Gw;         /// fator geometrico para
            calculo da c.c. interna (poco)
48
49  private: /// private functions
50          void createPosicoes(CReservoir* reservoir);
51          void createVolumes(CReservoir* reservoir, double Ac);
52          void createPermeabilidades(CReservoir* reservoir,
            CDiscretization* discretization);
53          void createFatorGeometricos(CReservoir* reservoir,
            CDiscretization* discretization, CWell* well);
54          void createTime(CDiscretization* discretization, CWell*
            well);
55
56  public:
57          /// funcoes get
58          double get_time( int i) { return time[i]; }
59
60          double get_r( int i)    { return r[i]; }
61          double get_rph( int i)  { return rph[i]; }
62          double get_rmh( int i)  { return rmh[i]; }
63
64          double get_z( int i)    { return z[i]; }
65          double get_zph( int i)  { return zph[i]; }
66          double get_zmh( int i)  { return zmh[i]; }
67
68          double get_Vb( int i)   { return Vb[i]; }
69          double get_Vb_ac( int i){ return Vb_ac[i]; }
70
71          double get_kr( int i)   { return kr[i]; }
72          double get_kz( int i)   { return kz[i]; }
73
74          double get_kiph( int i) { return kiph[i]; }
75          double get_kimh( int i) { return kimh[i]; }
76          double get_kjph( int i) { return kjph[i]; }
77          double get_kjmh( int i) { return kjmh[i]; }
78
79          double get_Giph( int i) { return Giph[i]; }
80          double get_Gimh( int i) { return Gimh[i]; }
```

```
81          double get_Gjph( int i) { return Gjph[i]; }
82          double get_Gjmh( int i) { return Gjmh[i]; }
83
84          double get_Gw( int i)   { return Gw[i]; }
85
86          /// get vetor completo
87          std::vector<double> get_time() { return time; }
88
89          std::vector<double> get_r() { return r; }
90          std::vector<double> get_rph() { return rph; }
91          std::vector<double> get_rmh() { return rmh; }
92
93          std::vector<double> get_z() { return z; }
94          std::vector<double> get_zph() { return zph; }
95          std::vector<double> get_zmh() { return zmh; }
96
97          std::vector<double> get_Vb() { return Vb; }
98          std::vector<double> get_Vb_ac() { return Vb_ac; }
99
100         std::vector<double> get_kr() { return kr; }
101         std::vector<double> get_kz() { return kz; }
102
103         std::vector<double> get_kiph() { return kiph; }
104         std::vector<double> get_kimh() { return kimh; }
105         std::vector<double> get_kjph() { return kjph; }
106         std::vector<double> get_kjmh() { return kjmh; }
107
108         std::vector<double> get_Giph() { return Giph; }
109         std::vector<double> get_Gimh() { return Gimh; }
110         std::vector<double> get_Gjph() { return Gjph; }
111         std::vector<double> get_Gjmh() { return Gjmh; }
112
113         std::vector<double> get_Gw() { return Gw; }
114
115
116         int get_ntotal()                                { return nr
                * nz; }
117         int get_nr()                                    { return nr
               ; }
118         int get_nz()                                    { return nz
               ; }
119
```

```
120         double get_alpha()                                  { return
               alpha; }
121         double get_omega()                                  { return
               omega; }
122         double get_dtheta()                                 { return
               dtheta; }
123 };
124 #endif
```

Apresenta-se na listagem 6.15 o arquivo com o código da implementação da classe CGrid.

Listing 6.13: Arquivo de implementação da classe CGrid.

```
1 #include "CGrid.hpp"
2
3 CGrid::CGrid(CReservoir* reservoir, CDiscretization* discretization
      , CWell* well) {
4         nr = discretization->get_nr();
5         nz = discretization->get_nz();
6         dtheta = reservoir->get_theta();
7         createPosicoes(reservoir);
8         createVolumes(reservoir, discretization->get_Ac());
9         createPermeabilidades(reservoir, discretization);
10        createFatorGeometricos(reservoir, discretization, well);
11        createTime(discretization, well);
12 }
13
14 void CGrid::createPosicoes(CReservoir* reservoir) {
15        alpha = pow(reservoir->get_re() / reservoir->get_rw(), 1.0
             / nr);
16        omega = log((alpha - 1) / log(alpha)) / log(alpha);
17
18        /// -------------- raio --------------
19        r.push_back(reservoir->get_rw() * log(alpha) / (1-(1/alpha)
             ));
20        rmh.push_back(reservoir->get_rw());
21        for (int i = 1; i < nr; i++) {
22                r.push_back(r[0] * pow(alpha, i));
23                rph.push_back((r[i] - r[i-1]) / log(alpha));
24                rmh.push_back(rph[i-1]);
25        }
26        rph.push_back(reservoir->get_re());
27
```

```
28          /// ---------- profundidade ----------
29          std::vector<double> dz = reservoir->get_dz();
30          for (int j = 0; j < nz; j++) {
31                  zmh.push_back(0.0+(j==0?0:dz[j]+zmh[j-1]));
32                  zph.push_back(zmh[j]+dz[j]);
33                  z.push_back((zmh[j] + zph[j]) / 2.0);
34          }
35}
36
37void CGrid::createVolumes(CReservoir* reservoir, double Ac) {
38          double vb;
39          std::vector<double> dz = reservoir->get_dz();
40          for (int k = 0; k < nz; k++) {
41                  for (int i = 0; i < nr; i++) {
42                          vb = 0.5 * (pow(rph[i], 2) - pow(rmh[i], 2)
                               ) * reservoir->get_theta() * dz[k];
43                          Vb.push_back( vb );
44                          Vb_ac.push_back(vb * Ac);
45                  }
46          }
47}
48
49void CGrid::createPermeabilidades(CReservoir* reservoir,
    CDiscretization* discretization) {
50          int nrs = discretization->get_nrs();
51
52          double k0r = reservoir->get_k0r();
53          double k0z = reservoir->get_k0z();
54
55          double krs = k0r / (reservoir->get_S() / log(rph[nrs] /
               reservoir->get_rw()) + 1.0);
56          double kzs = k0z / (reservoir->get_S() / log(rph[nrs] /
               reservoir->get_rw()) + 1.0);
57
58          /// nos centros dos volumes
59          for (int k = 0; k < nz; k++) {
60                  for (int i = 0; i < nr; i++) {
61                          /// regiao danificada
62                          if (i <= nrs) {
63                                  kr.push_back(k0r + krs);
64                                  kz.push_back(k0z + kzs);
65                          }
```

```
66                          /// regiao normal
67                          else {
68                                  kr.push_back(k0r);
69                                  kz.push_back(k0z);
70                          }
71                          /// volume infinito
72                          if (reservoir->get_infVol() && k == nz-1) {
73                                  kr[i + k * nr] = kr[i + k * nr] *
                                       1.0e4; /// multiplicador
                                       arbitrario, valor deve ser
                                       grande
74                                  kz[i + k * nr] = kz[i + k * nr] *
                                       1.0e4; /// multiplicador
                                       arbitrario, valor deve ser
                                       grande
75                          }
76
77                          /// permeabilidades homogeneas entre as
                               camadas
78                          kiph.push_back(k0r);
79                          kimh.push_back(k0r);
80
81                          kjph.push_back(k0z);
82                          kjmh.push_back(k0z);
83                  }
84          }
85}
86
87void CGrid::createFatorGeometricos(CReservoir* reservoir,
    CDiscretization* discretization, CWell* well) {
88          double fatorGeometrico;
89          std::vector<double> partial = well->get_partial();
90          for (int k = 0; k < nz; k++) {
91                  for (int i = 0; i < nr; i++) {
92                  /// fator geometrico em i - 1/2
93                  if (i == 0)
94                          Gimh.push_back(0.0);
95                  else
96                          Gimh.push_back(discretization->get_Bc() *
                               rmh[i] * kimh[i + k * nr] * reservoir->
                               get_theta() * (zph[k] - zmh[k]) / (r[i]
                               - r[i - 1]));
```

```
97
98                  /// fator geometrico em i - 1/2
99                  if (i == nr-1)
100                         Giph.push_back(0.0);
101                 else
102                         Giph.push_back(discretization->get_Bc() *
                               rph[i] * kiph[i + k * nr] * reservoir->
                               get_theta() * (zph[k] - zmh[k]) / (r[i
                               +1] - r[i]));
103
104                 /// fator geometrico em j - 1/2
105                 if (k == 0)
106                         Gjmh.push_back(0.0);
107                 else
108                         Gjmh.push_back(discretization->get_Bc() * (
                               pow(rph[i], 2) - pow(rmh[i], 2)) * kjmh[
                               i + k * nr] * reservoir->get_theta() /
                               (2 * (z[k] - z[k - 1])));
109
110                 /// fator geometrico em j + 1/2
111                 if (k == nz-1)
112                         Gjph.push_back(0.0);
113                 else
114                         Gjph.push_back(discretization->get_Bc() * (
                               pow(rph[i], 2) - pow(rmh[i], 2)) * kjph[
                               i + k * nr] * reservoir->get_theta() /
                               (2 * (z[k+1] - z[k])));
115
116
117                 /// fator geometrico do poco
118                 if (i == 0)
119                         Gw.push_back(partial[k] * discretization->
                               get_Bc() * rmh[0] * kimh[k * nr] *
                               reservoir->get_theta() * (zph[k] - zmh[k
                               ]) / (r[0] - rmh[0]));
120                 }
121         }
122 }
123
124 void CGrid::createTime(CDiscretization* discretization, CWell* well
    ) {
125         std::vector<double> tp = well->get_tp();
```

```
126         double dtmin = discretization->get_dtmin();
127         time.push_back(tp[0]);
128         for (int i = 1; i < tp.size(); i++) {
129                 for (int t = 0; t < discretization->get_ntp(); t++)
                        {
130                         time.push_back(tp[i-1] + pow(10, t * (log10
                            (tp[i] - tp[i - 1]) - log10(dtmin)) / (
                            discretization->get_ntp()-1.0))*dtmin);
131                 }
132         }
133 }
```

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

Listing 6.14: Arquivo de cabeçalho da classe CGrid.

```
1 #ifndef CGRID_HPP
2 #define CGRID_HPP
3
4 #include  <vector>
5 #include <math.h>
6
7 #include "CReservoir.hpp"
8 #include "CDiscretization.hpp"
9 #include "CWell.hpp"
10
11 class CGrid {
12 public:
13         CGrid(CReservoir* reservoir, CDiscretization*
             discretization, CWell* well);
14
15 private:
16         int nr, nz;
17         double alpha;
18         double omega;
19         double dtheta;
20
21         std::vector<double> time;        /// vetor dos tempos
22
23         std::vector<double> r;           /// posicao do centro do
             bloco em relacao a x
24         std::vector<double> rph;         /// posicao em x + 1/2
25         std::vector<double> rmh;         /// posicao em x - 1/2
26
```

```
27          std::vector<double> z;          /// posicao do centro do
               bloco em relacao a z
28          std::vector<double> zph;        /// posicao em z + 1/2
29          std::vector<double> zmh;        /// posicao em z - 1/2
30
31          std::vector<double> Vb;         /// volume
32          std::vector<double> Vb_ac;      /// volume corrigido
33
34          std::vector<double> kr;         /// permeabilidade
               horizontal
35          std::vector<double> kz;         /// permeabilidade vertical
36
37          std::vector<double> kiph;       /// permeabilidade em i +
               1/2
38          std::vector<double> kimh;       /// permeabilidade em i -
               1/2
39          std::vector<double> kjph;       /// permeabilidade em j +
               1/2
40          std::vector<double> kjmh;       /// permeabilidade em j -
               1/2
41
42          std::vector<double> Giph;       /// fator geometrico da
               transmissibilidade em i + 1/2
43          std::vector<double> Gimh;       /// fator geometrico da
               transmissibilidade em i - 1/2
44          std::vector<double> Gjph;       /// fator geometrico da
               transmissibilidade em j + 1/2
45          std::vector<double> Gjmh;       /// fator geometrico da
               transmissibilidade em j - 1/2
46
47          std::vector<double> Gw;         /// fator geometrico para
               calculo da c.c. interna (poco)
48
49  private: /// private functions
50          void createPosicoes(CReservoir* reservoir);
51          void createVolumes(CReservoir* reservoir, double Ac);
52          void createPermeabilidades(CReservoir* reservoir,
               CDiscretization* discretization);
53          void createFatorGeometricos(CReservoir* reservoir,
               CDiscretization* discretization, CWell* well);
54          void createTime(CDiscretization* discretization, CWell*
               well);
```

```
55
56 public:
57         /// funcoes get
58         double get_time( int i) { return time[i]; }
59
60         double get_r( int i)    { return r[i]; }
61         double get_rph( int i)  { return rph[i]; }
62         double get_rmh( int i)  { return rmh[i]; }
63
64         double get_z( int i)    { return z[i]; }
65         double get_zph( int i)  { return zph[i]; }
66         double get_zmh( int i)  { return zmh[i]; }
67
68         double get_Vb( int i)   { return Vb[i]; }
69         double get_Vb_ac( int i){ return Vb_ac[i]; }
70
71         double get_kr( int i)   { return kr[i]; }
72         double get_kz( int i)   { return kz[i]; }
73
74         double get_kiph( int i) { return kiph[i]; }
75         double get_kimh( int i) { return kimh[i]; }
76         double get_kjph( int i) { return kjph[i]; }
77         double get_kjmh( int i) { return kjmh[i]; }
78
79         double get_Giph( int i) { return Giph[i]; }
80         double get_Gimh( int i) { return Gimh[i]; }
81         double get_Gjph( int i) { return Gjph[i]; }
82         double get_Gjmh( int i) { return Gjmh[i]; }
83
84         double get_Gw( int i)   { return Gw[i]; }
85
86         /// get vetor completo
87         std::vector<double> get_time() { return time; }
88
89         std::vector<double> get_r() { return r; }
90         std::vector<double> get_rph() { return rph; }
91         std::vector<double> get_rmh() { return rmh; }
92
93         std::vector<double> get_z() { return z; }
94         std::vector<double> get_zph() { return zph; }
95         std::vector<double> get_zmh() { return zmh; }
96
```

```
97          std::vector<double> get_Vb() { return Vb; }
98          std::vector<double> get_Vb_ac() { return Vb_ac; }
99
100         std::vector<double> get_kr() { return kr; }
101         std::vector<double> get_kz() { return kz; }
102
103         std::vector<double> get_kiph() { return kiph; }
104         std::vector<double> get_kimh() { return kimh; }
105         std::vector<double> get_kjph() { return kjph; }
106         std::vector<double> get_kjmh() { return kjmh; }
107
108         std::vector<double> get_Giph() { return Giph; }
109         std::vector<double> get_Gimh() { return Gimh; }
110         std::vector<double> get_Gjph() { return Gjph; }
111         std::vector<double> get_Gjmh() { return Gjmh; }
112
113         std::vector<double> get_Gw() { return Gw; }
114
115
116         int get_ntotal()                                    { return nr
                * nz; }
117         int get_nr()                                        { return nr
               ; }
118         int get_nz()                                        { return nz
               ; }
119
120         double get_alpha()                                  { return
               alpha; }
121         double get_omega()                                  { return
               omega; }
122         double get_dtheta()                                 { return
               dtheta; }
123 };
124 #endif
```

Apresenta-se na listagem 6.15 o arquivo com o código da implementação da classe CGrid.

Listing 6.15: Arquivo de implementação da classe CGrid.

```
1 #include "CGrid.hpp"
2
3 CGrid::CGrid(CReservoir* reservoir, CDiscretization* discretization
      , CWell* well) {
```

```
4         nr = discretization->get_nr();
5         nz = discretization->get_nz();
6         dtheta = reservoir->get_theta();
7         createPosicoes(reservoir);
8         createVolumes(reservoir, discretization->get_Ac());
9         createPermeabilidades(reservoir, discretization);
10        createFatorGeometricos(reservoir, discretization, well);
11        createTime(discretization, well);
12}
13
14void CGrid::createPosicoes(CReservoir* reservoir) {
15        alpha = pow(reservoir->get_re() / reservoir->get_rw(), 1.0
              / nr);
16        omega = log((alpha - 1) / log(alpha)) / log(alpha);
17
18        /// -------------- raio --------------
19        r.push_back(reservoir->get_rw() * log(alpha) / (1-(1/alpha)
              ));
20        rmh.push_back(reservoir->get_rw());
21        for (int i = 1; i < nr; i++) {
22                r.push_back(r[0] * pow(alpha, i));
23                rph.push_back((r[i] - r[i-1]) / log(alpha));
24                rmh.push_back(rph[i-1]);
25        }
26        rph.push_back(reservoir->get_re());
27
28        /// ---------- profundidade ----------
29        std::vector<double> dz = reservoir->get_dz();
30        for (int j = 0; j < nz; j++) {
31                zmh.push_back(0.0+(j==0?0:dz[j]+zmh[j-1]));
32                zph.push_back(zmh[j]+dz[j]);
33                z.push_back((zmh[j] + zph[j]) / 2.0);
34        }
35}
36
37void CGrid::createVolumes(CReservoir* reservoir, double Ac) {
38        double vb;
39        std::vector<double> dz = reservoir->get_dz();
40        for (int k = 0; k < nz; k++) {
41                for (int i = 0; i < nr; i++) {
42                        vb = 0.5 * (pow(rph[i], 2) - pow(rmh[i], 2)
                              ) * reservoir->get_theta() * dz[k];
```

```
43                          Vb.push_back( vb );
44                          Vb_ac.push_back(vb * Ac);
45                  }
46          }
47}
48
49void CGrid::createPermeabilidades(CReservoir* reservoir,
    CDiscretization* discretization) {
50          int nrs = discretization->get_nrs();
51
52          double k0r = reservoir->get_k0r();
53          double k0z = reservoir->get_k0z();
54
55          double krs = k0r / (reservoir->get_S() / log(rph[nrs] /
               reservoir->get_rw()) + 1.0);
56          double kzs = k0z / (reservoir->get_S() / log(rph[nrs] /
               reservoir->get_rw()) + 1.0);
57
58          /// nos centros dos volumes
59          for (int k = 0; k < nz; k++) {
60                  for (int i = 0; i < nr; i++) {
61                          /// regiao danificada
62                          if (i <= nrs) {
63                                  kr.push_back(k0r + krs);
64                                  kz.push_back(k0z + kzs);
65                          }
66                          /// regiao normal
67                          else {
68                                  kr.push_back(k0r);
69                                  kz.push_back(k0z);
70                          }
71                          /// volume infinito
72                          if (reservoir->get_infVol() && k == nz-1) {
73                                  kr[i + k * nr] = kr[i + k * nr] *
                                       1.0e4; /// multiplicador
                                       arbitrario, valor deve ser
                                       grande
74                                  kz[i + k * nr] = kz[i + k * nr] *
                                       1.0e4; /// multiplicador
                                       arbitrario, valor deve ser
                                       grande
75                          }
```

```
76
77                          /// permeabilidades homogeneas entre as
                                camadas
78                          kiph.push_back(k0r);
79                          kimh.push_back(k0r);
80
81                          kjph.push_back(k0z);
82                          kjmh.push_back(k0z);
83                  }
84          }
85}
86
87void CGrid::createFatorGeometricos(CReservoir* reservoir,
    CDiscretization* discretization, CWell* well) {
88          double fatorGeometrico;
89          std::vector<double> partial = well->get_partial();
90          for (int k = 0; k < nz; k++) {
91                  for (int i = 0; i < nr; i++) {
92                  /// fator geometrico em i - 1/2
93                  if (i == 0)
94                          Gimh.push_back(0.0);
95                  else
96                          Gimh.push_back(discretization->get_Bc() *
                                rmh[i] * kimh[i + k * nr] * reservoir->
                                get_theta() * (zph[k] - zmh[k]) / (r[i]
                                - r[i - 1]));
97
98                  /// fator geometrico em i - 1/2
99                  if (i == nr-1)
100                         Giph.push_back(0.0);
101                 else
102                         Giph.push_back(discretization->get_Bc() *
                                rph[i] * kiph[i + k * nr] * reservoir->
                                get_theta() * (zph[k] - zmh[k]) / (r[i
                                +1] - r[i]));
103
104                 /// fator geometrico em j - 1/2
105                 if (k == 0)
106                         Gjmh.push_back(0.0);
107                 else
108                         Gjmh.push_back(discretization->get_Bc() * (
                                pow(rph[i], 2) - pow(rmh[i], 2)) * kjmh[
```

```
                        i + k * nr] * reservoir->get_theta() /
                        (2 * (z[k] - z[k - 1])));
109
110                 /// fator geometrico em j + 1/2
111                 if (k == nz-1)
112                         Gjph.push_back(0.0);
113                 else
114                         Gjph.push_back(discretization->get_Bc() * (
                            pow(rph[i], 2) - pow(rmh[i], 2)) * kjph[
                            i + k * nr] * reservoir->get_theta() /
                            (2 * (z[k+1] - z[k])));
115
116
117                 /// fator geometrico do poco
118                 if (i == 0)
119                         Gw.push_back(partial[k] * discretization->
                            get_Bc() * rmh[0] * kimh[k * nr] *
                            reservoir->get_theta() * (zph[k] - zmh[k
                            ]) / (r[0] - rmh[0]));
120                 }
121         }
122}
123
124void CGrid::createTime(CDiscretization* discretization, CWell* well
    ) {
125         std::vector<double> tp = well->get_tp();
126         double dtmin = discretization->get_dtmin();
127         time.push_back(tp[0]);
128         for (int i = 1; i < tp.size(); i++) {
129                 for (int t = 0; t < discretization->get_ntp(); t++)
                        {
130                         time.push_back(tp[i-1] + pow(10, t * (log10
                            (tp[i] - tp[i - 1]) - log10(dtmin)) / (
                            discretization->get_ntp()-1.0))*dtmin);
131                 }
132         }
133}
```

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

Listing 6.16: Arquivo de cabeçalho da classe CGnuplot.

```
1#ifndef CGNUPLOT_HPP
2#define CGNUPLOT_HPP
```

```
3
4#include <vector>
5#include <string>
6#include <iostream>
7#include <stdio.h>
8#include <stdlib.h>
9
10#ifdef _WIN32
11#define GNUPLOT_NAME "C:\\Program\" \"Files\\gnuplot\\bin\\gnuplot␣
    -p"
12#else
13#define GNUPLOT_NAME "gnuplot"
14#endif
15
16class CGnuplot {
17public:
18        CGnuplot() {}
19
20        static void plot(std::string name, std::string xlabel, std
            ::string ylabel, std::string saveName);
21        static void semilogx(std::string name, std::string xlabel,
            std::string ylabel, std::string saveName);
22        static void semilogy(std::string name, std::string xlabel,
            std::string ylabel, std::string saveName);
23};
24#endif
```

Apresenta-se na listagem 6.17 o arquivo com o código da implementação da classe CGnuplot.

Listing 6.17: Arquivo de implementação da classe CGnuplot.

```
1#include "CGnuplot.hpp"
2
3using namespace std;
4
5void CGnuplot::plot(string name, string xlabel, string ylabel,
    string saveName) {
6#ifdef _WIN32
7        FILE* pipe = _popen(GNUPLOT_NAME, "w");
8#else
9        FILE* pipe = popen(GNUPLOT_NAME, "w");
10#endif
11        fprintf(pipe, ("set␣xlabel␣'" + xlabel + "'\n").c_str());
```

```
12          fprintf(pipe, ("set␣ylabel␣'" + ylabel + "'\n").c_str());
13          fprintf(pipe, "unset␣key\n");
14          fprintf(pipe, ("plot␣'" + name + "'␣with␣linespoints␣
                linestyle␣1\n").c_str());
15          fprintf(pipe, "set␣term␣pngcairo\n");
16          fprintf(pipe, ("set␣output␣'" + saveName + "'\n").c_str());
17          fprintf(pipe, "replot\n");
18          fprintf(pipe, "set␣term␣win\n");
19          fflush(pipe);
20  }
21
22  void CGnuplot::semilogy(string name, string xlabel, string ylabel,
        string saveName) {
23  #ifdef _WIN32
24          FILE* pipe = _popen(GNUPLOT_NAME, "w");
25  #else
26          FILE* pipe = popen(GNUPLOT_NAME, "w");
27  #endif
28          fprintf(pipe, ("set␣xlabel␣'" + xlabel + "'\n").c_str());
29          fprintf(pipe, ("set␣ylabel␣'" + ylabel + "'\n").c_str());
30          fprintf(pipe, ("set␣logscale␣y\n"));
31          fprintf(pipe, "unset␣key\n");
32          fprintf(pipe, ("plot␣'" + name + "'␣with␣linespoints␣
                linestyle␣1\n").c_str());
33          fprintf(pipe, "set␣term␣pngcairo\n");
34          fprintf(pipe, ("set␣output␣'" + saveName + "'\n").c_str());
35          fprintf(pipe, "replot\n");
36          fprintf(pipe, "set␣term␣win\n");
37          fflush(pipe);
38  }
39
40  void CGnuplot::semilogx(string name, string xlabel, string ylabel,
        string saveName) {
41  #ifdef _WIN32
42          FILE* pipe = _popen(GNUPLOT_NAME, "w");
43  #else
44          FILE* pipe = popen(GNUPLOT_NAME, "w");
45  #endif
46          fprintf(pipe, ("set␣xlabel␣'" + xlabel + "'\n").c_str());
47          fprintf(pipe, ("set␣ylabel␣'" + ylabel + "'\n").c_str());
48          fprintf(pipe, ("set␣logscale␣x\n"));
49          fprintf(pipe, "unset␣key\n");
```

```
50          fprintf(pipe, ("plot␣'" + name + "'␣with␣linespoints␣
               linestyle␣1\n").c_str());
51          fprintf(pipe, "set␣term␣pngcairo\n");
52          fprintf(pipe, ("set␣output␣'" + saveName + "'\n").c_str());
53          fprintf(pipe, "replot\n");
54          fprintf(pipe, "set␣term␣win\n");
55          fflush(pipe);
56  }
```

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

Listing 6.18: Arquivo de cabeçalho da classe CGas.

```
1 #ifndef CGAS_HPP
2 #define CGAS_HPP
3
4 #include<math.h>
5 #include <string>
6 #include<iostream>
7 #include "CFluido.hpp"
8
9 #include <iostream>
10
11 class CGas : public CFluido{
12 public:
13          CGas() {}
14
15 public:
16          const double DELTA = 1.0e-5;
17          double calc_b(double p);
18          double calc_rho(double p);
19          double calc_dbdp(double p);
20          double calc_mu(double p);
21          double calc_dmudp(double p);
22
23          double Z_KIAM(double p);
24          double mu_LGE(double rho);
25
26          std::string get_type() { return type; }
27
28 private:
29          std::string type = "gas";
30
31          double cf{ 0.00215094 };          /// compressibilidade do
```

```
            fluido na condi??o inicial [cm^2/kgf]
32          double p0{ 1.0335123 };          /// pressao padrao [kgf/cm
               ^2]
33          double mu{ 0.0262317 };          /// viscosidade na condicao
                inicial [cp]
34          double T0{ 288.75 };             /// temperatura absoluta
               padrao [K]
35          double T{ 353.15 };                      /// temperatura do
               fluido no reservat?rio [K]
36          double Tpc{ 216.32 };            /// temperatura
               pseudocritica [K]
37          double Ppc{ 46.34 };             /// pressao pseudocritica [
               kgf/cm^2]
38          double Ma{ 20.3 };                       /// massa molecular
                aparente [kg/kg-mol]
39
40          const double R = 0.08478;        ///constante universal dos
               gases (ANP) [(kgf/cm^2)*(m^3)/(kg-mol*K)]
41  };
42  #endif
```

Apresenta-se na listagem 6.19 o arquivo com o código da implementação da classe CGas.

Listing 6.19: Arquivo de implementação da classe CGas.

```
1  #include "CGas.hpp"
2
3  double CGas::calc_b(double p) {
4          return (T0 * p) / (p0 * Z_KIAM(p) * T);
5  }
6
7  double CGas::calc_rho(double p) {
8          return (p * Ma) / (Z_KIAM(p) * R * T);
9  }
10
11 double CGas::calc_dbdp(double p) {
12         return T0 / (p0 * T * p * DELTA) * ((p * (1.0 + DELTA) /
               Z_KIAM(p * (1.0 + DELTA))) - (p / Z_KIAM(p)));
13 }
14
15 double CGas::calc_mu(double p) {
16         return mu_LGE(0.001 * calc_rho(p));
17 }
```

```
18
19 double CGas::calc_dmudp(double p) {
20
21         double pf = p * (1.0 + DELTA);
22         double Zf = Z_KIAM(pf);
23
24         double rhof = (pf * Ma) / (Zf * R * T);
25         double muf = mu_LGE(0.001 * rhof);
26         double mu = mu_LGE(0.001 * calc_rho(p));
27
28         return (muf - mu) / (p * DELTA);
29 }
30
31 double CGas::Z_KIAM(double p) {
32         double ppr = p / Ppc;
33         double tpr = T / Tpc;
34
35         double A1 = +0.3178420;
36         double A2 = +0.3822160;
37         double A3 = -7.7683540;
38         double A4 = +14.290531;
39         double A5 = +0.0000020;
40         double A6 = -0.0046930;
41         double A7 = +0.0962540;
42         double A8 = +0.1667200;
43         double A9 = +0.9669100;
44         double A10 = +0.0630690;
45         double A11 = -1.9668470;
46         double A12 = +21.058100;
47         double A13 = -27.024600;
48         double A14 = +16.230000;
49         double A15 = +207.78300;
50         double A16 = -488.16100;
51         double A17 = +176.29000;
52         double A18 = +1.8845300;
53         double A19 = +3.0592100;
54
55         double t = 1 / tpr;
56         double A = A1 * t * exp(A2 * pow(1.0 - t, 2)) * ppr;
57         double B = A3 * t + A4 * pow(t, 2) + (A5 * pow(t, 6)) * pow
               (ppr, 6);
58         double C = A9 + A8 * t * ppr + A7 * pow(t, 2) * pow(ppr, 2)
```

```
            + A6 * pow(t, 3) * pow(ppr, 3);
59          double D = A10 * t * exp(A11 * pow(1.0 - t, 2));
60          double E = A12 * t + A13 * pow(t, 2) + A14 * pow(t, 3);
61          double F = A15 * t + A16 * pow(t, 2) + A17 * pow(t, 3);
62          double G = A18 + A19 * t;
63          double y = (D * ppr) / ((1 + pow(A, 2)) / C - (pow(A, 2)*B)
                / pow(C, 3));
64
65          double Z = D * ppr*(1 + y + pow(y, 2) - pow(y, 3))
66                  / ((D * ppr + E * pow(y, 2) - F * pow(y, G)) * pow
                       (1 - y, 3));
67
68          return Z;
69  }
70
71  double CGas::mu_LGE(double rho) {
72          double K = ((9.379 + 0.01607 * Ma) * pow(1.8 * T, 1.5)) /
                (209.2 + 19.26 * Ma + 1.8 * T);
73          double X = 3.448 + (986.4 / (1.8 * T)) + 0.01009 * Ma;
74          double Y = 2.447 - 0.2224 * X;
75          return (1.0e-4) * K * exp(X * pow(rho, Y));
76  }
```

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

Listing 6.20: Arquivo de cabeçalho da classe CFluido.

```
1 #ifndef CFLUIDO_HPP
2 #define CFLUIDO_HPP
3
4 #include <string>
5
6 class CFluido {
7 public:
8         virtual double calc_b(double p) { return 0.0; }
9         virtual double calc_dbdp(double p) { return 0.0; }
10
11        virtual double calc_mu(double p) { return 0.0; }
12        virtual double calc_dmudp(double p) { return 0.0; }
13
14        virtual std::string get_type() { return type; }
15 private:
16        std::string type;
17
```

```
18 };
19
20 #endif
```

Apresenta-se na listagem 6.21 o arquivo com o código da implementação da classeC-Fluido.

Listing 6.21: Arquivo de implementação da classe CFluido.

```
1 #include "CFluido.hpp"
```

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

Listing 6.22: Arquivo de cabeçalho da classe CDiscretization.

```
1 #ifndef CDISCRETIZATION_HPP
2 #define CDISCRETIZATION_HPP
3
4 class CDiscretization {
5
6 private:
7         int nz{ 2 };                                    /// qtd de
            volumes na altura
8         int nr{ 50 };                                   /// qtd de
            volumes na largura
9         int nrs{ 1 };                                   /// qtd de
            volumes na regiao danificada
10        int nt{ 100 };                                  /// qtd de
            tempos
11        int ntp{ 100 };                                 /// qtd de
            tempos
12        int max_iter{ 24 };                             /// numero
            maximo de iteracoes
13        double dtmin{ 1.0 / 3600 };             /// passo de tempo
            minimo [h]
14        double eps_NR{ 1.0e-6 };                /// tolerancia de
            convergencia dos residuos
15        double eps_MB{ 1.0e-8 };                /// tolerancia de
            convergencia do balanco de materiais
16        double Ac{ 24 };                                ///
            constante de conversao de unidades acumulo (ANP)
17        double Bc{ 0.0083621472 };              /// constante de
            conversao de unidades fluxo (ANP)
18
19 public:
20        CDiscretization() {}
```

```
21          CDiscretization(int _nz, int _nr, int _nrs, int _nt, int
               _ntp, int _max_iter, double _dtmin, double _eps_NR,
               double _eps_MB, double _Ac, double _Bc):
22                   nz{ _nz }, nr{ _nr }, nrs{ _nrs }, nt{ _nt }, ntp{
                        _ntp }, max_iter{ _max_iter }, dtmin{ _dtmin },
                        eps_NR{ _eps_NR }, eps_MB{ _eps_MB }, Ac{ _Ac },
                         Bc{ _Bc }{}
23          /// funcoes get
24          int get_nz()             { return nz; }
25          int get_nr()             { return nr; }
26          int get_nrs()            { return nrs; }
27          int get_nt()             { return nt; }
28          int get_ntp()            { return ntp; }
29          int get_max_iter()       { return max_iter; }
30          double get_dtmin()       { return dtmin; }
31          double get_eps_NR() { return eps_NR; }
32          double get_eps_MB() { return eps_MB; }
33          double get_Ac()          { return Ac; }
34          double get_Bc()          { return Bc; }
35  };
36  #endif
```

Apresenta-se na listagem 6.23 o arquivo com o código da implementação da classe CDiscretization.

Listing 6.23: Arquivo de implementação da classe CDiscretization.

```
1 #include "CDiscretization.hpp"
```

Capítulo 7

Teste

Todo projeto de engenharia passa por uma etapa de testes. Neste capítulo apresentamos alguns testes do software desenvolvido. Para isso, é validado o software desenvolvido, e é utilizado o artigo [Bilhartz and Ramey, 1977], onde ele mostra o efeito da penetração parcial.

Algumas definições importantes definidas pelo artigo:

- Razão de penetração: $b = \frac{h_w}{h}$, é a razão entre a altura aberta à produção e a altura total do reservatório.
- Espessura do poço adimensional: $h_D = \frac{h_w}{r_w}\sqrt{\frac{k_r}{k_w}}$;
- Tempo adimensional: $T_D = t\frac{C_1 k_r}{\phi\mu(c_f+c_\phi)r_w^2}$;
- Pressão adimensional: $P_D = (p_i - p)\frac{bk_r h}{C_2 q\mu}$;

Transformando os resultados do simulador para o tempo e pressão adimensional apresentado acima, é possível comparar os valores com a tabela 1 do artigo.

7.1 Teste 1

No teste 1, buscou-se o cenário onde b=1/2 e $h_D = 250$. Para isso, foi utilizado as seguintes propriedades:

- $h = 10m$;
- $r_w = 0.0447m$;
- $h_w = 5m$;

Além disso, foi utilizado:

- $k_r = 500md$;
- $k_z = 100md$;

- $\phi = 0.2$;
- $q = 1000 m^3/dia$;
- $p_i = 350 kgf/cm^2$;

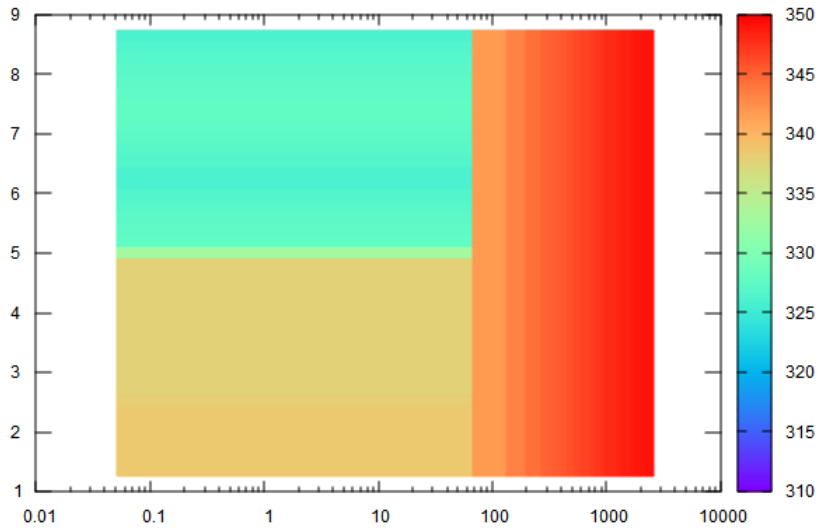

Figura 7.1: Teste 2, pressão ao longo do reservatório. A pressão é menor (região verde), por ser a região aberta à produção.

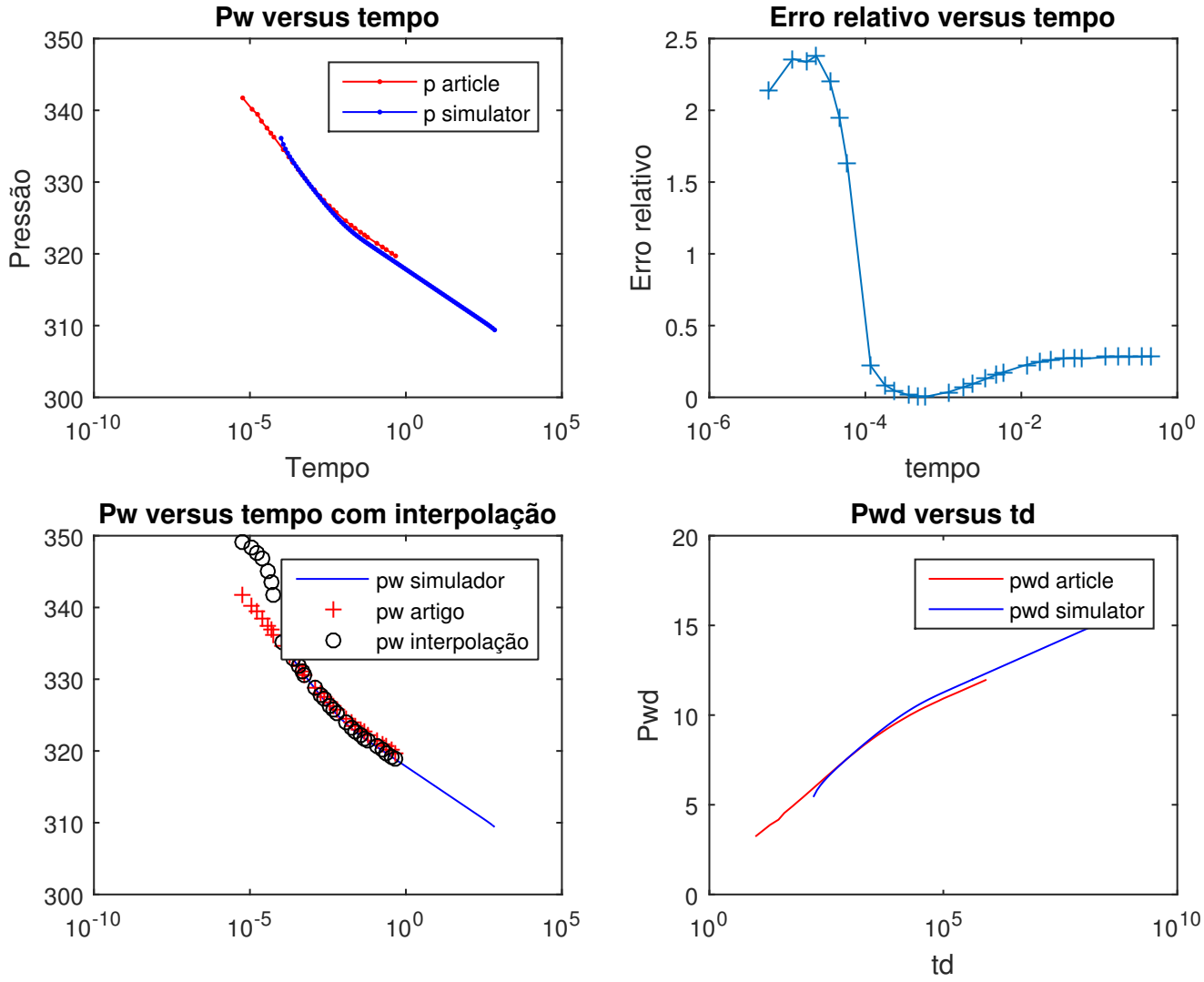

Figura 7.2: No canto superior esquerdo, é apresentada a comparação da pressão do poço do simulador com o artigo. À esquerda, o erro ao longo do tempo. Em baixo na esquerda, é apresentado a comparação com interpolação do artigo, e na direita, a comparação com valores adimensionais.

Desconsiderando os primeiros pontos de resultado do simulador, o erro ficou próximo à 0,4%.

7.2 Teste 2

No teste 2, buscou-se o cenário onde b=1/4 e $h_D = 250$. Para isso, foi utilizado as mesmas propriedades do fluido do caso anterior, com mudanças em:

- $h = 44.7212m$;
- $r_w = 0.1m$;
- $h_w = 11.1803m$;

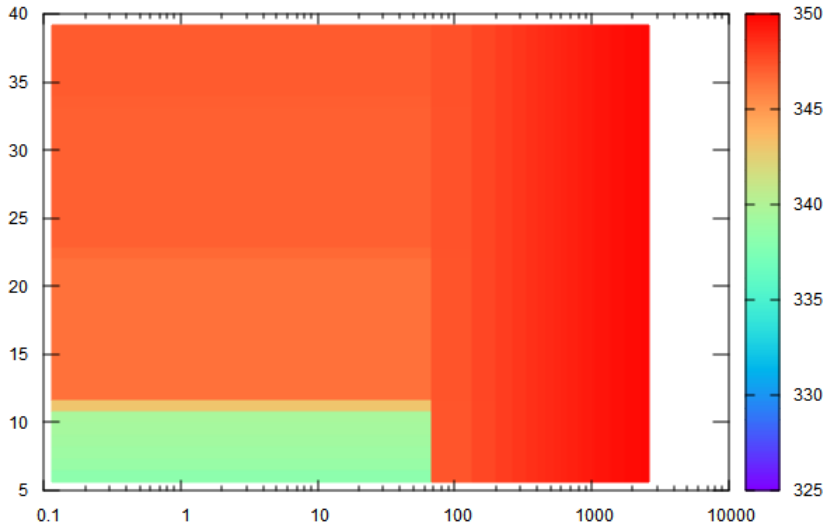

Figura 7.3: Teste 2, pressão ao longo do reservatório. A pressão é menor (região verde), por ser a região aberta à produção.

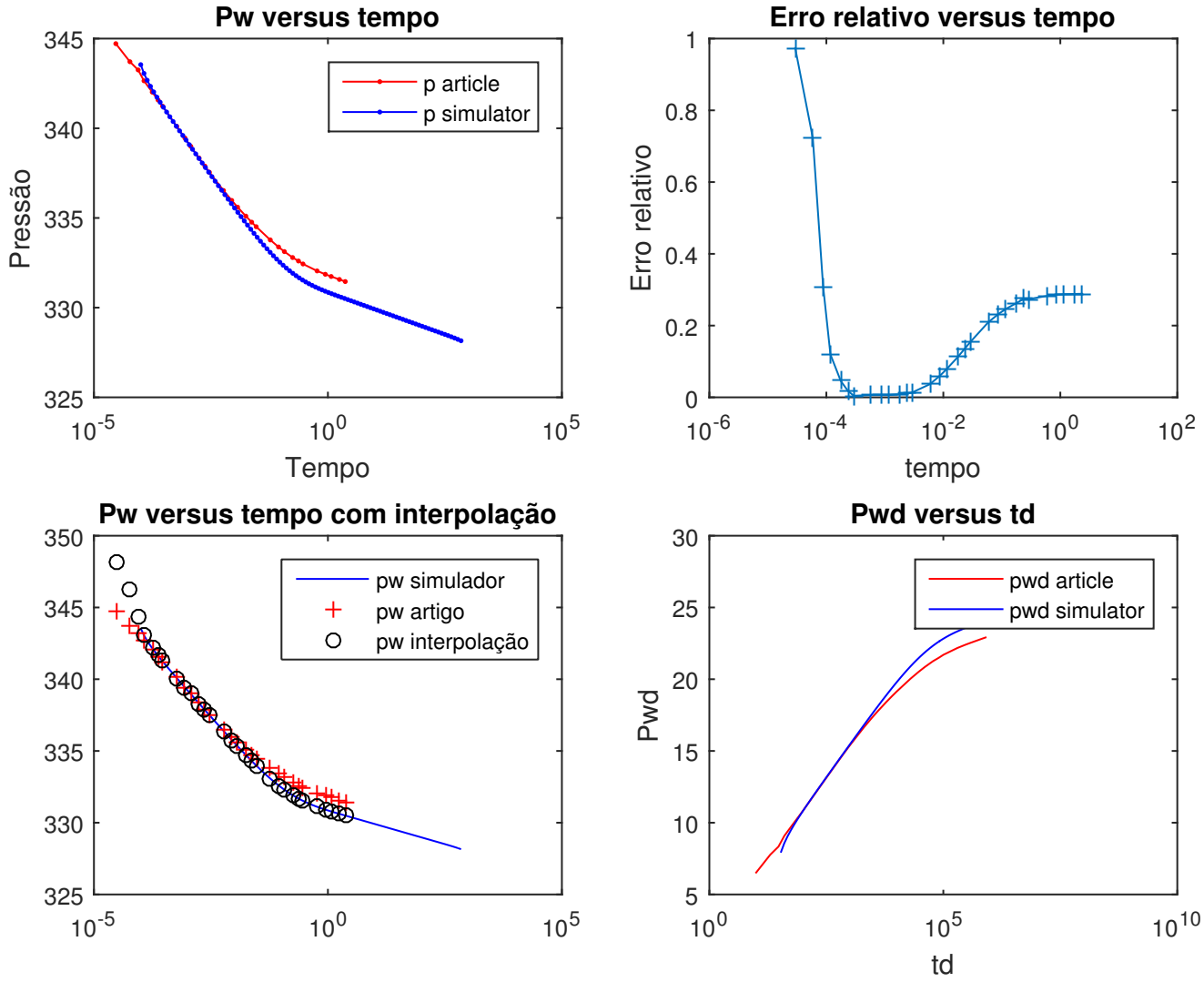

Figura 7.4: No canto superior esquerdo, é apresentada a comparação da pressão do poço do simulador com o artigo. À esquerda, o erro ao longo do tempo. Em baixo na esquerda, é apresentado a comparação com interpolação do artigo, e na direita, a comparação com valores adimensionais.

Desconsiderando os primeiros pontos de resultado do simulador, o erro ficou próximo à 0,38%.

É importante citar que o simulador desenvolvido varia as propriedades do fluido e da rocha com a pressão e foi modelado por volumes finitos, aumentando a precisão. O artigo comparado, não cita essa característica, tratando do assunto de forma mais abrangente. Além disso, por ser dos anos 1977, possuía menos poder computacional para simulações complexas.

Portanto, o erro do simulador com o artigo poderia ser parcialmente explicado por essa alta precisão das propriedades, da modelagem e avanço tecnológico.

Capítulo 8

Documentação

Todo projeto de engenharia precisa ser bem documentado. Neste sentido, apresenta-se neste capítulo a documentação de uso do "Simulador Monofásico 2D". Esta documentação tem o formato de uma apostila que explica passo a passo como usar o software.

8.1 Documentação do usuário

Descreve-se aqui o manual do usuário, um guia que explica, passo a passo a forma de instalação e uso do software Simulador Monofásico 2D.

8.1.1 Como rodar o software

Para rodar o software é necessário:

- Abrir o arquivo input.dat ou input.txt e preencher com as informações tal como será apresentado a seguir:
- Executar o arquivo main.cpp via terminal ou abri-lo diretamente de seu compilador C++ de preferência.
- Seguir as instruções auto-explicativas do programa enquanto ele é executado.

Aqui cabe uma resalva aos arquivos lidos do disco. Por mais que sejam arquivos-exemplo, eles são modelos de funcionamento e sua organização não pode ser alterada, caso seja, o simulador não rodará adequadamente. É preferível que a entrada seja por arquivo de disco, o que tornará a simulação menos laborosa. Porém, caso prefira, a entrada de dados pode ser manual diretamente na tela do simulador.

8.2 Documentação para desenvolvedor

Apresenta-se nesta seção a documentação para o 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).

A seguir é apresentado o arquivo a ser preenchido. Ao lado, uma explicação básica de cada termo:

```
---------------------------------- Simulador Monofásico 2D ----------------------------------
------------------------------------- Nicholas e Kevin -------------------------------------
-------------------------------------------------------------------------------------------

# Poço periodos: 2
tp: 0 1000        /// tempos de mudança na vazão na superfície [h]
qsc: 0 -500       /// vazões nos tempos de mudanca[m^3std/dia]
nz: 5             /// número de camadas do reservatório
dz | aberto/fechado  /// indicação da camada / aberto = 1; fechado = 0
1 | 1
1 | 0
1 | 0
1 | 0
1 | 0
rw: 0.09486       /// raio do poço

# Reservatório
isLiquid: 1       /// 1 = líquido; 0 = gás
re: 3000.0        /// raio externo do reservatório
theta: 0.52359866666      /// ângulo estudado do reservatorio
k0r: 500          /// permeabilidade horizontal
k0z: 100          /// permeabilidade vertical
cphi: 1.0e-4      /// compressibilidadae da formação
```

```
phi0: 0.2      /// porosidade inicial
p0: 1.033512      /// pressão de referência
p_i: 350.0      /// pressão inicial
S: 0      /// fator de película
Temperature: 353.15      /// temperatura do reservatorio

# Discretização
nr: 20      /// quantidade de volumes na largura
nrs: 1      /// quantidade de volumes na região danificada
nt: 100      /// quantidade de tempos
ntp: 100      /// quantidade de tempos
max_iter: 24      /// número máximo de iterações
dtmin: 1.0e-4      /// passo de tempo mínimo[h]
eps_NR: 1.0e-6      /// tolerância de convergência dos resíduos
eps_MB: 1.0e-8      /// tolerância de convergência do balanço de materiais
Ac: 24      /// constante de conversão de unidades acúmulo (ANP)
Bc: 0.0083621472      /// constante de conversão de unidades fluxo (ANP)

# Liquido
cf: 14.7e-5      /// compressibilidade do fluido[cm^2/kgf]
b0: 1.0      /// inverso do fator volume formação na pressão p0 [m^3std/m^3]
p0: 1.0335123      /// pressao de referência [kgf/cm^2]
mu: 1.0      /// viscosidade [cp]
cmu: 0.0
```

8.2.2 Como gerar a documentação usando doxygen

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

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

Para instalação do Doxygen foi criado a seguinte metodologia:

1. Acessar o link Doxygen;
2. Aba Downloads -> Sources and Binaries -> doxygen-1.9.2-setup.exe (48.0MB);
3. Proceder com a instalação;
4. Abrir doxywizard;
5. Seguir passo a passo no Tutorial Doxygen - Criando a documentação do seu programa pelo link Passos para documentar.
6. Teclar Run -> Run doxygen -> Show html input.

Gera-se então, em um diretório de escolha do usuário, saídas em latex, html (index.html) e um relatório no formato rtf (*Rich Text Format*) com todos os arquivos apresentados no link em html. Pode-se salvar o progresso em uma Doxyfile para não perder de vista.

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

A Figura 8.1 mostra o cabeçalho do projeto que contém logo do projeto, sinopse e versão.

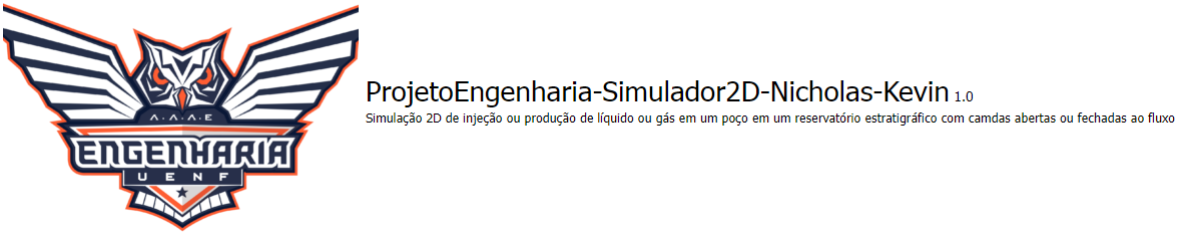

Figura 8.1: Cabeçalho do projeto - doxygen.

A Fig. 8.2 exibe a tela do doxygen que permite a listar as classes que o programa contém.

Figura 8.2: Lista de classes - doxygen.

Já a Fig. 8.3 exibe a tela do doxygen que permite a listar as arquivos que o programa contém.

Figura 8.3: Lista de arquivos - doxygen.

Por fim, a Fig. 8.4 mostra a tela do doxygen que permite acessar, por exemplo, o código da classe CSimulador.hpp.

CSimulador.hpp

Vá para a documentação desse arquivo.

```
1  #ifndef CSIMULADOR_HPP
2  #define CSIMULADOR_HPP
3
4  #include <vector>
5  #include <string>
6  #include <iostream>
7  #include <iomanip>
8  #include <ctime>
9  #include <fstream>
10
11 #include "CGas.hpp"
12 #include "CWell.hpp"
13 #include "CGrid.hpp"
14 #include "CProps.hpp"
15 #include "CMatrix.hpp"
16 #include "CFluido.hpp"
17 #include "CGnuplot.hpp"
18 #include "CLiquido.hpp"
19 #include "CReservoir.hpp"
20 #include "CDiscretization.hpp"
21
22 class CSimulador {
23 public:
24     CSimulador();
25
26     void run();
27 private:
28     const double PI = 3.141592;
29     CWell* well;
30     CGrid* grid;
31     CFluido* fluido;
32     CReservoir* reservoir;
33     CDiscretization* discretization;
34
35     std::vector<std::vector<double>> Pressure;
36     std::vector<double> WellPressure;
37
38 private:
39     std::vector<double> calc_H(CGrid* grid, CProps* props_n, CProps* props_nu, double dt);
40     std::vector<double> calc_Q(CGrid* grid, CWell* well, double time);
41     std::vector<double> calc_X(double pw, std::vector<double> p);
42     std::vector<std::vector<double>> calc_T(CGrid* grid, CProps* props_n);
43
44     std::vector<std::vector<double>> calc_eta(CGrid* grid, CProps* props_nu, double dt);
45     std::vector<std::vector<double>> calc_tau(CGrid* grid, CWell* well, CProps* props_nu, std::vector<double> p_nu, double pw_nu);
46
47     void plot(std::vector<double> time, std::vector<double> Pw);
48
49     bool isErrorNotAcceptable(double dt, std::vector<double> R, CGrid* grid, CProps* props_nu, double q, CDiscretization* discretization);
50
51     void read_data_and_start_objects(std::string nameFile);
52 };
53 #endif
```

Figura 8.4: Código fonte CSimulador - doxygen.

Referências Bibliográficas

[Bilhartz and Ramey, 1977] Bilhartz, H. L. and Ramey, H. J. J. (1977). The combined effects of storage, skin, and partial penetration on well test analysis. *Software - Practice and Experience*. 82

[K. Aziz, 1979] K. Aziz, A. S. (1979). *Petroleum Reservoir Simulation*. Applied Science Publishers, 1 edition. 14, 16

[Kareem et al., 2015] Kareem, L. A., Iwalewa, T. M., and Al-Marhoun, M. (2015). New explicit correlation for the compressibility factor of natural gas: linearized z-factor isotherms. 6(3):481–492. 14

[Lee and Eakin, 1966] Lee, A. L., G. M. H. and Eakin, B. E. (1966). The viscosity of natural gases. *Journal of Petroleum Technology*, 18(8):997–1000. 14

[MacDonald and Coats, 1970] MacDonald, R. and Coats, K. (1970). Methods for numerical simulation of water and gas coning. *Society of Petroleum Engineers Journal - SPE-2796-PA*, 10(4):425–436. 14

[Pico, 2018] Pico, C. E. (2018). SimulaÃ§Ã£o numÃ©rica de reservatÃ³$rioslep1841. NotasdeaulaLENEP/CCT/UENF, 1.1, 14$

[ROSA, 2006] ROSA, Adalberto JosÃ©; CARVALHO, R. d. S. X. J. A. D. (2006). *Engenharia de reservatÃ³$riosdepetr^3leo. Intercincia, RiodeJaneiro.$* 1

[T. Ertekin, 2001] T. Ertekin, J. H. Abou-Kassem, G. R. K. (2001). Basic Applied Reservoir Simulation. *SPE Textbook Series. Henry L. Doherty Memorial Fund of AIME, Society of Petroleum Engineers. 16*

Índice Remissivo