

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
PROF. CARLOS ENRIQUE PICO ORTIZ

MACAÉ - RJ
Dezembro - 2021

Sumário

1	Introdução	1
1.1	Escopo do problema	1
1.2	Objetivos	2
2	Especificação	3
2.1	Nome do sistema/produto	3
2.2	Especificação	3
2.2.1	Requisitos funcionais	4
2.2.2	Requisitos não funcionais	5
2.3	Casos de uso	5
2.3.1	Diagrama de caso de uso geral	5
2.3.2	Diagrama de caso de uso específico	6
3	Elaboração	8
3.1	Análise de domínio	8
3.2	Formulação teórica	9
3.2.1	Fluxo monofásico	10
3.2.2	Propriedades dos gases	13
3.2.3	Propriedades dos líquidos	13
3.2.4	Equação geral	14
3.2.5	Jacobiano	15
3.3	Identificação de pacotes – assuntos	16
3.4	Diagrama de pacotes – assuntos	17
4	AOO – Análise Orientada a Objeto	19
4.1	Diagramas de classes	19
4.1.1	Dicionário de classes	20
4.2	Diagrama de sequência – eventos e mensagens	21
4.2.1	Diagrama de sequência geral	22
4.2.2	Diagrama de sequência específico	22
4.3	Diagrama de comunicação – colaboração	23
4.4	Diagrama de máquina de estado	24

4.5	Diagrama de atividades	25
5	Projeto	26
5.1	Projeto do sistema	26
5.2	Projeto orientado a objeto – POO	27
5.3	Diagrama de componentes	29
5.4	Diagrama de implantação	30
6	Implementação	32
6.1	Código fonte	32
7	Teste	80
7.1	Teste 1	80
7.2	Teste 2	82
8	Documentação	85
8.1	Documentação do usuário	85
8.1.1	Como rodar o software	85
8.2	Documentação para desenvolvedor	85
8.2.1	Dependências	86
8.2.2	Como gerar a documentação usando doxygen	87
	Referências Bibliográficas	91
9	Como modificar o arquivo inputdata	93
9.1	Modificando o tipo de fluido	93
9.2	Modificando as camadas do reservatório	96

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 que pode ser aplicada para resolver equações diferenciais parciais a um determinado volume de meio contínuo baseado, por exemplo, nas equações 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 previsões 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, se faz necessário um conhecimento sólido e completo de como ele se comportará e influenciará a dinâmica de pressões no reservatório ([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 quantificar o volume de fluido de completação utilizado em uma operação, por exemplo.

1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:
 - Desenvolver um 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 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 estratigráficas, 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 [$m d$];

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

c_{pf} calor específico 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 eficiência, abrangência e facilidade no reutilização 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 <i>Windows</i> , <i>GNU/Linux</i> ou <i>Mac</i> .

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:	<ol style="list-style-type: none"> 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 líquido 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 usuário ou sistema, ou seja, a interação do usuário com o sistema.

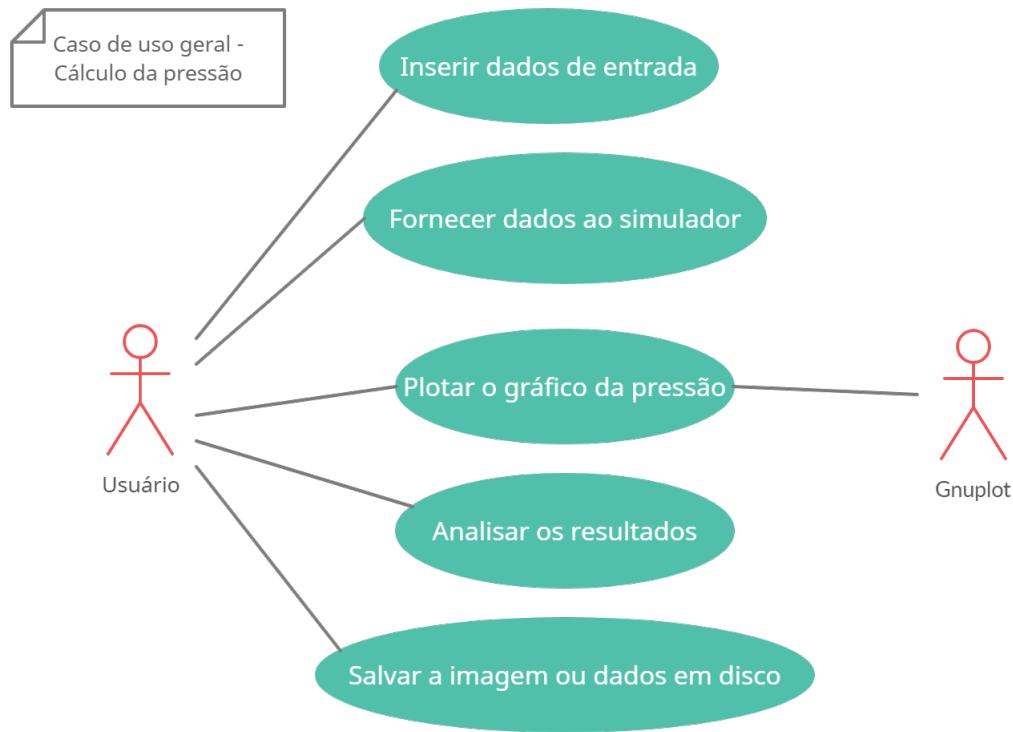


Figura 2.1: Diagrama de caso de uso – Caso de uso geral

2.3.2 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 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.

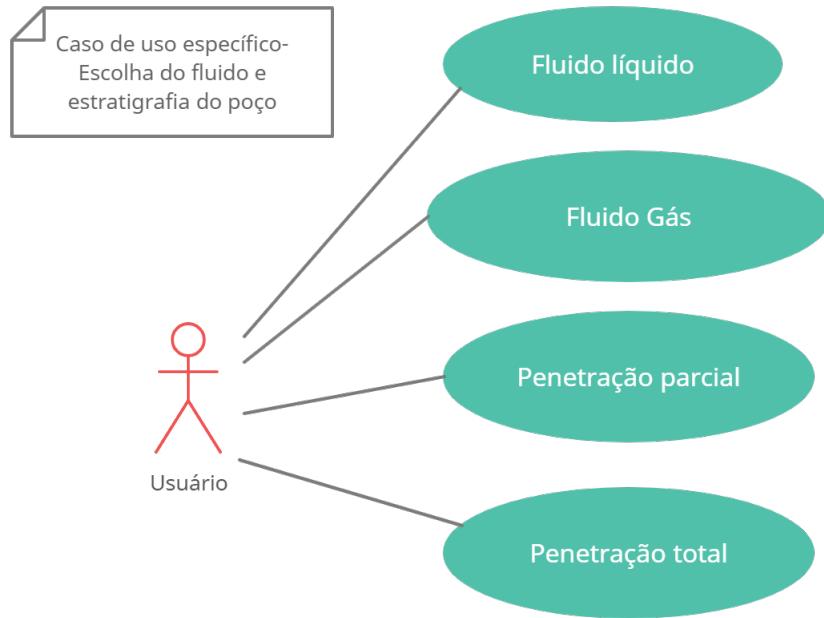


Figura 2.2: Diagrama de caso de uso específico – Escolha do fluido e estratigrafia do poço

Já no segundo caso de uso específico (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.

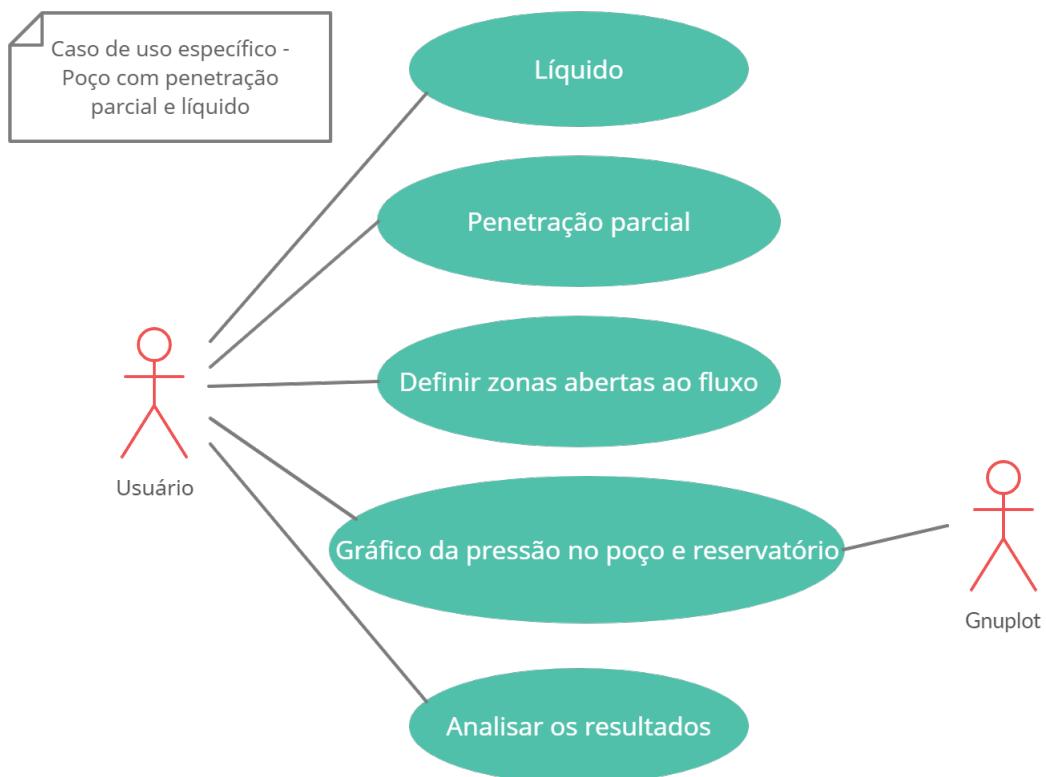


Figura 2.3: Diagrama de caso de uso específico – Poço com penetração parcial e líquido

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 análise 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 e 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 Engenharia de Reservatórios. Trata da utilização e do desenvolvimento de simuladores que buscam prever o comportamento de um reservatório de petróleo e de seus poços associados por meio de modelos matemáticos. Os simuladores podem ser do tipo *black oil* ou compostionais, no primeiro o óleo é considerado uma substância só, e no segundo uma mistura heterogênea.
- 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 profissionalizante. Neste software foi utilizado 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 dele sofrem diversas alterações físico-químicas 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 compreensão e vizualização.
- Software: serão utilizadas métodos e funções já existentes para a resolução de sistemas de matrizes.

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[ROSA, 2006].

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[P., 2014].

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. No entanto, 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 [Pico, 2018].

3.2.1 Fluxo monofásico

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

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

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:

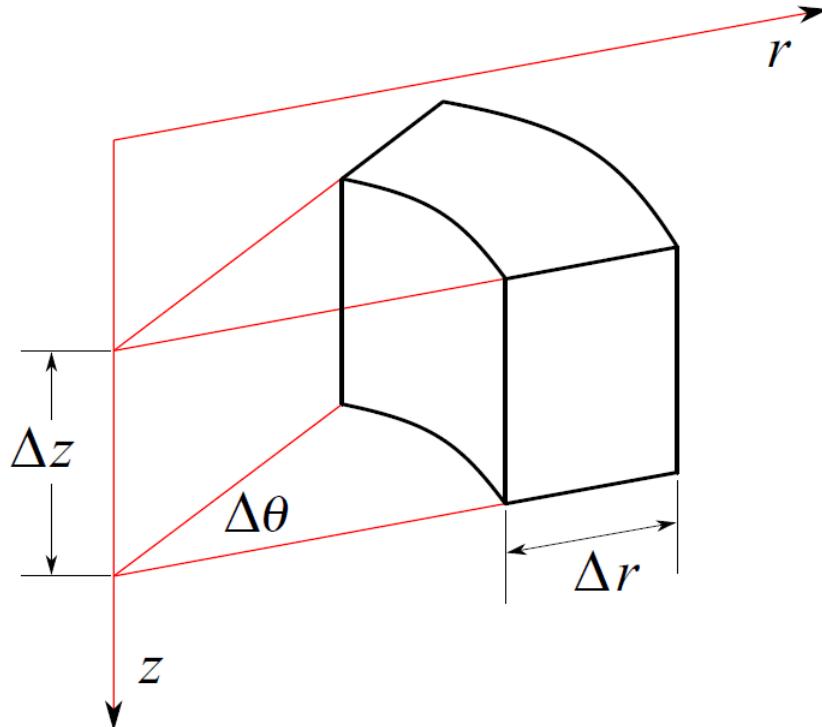


Figura 3.1: Elemento de volume em coordenadas cilíndricas

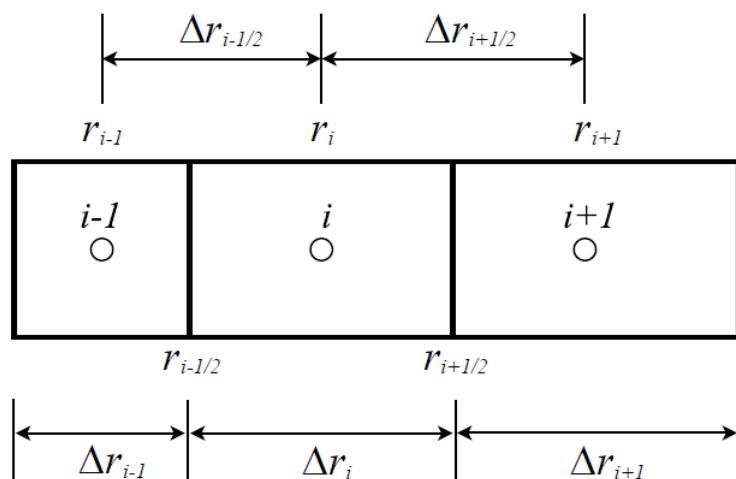


Figura 3.2: Malha radial não-homogênea

Aplicando-se a discretização por volumes finitos, podemos reescrever a Eq. 3.1 como:

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

onde a transmissibilidade T é definida por:

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

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

As propriedades dos fluidos nas interfaces são:

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

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

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

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

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

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

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

Seguem abaixo outras fórmulas utilizadas no desenvolvimento anterior colocadas aqui para não quebrar a linha de raciocínio:

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

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

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

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 um gás está definido pela equação de estado de gás real abaixo:

$$pV = ZnRT \quad (3.14)$$

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:

$$b = \frac{T_o p}{T p_o Z} \quad (3.15)$$

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

$$\rho = \frac{p M_a}{Z RT} \quad (3.16)$$

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

$$-\frac{\partial p}{\partial x} = \frac{\mu}{k} u + \beta \rho u |u| \quad (3.17)$$

onde β é o coeciente de Forchheimer, cuja dimensão é 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];[Pico, 2018]).

3.2.3 Propriedades dos líquidos

No caso de um líquido, as equações foram:

Para inverso do fator volume formação:

$$b_l = b_l^0 (1 + c_l (p_l - p_l^0)) \quad (3.18)$$

Para viscosidade:

$$\mu_l = \mu_l^0 (1 + c_l (p_l - p_l^0)) \quad (3.19)$$

3.2.4 Equação geral

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

$$R = TP + Q - H \quad (3.20)$$

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:

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

onde T é uma matriz não homogênea.

$$Q = [q_{sc}, 0, 0, \dots, 0]^T \quad (3.22)$$

$$H = [0, H_{\{1,1\}}, H_{\{2,1\}}, \dots, H_{\{n_r,n_z\}}]^T \quad (3.23)$$

$$P = [P_w, P_{1,1}, P_{2,1}, \dots, P_{n_r,n_z}]^T \quad (3.24)$$

Sendo:

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

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:

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

$$J = \left[\frac{\partial R_{i,j}}{\partial p_{i,j}} \right]_{nr*nx \times nr*nx} \quad (3.27)$$

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:

$$J = T + \tau - \eta \quad (3.28)$$

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 :

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

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

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

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

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

As derivadas das transmissibilidades são:

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

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

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

resultando em uma matriz com aparência igual ao da Transmissibilidade.

3.3 Identificação de pacotes – assuntos

A partir da análise dos modelos apresentados, pode-se identificar 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 separam, de acordo com suas características: rocha, fluido, aquífero, dados de produção, dados de injeção. Quando juntos, fornecem uma caracterização do reservatório como um todo e servem de base para os cálculos da simulação.
- Simulador: relaciona os pacotes, sendo responsável pela criação e destruição de objetos, assim como interagir com o usuário através de um interface via texto para definir 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 testes 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 matemáticas, teoremas. 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.

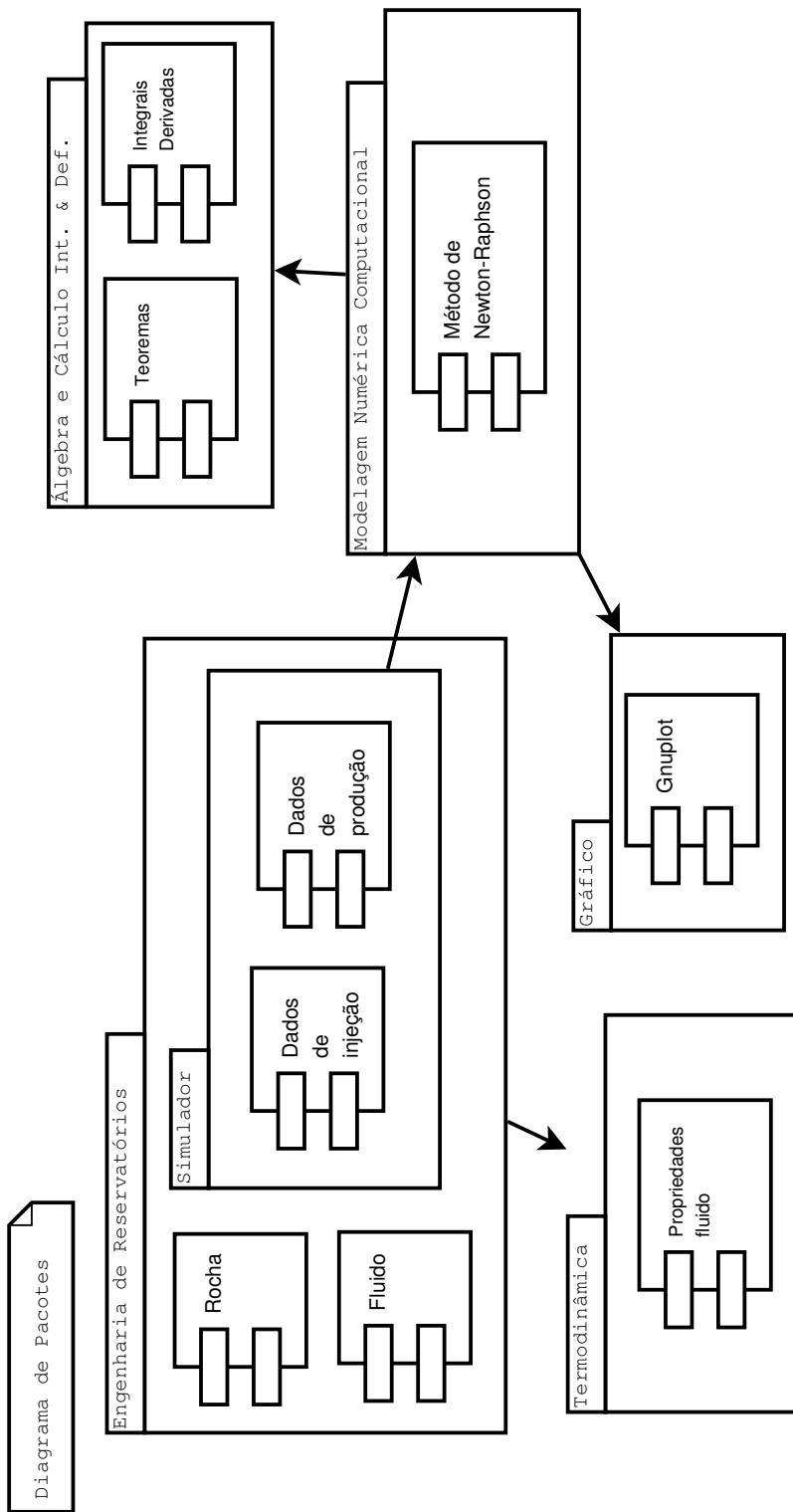


Figura 3.3: Diagrama de Pacotes

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.

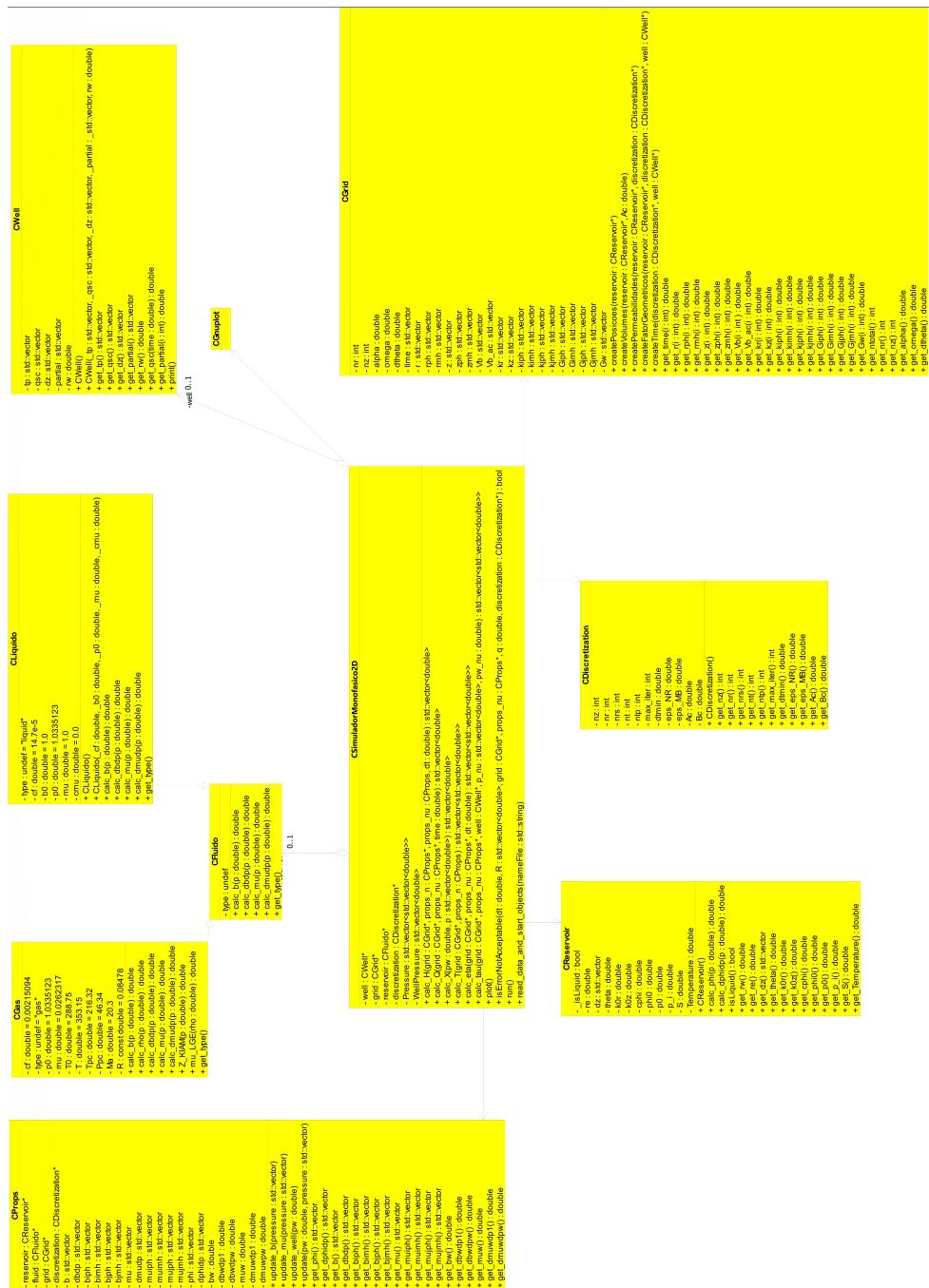


Figura 4.1: Diagrama de classes

4.1.1 Dicionário de classes

- Classe **CGas**: gás que satura o meio poroso, contendo 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 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.

- 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 sendo necessário 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 implementar as derivadas e possuir 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 do tempo e que são estáticas.
- Classe **CWell**: classe que representa o poço com propriedades características.
- Classe **CGnuplot**: classe que fornece os métodos necessários para a geração de gráficos. Sucintamente, o programa externo Gnuplot é uma ferramenta utilizada para criação dos gráficos da distribuição de pressão, em função do tempo obtida pelo simulador para o poço e reservatório.
- Classe **CSimuladorMonofasico2D**: 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 ponto 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:

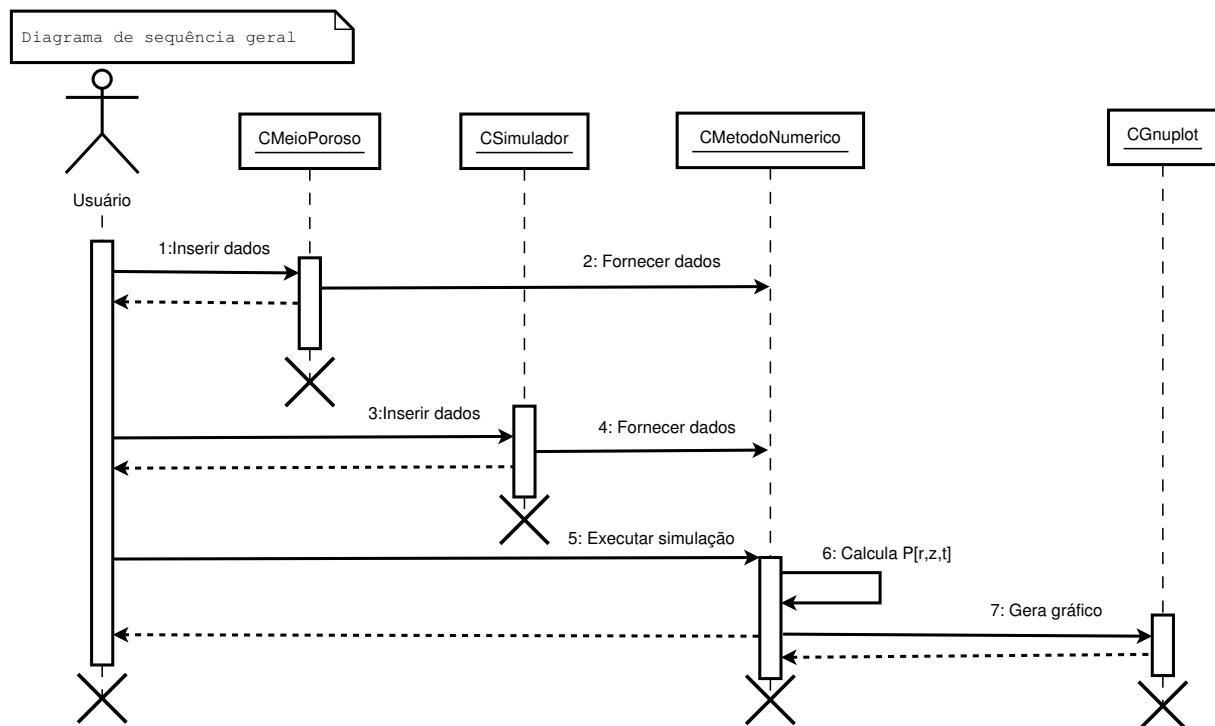


Figura 4.2: Diagrama de sequência geral

4.2.2 Diagrama de sequência específico

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

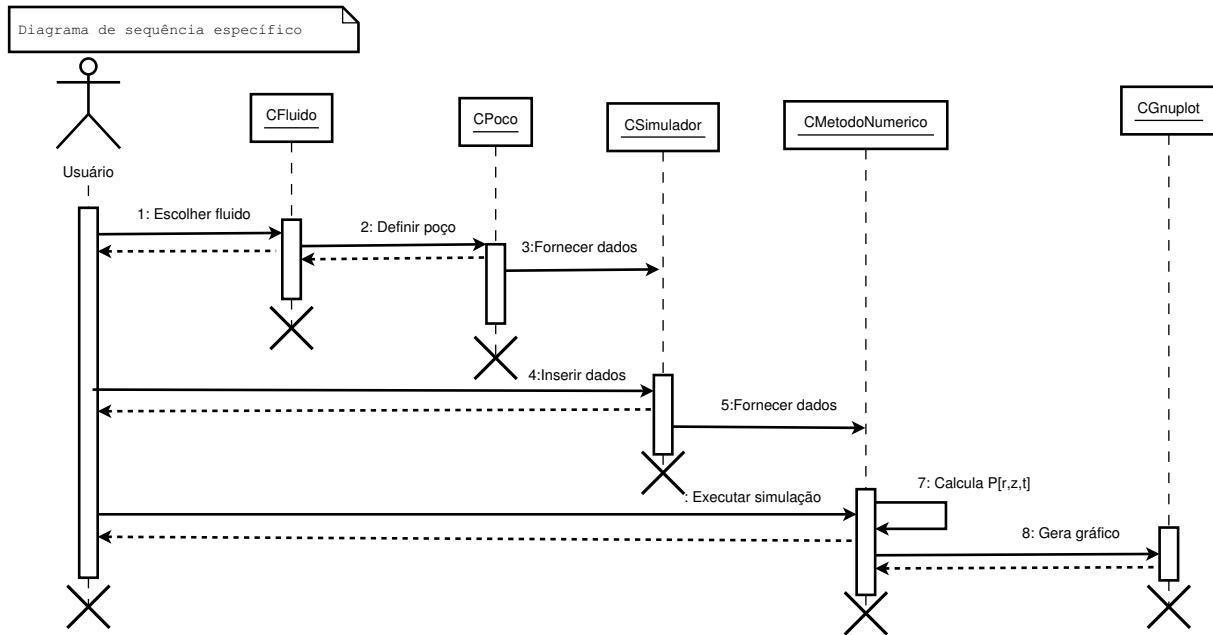


Figura 4.3: Diagrama de sequência específico

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.

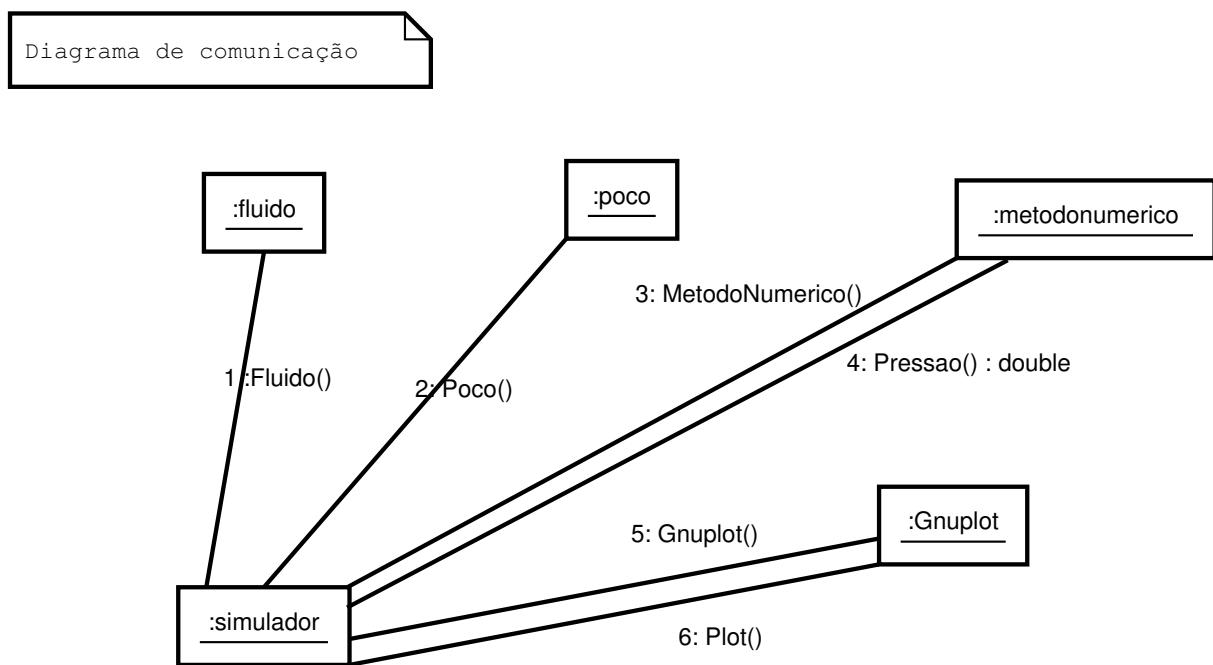


Figura 4.4: Diagrama de comunicação

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 parâmetros 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.

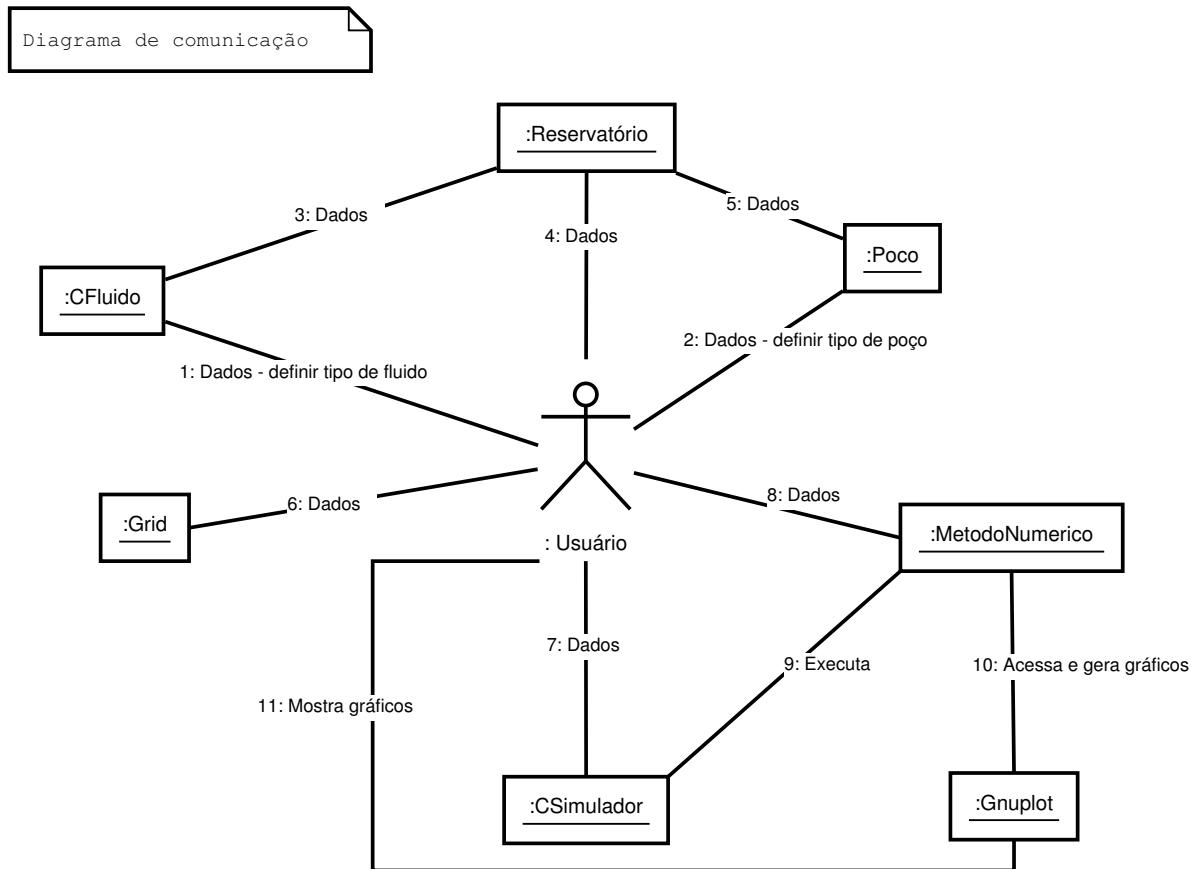


Figura 4.5: Diagrama de comunicação

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 CSimuladorMonofasico2D.

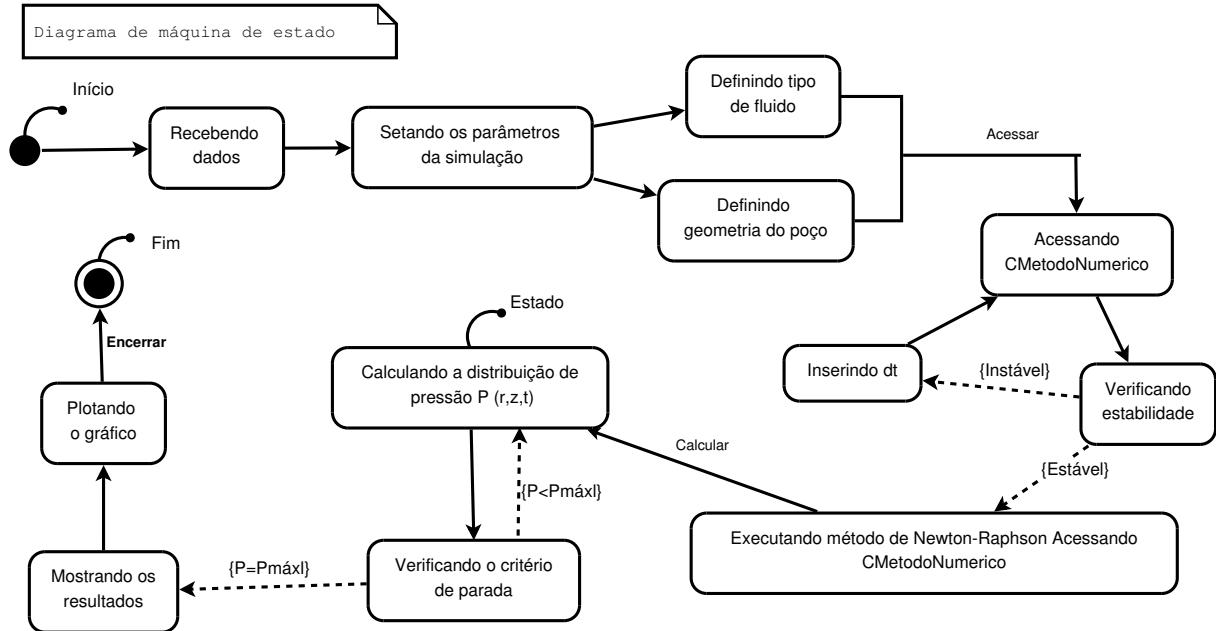


Figura 4.6: Diagrama de máquina de estado

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 $P(r, z, t)$ ”.

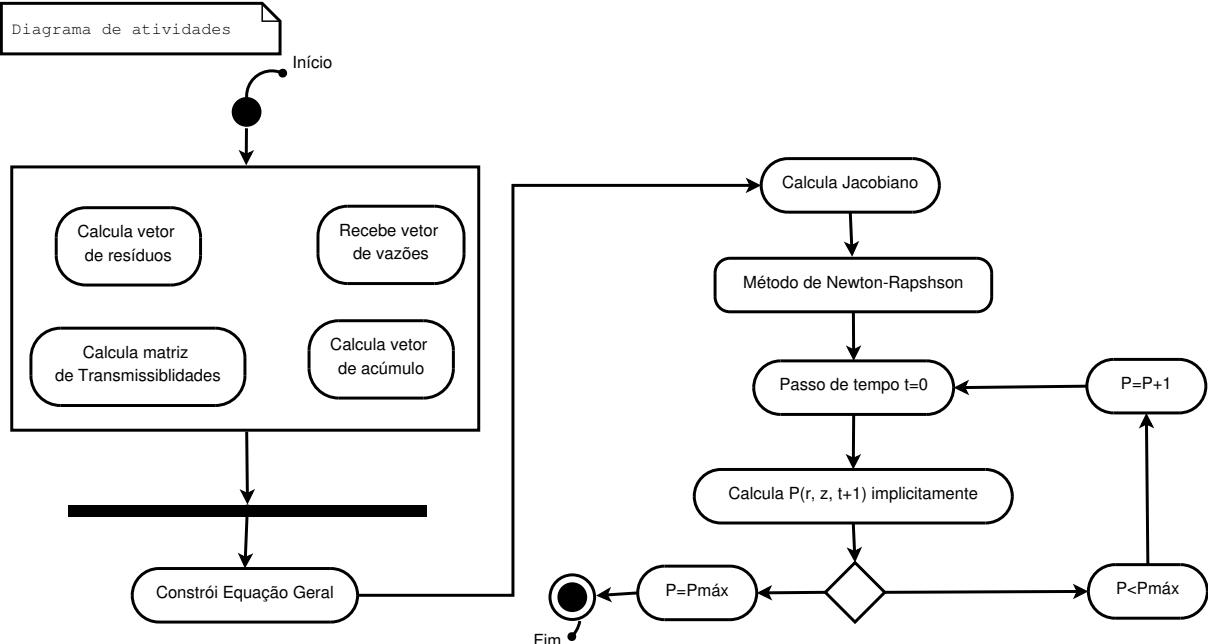


Figura 4.7: Diagrama de atividades

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, o qual envolve etapas como definição dos protocolos, interface API, uso de recursos, subdivisão do sistema em subsistemas, alocação dos subsistemas ao hardware, seleção das estruturas de controle, 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 ferramenta estratégica para resolver o problema, elaborar uma solução e garantir repetibilidade e expansão para problemas similares.

Nessa etapa serão avaliadas 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 plota figuras e gráficos.

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.

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.

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 CSimuladorMonofasico2D, que por sua vez associa-se com as classes CReservoir, CDiscretization, CGrid, CWell, CProps, CFluido, que se associam 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 CSimuladorMonofasico2D 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/métodos de classes com relações de herança.

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. Por ele, podemos perceber as dependências de cada componente. Por exemplo, o componente CSimuladorMonofasico2D depende de todos outros para funcionar. E por sua vez, o CFluido para funcionar, depende do CGas ou CLiquido.

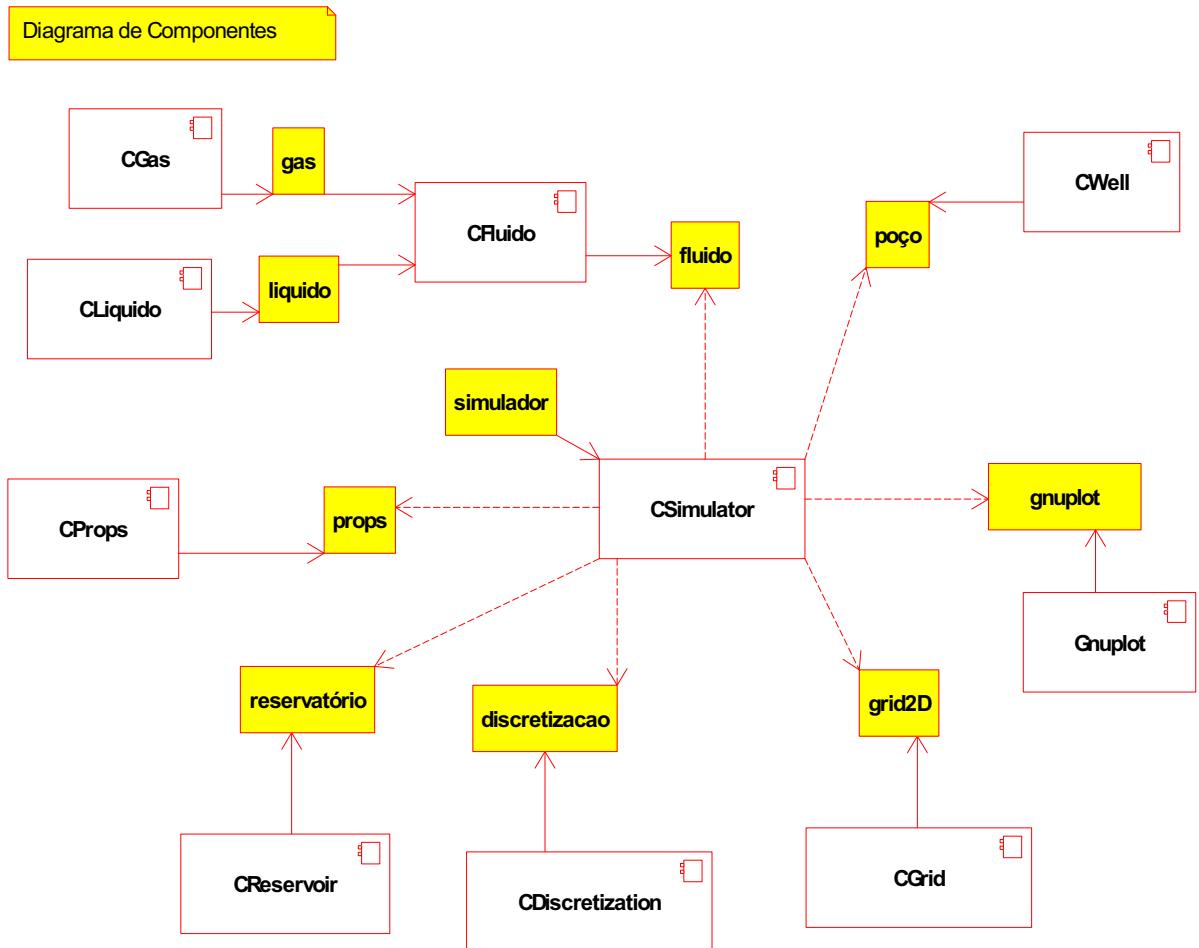


Figura 5.1: Diagrama de componentes

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, 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 fonte de dados

como dados do poço e dados de reservatório.

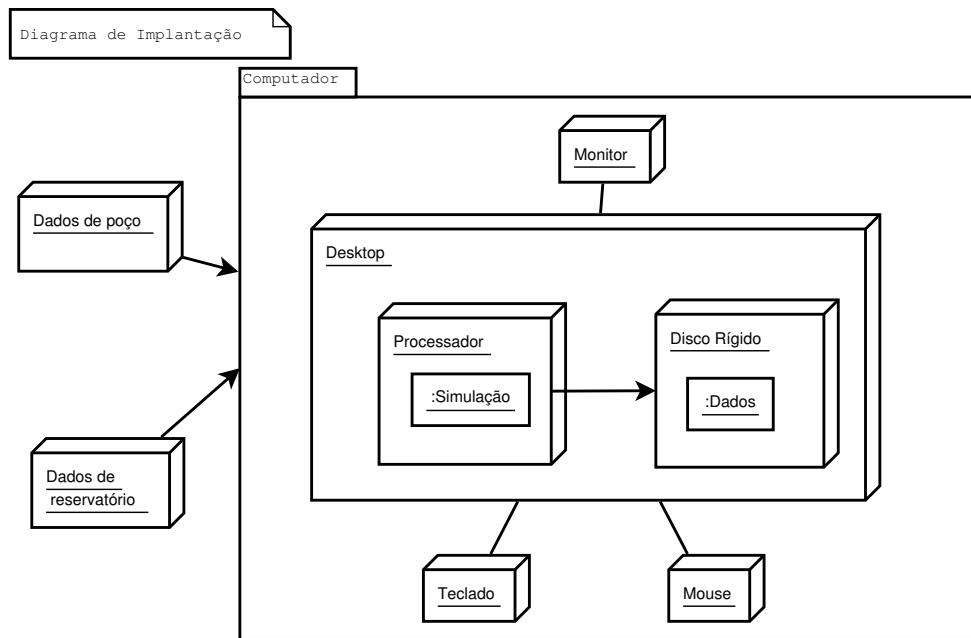


Figura 5.2: Diagrama de implantação

Capítulo 6

Implementação

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

Nota: os códigos devem ser documentados usando padrão **javadoc**. Posteriormente, foi usado o **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>.

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 "CSimuladorMonofasico2D.hpp"
2
3 int main() {
4     CSimuladorMonofasico2D* simulator = new
5         CSimuladorMonofasico2D;
6     simulator->run();
7     return 0;
}
```

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
11     de mudanca na vazao na superficie [h]
12     std::vector<double> qsc{0, -300 };        /// vazoes
13     nos tempos de mudanca [m^3 std / dia]
14     std::vector<double> dz{ 1,1 };            /// altura
15     de cada delta z
16     std::vector<double> partial{1,1};          /// porcentagem de abertura de cada delta z
17     double rw{ 0.09486 };                      /// raio do poco [m]
18
19 public:
20     CWell() {}
21     CWell(std::vector<double> _tp, std::vector<double> _qsc,
22             std::vector<double> _dz, std::vector<double> _partial,
23             double _rw) : tp{ _tp }, qsc{ _qsc }, dz{ _dz }, partial
24             { _partial }, rw{ _rw }){}
25     std::vector<double> Tp() { return tp; }
26     std::vector<double> Qsc() { return qsc; }
27     std::vector<double> Dz() { return dz; }
28     std::vector<double> Partial() { return partial; }
29     double Rw() { return rw; }
30     double Qsc(double time);
31     double Partial(int i) { return partial[i]; }
32     void Print();
33 };
34 #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.

```

1 #include "CWell.hpp"
2
3 double CWell::Qsc(double t) {
4     bool end = true;
5     double q;

```

```

6      for (int i = 0; i < tp.size(); i++) {
7          if (t <= tp[i]) { // esse 1 eh so para gerar uma
8              margem, eh muito literal o igual
9              q = qsc[i];
10             end = false;
11         }
12     return end ? qsc[qsc.size() - 1] : q;
13 }
14
15 void CWell::Print() {
16     std::cout << "\nObjeto Well" << std::endl;
17     std::cout << "tp:" ;
18     for (int i = 0; i < tp.size(); i++) {
19         std::cout << tp[i] << " ";
20     }
21
22     std::cout << "\nqsc:" ;
23     for (int i = 0; i < qsc.size(); i++) {
24         std::cout << qsc[i] << " ";
25     }
26     std::cout << "\ndz | partial:\n";
27     for (int i = 0; i < dz.size(); i++) {
28         std::cout << dz[i] << " | " << partial[i] << std::
29             endl;
30     }
31     std::cout << "rw:" << rw << std::endl;
}

```

Apresenta-se na listagem 6.4 o arquivo com o código da classe CSimuladorMonofásico2D.

Listing 6.4: Arquivo de cabeçalho da classe CSimuladorMonofásico2D.

```

1 #ifndef CSimuladorMonofasico2D_HPP
2 #define CSimuladorMonofasico2D_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 "CLiquid.hpp"
19 #include "CReservoir.hpp"
20 #include "CDiscretization.hpp"

21
22 class CSimuladorMonofasico2D {
23 public:
24     CSimuladorMonofasico2D();
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;           /// todas
36                                         // as pressões em todo o tempo
37                                         std::vector<double> WellPressure;          ///
38                                         // todas as pressões do poço
39
40 private:
41     std::vector<double> calc_H(CProps* props_n, CProps*
42                                 props_nu, double dt);
43     std::vector<double> calc_Q(double time);
44     std::vector<double> calc_X(double pw, std::vector<double> p
45                                );
46     std::vector<std::vector<double>> calc_T(CProps* props_n);
47
48     std::vector<std::vector<double>> calc_eta(CProps* props_nu,
49                                              double dt);
50     std::vector<std::vector<double>> calc_tau(CProps* props_nu,
51                                              std::vector<double> p_nu, double pw_nu);
```

```

47     void plot(std::vector<double> time, std::vector<double> Pw)
48     ;
49
50     bool isErrorNotAcceptable(double dt, std::vector<double> R,
51                               CProps* props_nu, double q);
52
53     void read_data_and_start_objects(std::string nameFile);
54 };
54#endif

```

Apresenta-se na listagem 6.5 o arquivo com o código da implementação da classe CSimuladorMonofasico2D.

Listing 6.5: Arquivo de implementação da classe CSimuladorMonofasico2D.

```

1 #include "CSimuladorMonofasico2D.hpp"
2
3 CSimuladorMonofasico2D::CSimuladorMonofasico2D() {
4     std::cout << "Nome do arquivo:";
5     std::string nameFile;
6     std::cin >> nameFile;
7
8     //nameFile = "input.dat";
9     std::cout << nameFile << std::endl;
10    //std::cin >> nameFile;
11    read_data_and_start_objects(nameFile);
12}
13
14 void CSimuladorMonofasico2D::run() {
15     CProps* props_n = new CProps(grid, fluido, reservoir,
16                                   discretization);
17     CProps* props_nu = new CProps(grid, fluido, reservoir,
18                                   discretization);
19
20     /// var de ajuda
21     int nr = discretization->Nr();
22     int nz = discretization->Nz();
23     int iteracoes = 0;
24     double erro_MB = 1;
25     double erro_NR = 1;
26     double dt;
27
28     /// variaveis das pressoes - pressao no tempo anterior /

```

```

        pressao_iteracao n / pressao_iteracao n+1
27      double pw_n           = reservoir->P_i();
28      double pw_nu          = reservoir->P_i();
29      std::vector<double> Pw(grid->Nt());
30      std::vector<double> p_n(nr * nz, reservoir->P_i());
31      std::vector<double> p_nu(nr * nz, reservoir->P_i());
32
33      /// começo a preparar o loop
34      std::vector<double> H;
35      std::vector<double> Q;
36      std::vector<double> X;
37      std::vector<std::vector<double>> T;
38      std::vector<double> R(nr*nz+1);
39
40      std::vector<std::vector<double>> eta(nr*nz+1, std::vector<
41          double>(nr*nz+1));
42      std::vector<std::vector<double>> tau(nr * nz + 1, std::vector<
43          double>(nr * nz + 1));
44      std::vector<std::vector<double>> J(nr * nz + 1, std::vector<
45          double>(nr * nz + 1));
46      std::vector<double> dX;
47
48      Pw[0] = reservoir->P_i();
49
50      clock_t begin_time;
51      /// loop do tempo
52      for (unsigned int t = 1; t < grid->Nt(); t++) {
53          dt = grid->Time(t) - grid->Time(t - 1);
54
55          props_n->Update(pw_n, p_n);
56
57          begin_time = clock();
58
59          do {
60              props_nu->Update(pw_nu, p_nu);
61
62              H = calc_H( props_n, props_nu, dt);
63              Q = calc_Q( grid->Time(t));
64              X = calc_X(pw_nu, p_nu);
65              T = calc_T( props_nu);
66
67              for (int i = 0; i < nr * nz + 1; i++) { ///

```

```

a linha da multiplicacao
65      R[i] = 0.0;
66      for (int j = 0; j < nr * nz + 1; j++)
67          /// os termos da
68          multiplicacao
69          R[i] += T[i][j] * X[j];
70
71
72      /// matriz jacobiana
73      eta = calc_eta( props_nu, dt);
74      tau = calc_tau( props_nu, p_nu, pw_nu);
75
76      for (int i = 0; i < nr * nz + 1; i++) /// a
77          linha da multiplicacao
78          for (int j = 0; j < nr * nz + 1; j++)
79              /// os termos da
80              multiplicacao
81              J[i][j] = T[i][j] + tau[i][j] - eta[i][j];
82
83      dX = CMatrix::LU_solver(J, R);
84      //dX = CMatrix::GaussSolver(R, J); ///
85      calculo a solucao do sistema linear
86
87      iteracoes++;
88  } while (iteracoes < 10 && isErrorNotAcceptable(dt,
89      R, props_nu, Q[0]));
90
91      std::cout << "Tempo:" << grid->Time(t) << " "
92      iteracoes" << iteracoes << " "
93      float(clock() - begin_time) / CLOCKS_PER_SEC <<
94      std::endl;
95
96      p_n = p_nu;
97      pw_n = pw_nu;
98      iteracoes = 0;
99      Pw[t] = pw_nu;

```

```

95                 Pressure.push_back(p_n);
96             }
97             CMatrix::mostrarMatriz(Pw, "Pw");
98             plot(grid->Time(), Pw);
99             plotSurface();
100        }
101
102 std::vector<double> CSimuladorMonofasico2D::calc_H(CProps* props_n,
103             CProps* props_nu, double dt) {
104             std::vector<double> H(grid->Ntotal()+1, 0.0);
105             for (int i = 0; i < grid->Ntotal(); i++)
106                 H[i+1] = grid->Vb_ac(i) * (props_nu->B(i)* props_nu
107                     ->Phi(i) - props_n->B(i)* props_n->Phi(i)) / dt;
108             return H;
109 }
110
111 std::vector<double> CSimuladorMonofasico2D::calc_Q( double time) {
112     std::vector<double> Q(grid->Ntotal() + 1, 0.0);
113     Q[0] = well->Qsc(time) * (grid->DTheta() / (2 * PI));
114     return Q;
115 }
116
117 std::vector<double> CSimuladorMonofasico2D::calc_X(double pw, std::
118             vector<double> p) {
119             std::vector<double> X(p.size() + 1);
120             X[0] = pw;
121             for (unsigned int i = 1; i < X.size(); i++)
122                 X[i] = p[i-1];
123             return X;
124         }
125
126 std::vector<std::vector<double>> CSimuladorMonofasico2D::calc_T(
127             CProps* props_n) {
128             int nt = grid->Ntotal()+1;
129             int nr = grid->Nr();
130             int nz = grid->Nz();
131             int i;
132
133             double Right;
134             double Left;
135             double Bottom;
136             double Top;

```

```

133     double Well;
134
135     std::vector<std::vector<double>> T(nt, std::vector<double>(
136         nt, 0.0));
137
138     T[0].resize(nt, 0.0);
139     for (int k = 1; k < nt; k++) { // rodo sem o poco
140         i = k - 1;
141         //T[k].resize(nt, 0.0);
142         Right = 0.0;
143         Left = 0.0;
144         Bottom = 0.0;
145         Top = 0.0;
146         Well = 0.0;
147
148         /// passo por todas as colunas da matriz
149         if (i - nr >= 0) {
150             Top = grid->Gjmh(i) * props_n->Bjmh(i) /
151                 props_n->Mujmh(i);
152             T[k][k - nr] = Top;
153         }
154         if (i + nr < nt-1) {
155             Bottom = grid->Gjph(i) * props_n->Bjph(i) /
156                 props_n->Mujph(i);
157             T[k][k + nr] = Bottom;
158         }
159         if ((i - 1) % nr >= 0) {
160             Left = grid->Gimh(i) * props_n->Bimh(i) /
161                 props_n->Muimh(i);
162             T[k][k - 1] = Left;
163         }
164         if ((i + 1) % nr > 0) {
165             Right = grid->Giph(i) * props_n->Biph(i) /
166                 props_n->Muiph(i);
167             T[k][k + 1] = Right;
168         }
169         if (i % nr == 0) {
170             Well = well->Partial((int)i/nr) * grid->Gw
171                 ((int)i/nr) * props_n->Bw() / props_n->
172                 Muw();
173             T[k][0] = Well;
174             T[0][k] = Well;

```

```

168                     T[0][0] -= Well;
169                 }
170             T[k][k] = -Top - Bottom - Right - Left - Well;
171         }
172     return T;
173 }
174
175 std::vector<std::vector<double>> CSimuladorMonofasico2D::calc_eta(
176     CProps* props_nu, double dt) {
177     int nt = grid->Ntotal() + 1;
178     int nr = grid->Nr();
179     int nz = grid->Nz();
180
181     std::vector<std::vector<double>> eta(nt, std::vector<double
182         >(nt, 0.0));
183     eta[0].resize(nt, 0.0);
184     for (int i = 1; i < nt; i++) {
185         eta[i][i] = grid->Vb_ac(i-1)*(props_nu->B(i-1)*
186             props_nu->Dphidp(i-1) + props_nu->Phi(i-1) *
187             props_nu->Dbdp(i-1)) / dt;
188     }
189     return eta;
190 }
191
192
193 std::vector<std::vector<double>> CSimuladorMonofasico2D::calc_tau(
194     CProps* props_nu, std::vector<double> p_nu, double pw_nu) {
195     int nt = grid->Ntotal();
196     int nr = grid->Nr();
197     int nz = grid->Nz();
198
199     double omega = grid->Omega();
200
201     std::vector<std::vector<double>> tau(nt + 1, std::vector<
202         double>(nt+1, 0.0));
203
204     /// bottom
205     for (int i = 0; i < nt - nr; i++) {
206         tau[i + 1][i + 1 + nr] = (p_nu[i] - p_nu[i + nr]) *
207             grid->Gjph(i)
208             * (props_nu->Dbdp(i) * (props_nu->Mu(i + nr
209                 ) + props_nu->Mu(i))
210                 - props_nu->Dmudp(i) * (props_nu->B

```

```

202                               (i + nr) + props_nu->B(i)))
203             / pow(props_nu->Mu(i) + props_nu->Mu(i + nr
204               ), 2);
205             tau[i + 1][i + 1] += tau[i + 1][i + 1 + nr];
206         }
207
208         /// top
209         for (int i = nr; i < nt; i++) {
210             tau[i + 1][i + 1 - nr] = (p_nu[i] - p_nu[i - nr]) *
211               grid->Gjmh(i)
212               * (props_nu->Dbdp(i) * (props_nu->Mu(i - nr
213                 ) + props_nu->Mu(i))
214                 - props_nu->Dmudp(i) * (props_nu->B
215                   (i - nr) + props_nu->B(i)))
216             / pow(props_nu->Mu(i) + props_nu->Mu(i - nr
217               ), 2);
218             tau[i + 1][i + 1] += tau[i + 1][i + 1 - nr];
219         }
220
221         /// left
222         for (int i = 1; i < nt; i++) {
223             tau[i+1][i] = grid->Gimh(i) * (1.0 - omega) * (p_nu
224               [i - 1] - p_nu[i])
225               * (props_nu->Dbdp(i) / props_nu->Muimh(i)
226                 - (props_nu->Bimh(i) / pow(props_nu
227                   ->Muimh(i), 2)) * props_nu->
228                   Dmudp(i - 1));
229             tau[i+1][i+1] = tau[i+1][i];
230         }
231
232         /// right
233         for (int i = 1; i < nt - 1; i++) {
234             tau[i][i + 1] = grid->Giph(i) * omega * (p_nu[i +
235               1] - p_nu[i])
236               * (props_nu->Dbdp(i) / props_nu->Muiph(i)
237                 - (props_nu->Biph(i) / pow(props_nu
238                   ->Muiph(i), 2)) * props_nu->
239                   Dmudp(i - 1));
240             tau[i][i] = tau[i][i + 1];
241         }
242
243         /// well

```

```

232     double temp_well;
233     for (int i = 0; i < nz; i++) {
234         tau[0][0] += (grid->Gw(i) / props_nu->Muw()) * (
235             props_nu->Dbwdpw() - (props_nu->Bw() / props_nu
236             ->Muw()) * props_nu->Dmuwdpw())*(p_nu[i*nr] -
237             pw_nu) * well->Partial(i);
238         temp_well = (grid->Gw(i) / props_nu->Muw()) * (
239             props_nu->Dbwdp1() - (props_nu->Bw() / props_nu
240             ->Muw()) * props_nu->Dmuwdp1()) * (p_nu[i * nr]
241             - pw_nu) * well->Partial(i);
242         tau[0][i * nr + 1] += temp_well;
243         tau[0][0] -= temp_well;
244         tau[i * nr + 1][i * nr + 1] -= temp_well;
245         tau[i * nr + 1][0] += (grid->Gw(i) / props_nu->Muw
246             () * (props_nu->Dbwdp1() - (props_nu->Bw() /
247             props_nu->Muw()) * props_nu->Dmuwdp1()) * (p_nu[
248             i * nr] - pw_nu) * well->Partial(i);
249     }
250     return tau;
251 }
252
253
254 void CSimuladorMonofasico2D::plot(std::vector<double> time, std::
255     vector<double> Pw) {
256     std::string name = ("Pw_versus_time");
257
258     std::ofstream outdata; //save data
259     outdata.open((name + ".dat").c_str());
260     outdata << "#time Temperature" << std::endl;
261     for (int i = 0; i < Pw.size(); i++)
262         outdata << time[i] << " " << Pw[i] << std::endl;
263
264     CGnuplot::semilogx((name + ".dat").c_str(), "time", "Pw", (
265         name + ".png").c_str());
266 }
267
268
269 void CSimuladorMonofasico2D::plotSurface() {
270     std::vector<double> pressure = Pressure[Pressure.size() -
271         1];
272
273     std::string name = ("P_versus_xz");
274     std::ofstream outdata; //save data
275     outdata.open((name + ".dat").c_str());

```

```

262
263     outdata << "#xuzPressure" << std::endl;
264     for (int z = 0; z < grid->Nz(); z++)
265         for (int r = 0; r < grid->Nr(); r++)
266             outdata << grid->R(r) << " " << grid->Z(z)
267             << " " << pressure[grid->Nr()*z+r] <<
268             std::endl;
269
270     CGnuplot::surfacePlot((name + ".dat").c_str(), (name + "."
271     png").c_str());
272
273     std::cin.get();
274 }
275
276 bool CSimuladorMonofasico2D::isErrorNotAcceptable(double dt, std::
277     vector<double> R, CProps* props_nu, double q) {
278     double sum_R_MB = R[0], sum_Divisor_MB = 0.0, NR = 0.0;
279     for (int i = 0; i < R.size()-1; i++) {
280         sum_R_MB += R[i+1];
281         sum_Divisor_MB += (grid->Vb_ac(i) * props_nu->Phi(i));
282         NR += R[i + 1] / (grid->Vb_ac(i) * props_nu->Phi(i));
283     }
284     double MB = dt * abs(sum_R_MB) / sum_Divisor_MB;
285
286     double biggestNR = abs(R[0] / q) > NR ? abs(R[0] / q) : NR;
287     return (MB > discretization->Eps_MB() || biggestNR >
288             discretization->Eps_NR());
289 }
290
291 void CSimuladorMonofasico2D::read_data_and_start_objects(std::
292     string nameFile) {
293     std::ifstream file(nameFile);
294
295     std::string text;
296     std::getline(file, text);
297     std::getline(file, text);
298     std::getline(file, text);
299     std::getline(file, text);
300     std::getline(file, text);
301
302 }
```

```
296     /// Pegando as variaveis do Poco
297     std::getline(file, text);
298
299     std::vector<double> tp;
300     std::vector<double> qsc;
301     std::vector<double> dz;
302     std::vector<double> partial;
303
304     int periodos;
305     file >> text; file >> text;
306     periodos = std::stoi(text);
307
308     /// pego os valores dos tps
309     file >> text;
310     for (int i = 0; i < periodos; i++) {
311         file >> text;
312         tp.push_back(std::stof(text));
313     }
314     std::getline(file, text);
315
316     /// pego os valores dos qsc
317     file >> text;
318     for (int i = 0; i < periodos; i++) {
319         file >> text;
320         qsc.push_back(std::stof(text));
321     }
322     std::getline(file, text);
323
324     /// pego os valores do h e partial
325     file >> text; file >> text;
326     periodos = std::stoi(text);
327     std::getline(file, text);
328     std::getline(file, text);
329
330     for (int i = 0; i < periodos; i++) {
331         file >> text;
332         dz.push_back(std::stof(text));
333         file >> text; file >> text;
334         partial.push_back(std::stof(text));
335     }
336
337     file >> text; file >> text;
```

```
338     double rw = std::stof(text);
339     well = new CWell(tp, qsc, dz, partial, rw);
340
341     /**
342      /// RESERVOIR
343      /**
344      std::getline(file, text);           std::getline(file, text);
345      std::getline(file, text);
346      file >> text; file >> text;
347      bool isLiquid = std::stoi(text);
348
349      file >> text; file >> text;
350      double re = std::stof(text); std::getline(file, text);
351
352      file >> text; file >> text;
353      double theta = std::stof(text); std::getline(file, text);
354
355      file >> text; file >> text;
356      double k0r = std::stof(text); std::getline(file, text);
357
358      file >> text; file >> text;
359      double k0z = std::stof(text); std::getline(file, text);
360
361      file >> text; file >> text;
362      double cphi = std::stof(text); std::getline(file, text);
363
364      file >> text; file >> text;
365      double phi0 = std::stof(text); std::getline(file, text);
366
367      file >> text; file >> text;
368      double p0 = std::stof(text); std::getline(file, text);
369
370      file >> text; file >> text;
371      double p_i = std::stof(text); std::getline(file, text);
372
373      file >> text; file >> text;
374      double S = std::stof(text); std::getline(file, text);
375
376      file >> text; file >> text;
377      double Temperature = std::stof(text); std::getline(file,
text);
```

377

```
378     reservoir = new CReservoir(isLiquid, rw, re, dz, theta, k0r
379         , k0z, cphi, phi0, p0, p_i, S, Temperature);
380
381     /**
382      /**
383      std::getline(file, text); std::getline(file, text);
384      int nz = dz.size();
385
386      file >> text; file >> text;
387      int nr = std::stoi(text); std::getline(file, text);
388
389      file >> text; file >> text;
390      int nrs = std::stoi(text); std::getline(file, text);
391
392      file >> text; file >> text;
393      int nt = std::stoi(text); std::getline(file, text);
394
395      file >> text; file >> text;
396      int ntp = std::stoi(text); std::getline(file, text);
397
398      file >> text; file >> text;
399      int max_iter = std::stoi(text); std::getline(file, text);
400
401      file >> text; file >> text;
402      double dtmin = std::stof(text); std::getline(file, text);
403
404      file >> text; file >> text;
405      double eps_NR = std::stof(text); std::getline(file, text);
406
407      file >> text; file >> text;
408      double eps_MB = std::stof(text); std::getline(file, text);
409
410      file >> text; file >> text;
411      double Ac = std::stof(text); std::getline(file, text);
412
413      file >> text; file >> text;
414      double Bc = std::stof(text); std::getline(file, text);
415
416      discretization = new CDiscretization(nz, nr, nrs, nt, ntp,
417          max_iter, dtmin, eps_NR, eps_MB, Ac, Bc);
417
```

```
418     ///
419     /// FLUID
420     ///
421     if (reservoir->isLiquid()) {
422
423         std::getline(file, text); std::getline(file, text);
424         file >> text; file >> text;
425         double cf = std::stof(text); std::getline(file,
426             text);
427
428         file >> text; file >> text;
429         double b0 = std::stof(text); std::getline(file,
430             text);
431
432         file >> text; file >> text;
433         double p0 = std::stof(text); std::getline(file,
434             text);
435
436         file >> text; file >> text;
437         double cmu = std::stof(text); std::getline(file,
438             text);
439         fluido = new CLiquido(cf, b0, p0, mu, cmu);
440     }
441     else {
442
443         std::getline(file, text); std::getline(file, text);
444         file >> text; file >> text;
445         double cf = std::stof(text); std::getline(file,
446             text);
447
448         file >> text; file >> text;
449         double p0 = std::stof(text); std::getline(file,
450             text);
451
452         file >> text; file >> text;
```

```

452         double T0 = std::stof(text); std::getline(file,
453                                         text);
454
455         file >> text; file >> text;
456         double Tpc = std::stof(text); std::getline(file,
457                                         text);
458
459         file >> text; file >> text;
460         double Ppc = std::stof(text); std::getline(file,
461                                         text);
462
463         file >> text; file >> text;
464         double Ma = std::stof(text); std::getline(file,
465                                         text);
466
467         fluido = new CGas(cf, p0, mu, T0, Temperature, Tpc,
468                           Ppc, Ma);
469
470     }
471
472     grid = new CGrid(reservoir, discretization, well);
473 }

```

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     std::vector<double> dz{ 1, 1 };           /// altura
12     do reservatorio
13     double rw{ 0.09486 };                   /// raio interno (poco)
14     double ref{ 3000.0 };                   /// raio externo
15     double theta{3.141592/6};             /// angulo estudado do
16     reservatorio
17     double k0r{500};                      ///
18     permeabilidade horizontal
19     double k0z{ 100 };                     /// permeabilidade

```

```

    vertical
17     double cphi{1.0e-4};                                ///
18         compressibilidadae da formacao
19     double phi0{ 0.2 };                                /// porosidade
20         inicial
21     double p0{1.033512};                                /// pressao de
22         referencia
23     double p_i{350.0};                                /// pressao inicial
24     double s{ 0 };                                    /// fator de
25         pelicula
26     double temperature{353.15};                      /// temperatura do
27         reservatorio

28     bool aquifer{false};                                /// presenca de aquifero
29         por fronteira de Neumann?
30     bool infVol{false};                                /// presenca de
31         aquifero por volume infinito?

32     public:
33         CReservoir() {}
34         CReservoir( bool _isLiquid, double _rw, double _re, std::
35             vector<double> _dz, double _theta, double _k0r, double
36             _k0z, double _cphi, double _phi0, double _p0, double
37             _p_i, double _S, double _Temperature) : _isLiquid{
38             _isLiquid },
39             rw{ _rw }, re{ _re }, dz{ _dz }, theta{ _theta },
40             k0r{ _k0r }, k0z{ _k0z }, cphi{ _cphi }, phi0{
41             _phi0 }, p0{ _p0 }, p_i{ _p_i }, s{ _S },
42             temperature{ _Temperature } {}

43     double calc_phi(double p);
44     double calc_dphidp(double p);

45     /// funcoes get
46     bool isLiquid() { return _isLiquid; }
47     double Rw() { return rw; }
48     double Re() { return re; }
49     std::vector<double> Dz() { return dz; }
50     double Theta() { return theta; }
51     double K0r() { return k0r; }
52     double K0z() { return k0z; }
53     double Cphi() { return cphi; }

```

```

44     double Phi0() { return phi0; }
45     double P0() { return p0; }
46     double P_i() { return p_i; }
47     double S(){ return s; }
48     double Temperature() { return temperature; }
49     bool Aquifer() { return aquifer; }
50     bool InfVol() { return infVol; }
51 };
52 #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*
13             _reservoir, CDiscretization* _discretization);
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     std::vector<double> biph;
26     std::vector<double> bimh;
27     std::vector<double> bjph;
28     std::vector<double> bjmh;
29
30     std::vector<double> mu;
31     std::vector<double> dmudp;
32     std::vector<double> muiph;
33     std::vector<double> muimh;
34     std::vector<double> mujph;
35     std::vector<double> mujmh;
36
37     std::vector<double> phi;
38     std::vector<double> dphidp;
39
40     double bw = .0;
41     double dbwdp1 = .0;
42     double dbwdpw = .0;
43     double muw = .0;
44     double dmudp1 = .0;
45     double dmudpw = .0;
46
47     void update_b(std::vector<double> pressure);
48     void update_mu(std::vector<double> pressure);
49     void update_well(double pw);
50
51 public:
52     void Update(double pw, std::vector<double> pressure);
53
54     /// get com as posicoes desejadas
55     double Phi(int i) { return phi[i]; }
56     double Dphidp(int i) { return dphidp[i]; }
57
58
```

```

59     double B(int i)                                { return b[i]; }
60     double Dbdp(int i)                             { return dbdp[i]; }
61     double Biph(int i)                            { return biph[i]; }
62     double Bimh(int i)                            { return bimh[i]; }
63     double Bjph(int i)                            { return bjph[i]; }
64     double Bjmh(int i)                            { return bjmh[i]; }
65
66     double Mu(int i)                                { return mu[i]; }
67     double Dmudp(int i)                             { return dmudp[i]; }
68     double Muiph(int i)                            { return mujph[i]; }
69     double Muimh(int i)                            { return muimh[i]; }
70     double Mujph(int i)                            { return mujph[i]; }
71     double Mujmh(int i)                            { return mujmh[i]; }
72
73     /// props do poco
74     double Bw()                                     { return bw;
75     ; }
76     double Dbwdp1()                                { return dbwdp1; }
77     double Dbwdpw()                               { return dbwdpw; }
78     double Muw()                                    { return muw; }
79     double Dmuwdp1()                               { return dmuwdp1; }
80     double Dmuwdpw()                             { return dmuwdpw; }
81 };
81 #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(CGGrid* _grid, CFLUIDO* _fluido, CReservoir*
4   _reservoir, CDiscretization* _discretization) {
5   grid = _grid;
6   fluido = _fluido;
7   reservoir = _reservoir;
8   discretization = _discretization;
9
10  nr = discretization->Nr();
11  nz = discretization->Nz();
12
13  b.resize(nr*nz);
14  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->Omega();
60     int camada = -1;
61
62     for (int i = 0; i < nr*nz; i++) {
63         // guardo a camada para uso futuro
64         if (i % nr == 0) camada++;
65
66         /// fronteira externa radial
67         if ((i + 1) % nr == 0)
68             biph[i] = b[i];
69         else
70             biph[i] = (1.0 - omega) * b[i] + omega * b[
71                 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] * grid->Z(camada) + b[i+nr]
84                         * grid->Z(camada+1)) / (grid->Z(camada)
85                         + grid->Z(camada+1));
86
87         /// fronteira superior vertical
88         if (i < nr)
89             bjmh[i] = b[i];
90         else
91             bjmh[i] = bjph[i-nr];
92     }
93
94     void CProps::update_mu(std::vector<double> pressure) {
95         double omega = grid->Omega();
96         int camada = -1;
97
98
99 }
```

```

95     for (int i = 0; i < nr * nz; i++) {
96         // guardo a camada para uso futuro
97         if (i % nr == 0) camada++;
98
99         // fronteira externa radial
100        if ((i + 1) % nr == 0)
101            muiph[i] = mu[i];
102        else
103            muiph[i] = (1.0 - omega) * mu[i] + omega *
104                mu[i + 1];
105
106        // fronteira interna radial
107        if (i % nr == 0)
108            muimh[i] = mu[i];
109        else
110            muimh[i] = muiph[i - 1];
111
112        // fronteira inferior vertical
113        if (i >= (nz - 1) * nr)
114            mujph[i] = mu[i];
115        else
116            mujph[i] = (mu[i] * grid->Z(camada) + mu[i
117                + nr] * grid->Z(camada + 1)) / (grid->Z(
118                camada) + grid->Z(camada + 1));
119
120        // fronteira superior vertical
121        if (i < nr)
122            mujmh[i] = mu[i];
123        else
124            mujmh[i] = mujph[i - nr];
125    }
126}

```

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
7 class CLiquido : public CFluido{
8 public:

```

```

9      CLiquido() {}
10     CLiquido (float _cf, float _b0, float _p0, float _mu, float
11           _cmu) :cf{ _cf }, b0{ _b0 }, p0{ _p0 }, mu{ _mu }, cmu{
12             _cmu } {}
13
14     double calc_b(double p);
15     double calc_dbdp(double p);
16
17     double calc_mu(double p);
18     double calc_dmudp(double p);
19
20 private:
21     std::string type = "liquid";
22
23     double cf{ 14.7e-5 };           /// compressibilidade do
24               fluido [cm^2/kgf]
25     double b0{ 1.0 };              /// inverso do
26               fator volume formacao na pressao p0 [m^3 std / m^3]
27     double p0{ 1.0335123 };        /// pressao de referencia [
28               kgf/cm^2]
29     double mu{ 1.0 };             /// viscosidade [cp
30               ]
31     double cmu{ 0.0 };
32 };
33 #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
3 double CLiquido::calc_b(double p) {      return b0 * (1.0 + cf * (p
4           - p0)); }
5
6 double CLiquido::calc_dbdp(double p) {   return b0 * cf; }
7
8 double CLiquido::calc_mu(double p) {    return mu * (1.0 + cmu * (p -
9           p0)); }
10
11 double 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
11 class CGrid {
12 public:
13     CGrid(CReservoir* reservoir, CDiscretization*
14           discretization, CWell* well);
15
16 private:
17     int nr, nz;
18     double alpha;
19     double omega;
20     double dtheta;
21
22     std::vector<double> time;           /// vetor dos tempos
23
24     std::vector<double> r;              /// posicao do centro do
25             bloco em relacao a x
26     std::vector<double> rph;            /// posicao em x + 1/2
27     std::vector<double> rmh;            /// posicao em x - 1/2
28
29     std::vector<double> z;              /// posicao do centro do
30             bloco em relacao a z
31     std::vector<double> zph;            /// posicao em z + 1/2
32     std::vector<double> zmh;            /// posicao em z - 1/2
33
34     std::vector<double> vb;             /// volume
35     std::vector<double> vb_ac;          /// volume corrigido
36
37     std::vector<double> kr;             /// permeabilidade
38             horizontal
39     std::vector<double> kz;             /// permeabilidade vertical

```

```

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     std::vector<double> Time() { return time; }
59     double Time( int i) { return time[i]; }
60
61     double R( int i) { return r[i]; }
62     double Rph( int i) { return rph[i]; }
63     double Rmh( int i) { return rmh[i]; }
64
65     double Z( int i) { return z[i]; }
66     double Zph( int i) { return zph[i]; }

```

```

67     double Zmh( int i)           { return zmh[i]; }
68
69     double Vb( int i)           { return vb[i]; }
70     double Vb_ac( int i){ return vb_ac[i]; }
71
72     double Kr( int i)           { return kr[i]; }
73     double Kz( int i)           { return kz[i]; }
74
75     double Kiph( int i)          { return kiph[i]; }
76     double Kimh( int i)          { return kimh[i]; }
77     double Kjph( int i)          { return kjph[i]; }
78     double Kjmh( int i)          { return kjmh[i]; }
79
80     double Giph( int i)          { return giph[i]; }
81     double Gimh( int i)          { return gimh[i]; }
82     double Gjph( int i)          { return gjph[i]; }
83     double Gjmh( int i)          { return gjmh[i]; }
84
85     double Gw( int i)           { return gw[i]; }
86
87     int Ntotal()                 { return nr * nz; }
88     int Nr()                     { return nr; }
89     int Nz()                     { return nz; }
90     int Nt()                     { return time.size(); }
91
92     double Alpha()                { return alpha; }
93     double Omega()                { return omega; }
94     double DTheta()               { return dtheta; }
95 };
96 #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
4               , CWell* well) {
5     nr = discretization->Nr();
6     nz = discretization->Nz();
7     dtheta = reservoir->Theta();
8     createPosicoes(reservoir);

```

```

8         createVolumes(reservoir, discretization->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->Re() / reservoir->Rw(), 1.0 / nr);
16     omega = log((alpha - 1) / log(alpha)) / log(alpha);
17
18     /// ----- raio -----
19     r.push_back(reservoir->Rw() * log(alpha) / (1-(1/alpha)));
20     rmh.push_back(reservoir->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->Re());
27
28     /// ----- profundidade -----
29     std::vector<double> dz = reservoir->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
37 void CGrid::createVolumes(CReservoir* reservoir, double Ac) {
38     double _vb;
39     std::vector<double> dz = reservoir->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],
43                 2)) * reservoir->Theta() * dz[k];
44             vb.push_back(_vb );
45             vb_ac.push_back(_vb * Ac);
46         }
47     }
48 }
```

```
49 void CGrid::createPermeabilidades(CReservoir* reservoir,
    CDiscretization* discretization) {
50     int nrs = discretization->Nrs();
51
52     double k0r = reservoir->K0r();
53     double k0z = reservoir->K0z();
54
55     double krs = k0r / (reservoir->S() / log(rph[nrs] /
56         reservoir->Rw()) + 1.0);
57     double kzs = k0z / (reservoir->S() / log(rph[nrs] /
58         reservoir->Rw()) + 1.0);
59
60     /// nos centros dos volumes
61     for (int k = 0; k < nz; k++) {
62         for (int i = 0; i < nr; i++) {
63             /// regiao danificada
64             if (i <= nrs) {
65                 kr.push_back(k0r + krs);
66                 kz.push_back(k0z + kzs);
67             }
68             /// regiao normal
69             else {
70                 kr.push_back(k0r);
71                 kz.push_back(k0z);
72             }
73             /// volume infinito
74             if (reservoir->InfVol() && k == nz-1) {
75                 kr[i + k * nr] = kr[i + k * nr] *
76                     1.0e4; // multiplicador
77                     arbitrario, valor deve ser
78                     grande
79                 kz[i + k * nr] = kz[i + k * nr] *
80                     1.0e4; // multiplicador
81                     arbitrario, valor deve ser
82                     grande
83             }
84
85             /// permeabilidades homogeneas entre as
86             camadas
87             kiph.push_back(k0r);
88             kimh.push_back(k0r);
89
90 }
```

```

81                     kjph.push_back(k0z);
82                     kjmh.push_back(k0z);
83                 }
84             }
85         }
86
87 void CGrid::createFatorGeometricos(CReservoir* reservoir,
88                                     CDiscretization* discretization, CWell* well) {
89
90     double fatorGeometrico;
91
92     std::vector<double> partial = well->Partial();
93
94     for (int k = 0; k < nz; k++) {
95
96         for (int i = 0; i < nr; i++) {
97
98             // fator geometrico em i - 1/2
99
100            if (i == 0)
101
102                gimh.push_back(0.0);
103
104            else
105
106                gimh.push_back(discretization->Bc() * rmh[i]
107
108                    ] * kimh[i + k * nr] * reservoir->Theta()
109
110                    () * (zph[k] - zmh[k]) / (r[i] - r[i - 1]));
111
112
113            // fator geometrico em i - 1/2
114
115            if (i == nr-1)
116
117                giph.push_back(0.0);
118
119            else
120
121                giph.push_back(discretization->Bc() * rph[i]
122
123                    ] * kiph[i + k * nr] * reservoir->Theta()
124
125                    () * (zph[k] - zmh[k]) / (r[i+1] - r[i]));
126
127
128            // fator geometrico em j - 1/2
129
130            if (k == 0)
131
132                gjmh.push_back(0.0);
133
134            else
135
136                gjmh.push_back(discretization->Bc() * (pow(
137
138                    rph[i], 2) - pow(rmh[i], 2)) * kjmh[i +
139
140                    k * nr] * reservoir->Theta() / (2 * (z[k]
141
142                    ] - z[k - 1])));
143
144
145            // fator geometrico em j + 1/2
146
147            if (k == nz-1)
148
149                gjph.push_back(0.0);

```

```

113             else
114                 gjph.push_back(discretization->Bc() * (pow(
115                     rph[i], 2) - pow(rmh[i], 2)) * kjph[i +
116                     k * nr] * reservoir->Theta() / (2 * (z[k +
117                     +1] - z[k])));
118
119             /// fator geometrico do poco
120             if (i == 0)
121                 gw.push_back(partial[k] * discretization->
122                     Bc() * rmh[0] * kimh[k * nr] * reservoir
123                     ->Theta() * (zph[k] - zmh[k]) / (r[0] -
124                     rmh[0]));
125         }
126     }
127 }
128
129 void CGrid::createTime(CDiscretization* discretization, CWell* well
130 ) {
131     std::vector<double> tp = well->Tp();
132     double dtmin = discretization->DTmin();
133     time.push_back(tp[0]);
134     for (int i = 1; i < tp.size(); i++) {
135         for (int t = 0; t < discretization->Ntp(); t++) {
136             time.push_back(tp[i-1] + pow(10, t * (log10(
137                 (tp[i] - tp[i - 1]) - log10(dtmin)) / (
138                 discretization->Ntp() - 1.0))*dtmin);
139         }
140     }
141 }
```

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*
14             discretization, CWell* well);
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
24         bloco em relacao a x
25     std::vector<double> rph;            /// posicao em x + 1/2
26     std::vector<double> rmh;            /// posicao em x - 1/2
27
28     std::vector<double> z;              /// posicao do centro do
29         bloco em relacao a z
30     std::vector<double> zph;            /// posicao em z + 1/2
31     std::vector<double> zmh;            /// posicao em z - 1/2
32
33     std::vector<double> vb;             /// volume
34     std::vector<double> vb_ac;          /// volume corrigido
35
36     std::vector<double> kr;             /// permeabilidade
37         horizontal
38     std::vector<double> kz;              /// permeabilidade vertical
39
40     std::vector<double> kiph;            /// permeabilidade em i +
41         1/2
42     std::vector<double> kimh;            /// permeabilidade em i -
43         1/2
44     std::vector<double> kjph;            /// permeabilidade em j +
45         1/2
46     std::vector<double> kjmh;            /// permeabilidade em j -
47         1/2
48
49     std::vector<double> giph;            /// fator geometrico da
50         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,
53         CDiscretization* discretization);
54     void createFatorGeometricos(CReservoir* reservoir,
55         CDiscretization* discretization, CWell* well);
56     void createTime(CDiscretization* discretization, CWell*
57         well);
58
59 public:
60     /// funcoes get
61     std::vector<double> Time() { return time; }
62     double Time( int i) { return time[i]; }
63
64     double R( int i) { return r[i]; }
65     double Rph( int i) { return rph[i]; }
66     double Rmh( int i) { return rmh[i]; }
67
68     double Z( int i) { return z[i]; }
69     double Zph( int i) { return zph[i]; }
70     double Zmh( int i) { return zmh[i]; }
71
72     double Vb( int i) { return vb[i]; }
73     double Vb_ac( int i){ return vb_ac[i]; }
74
75     double Kr( int i) { return kr[i]; }
76     double Kz( int i) { return kz[i]; }
77
78     double Kiph( int i) { return kiph[i]; }
79     double Kimh( int i) { return kimh[i]; }
80     double Kjph( int i) { return kjph[i]; }
```

```

78     double Kjmh( int i)      { return kjmh[i]; }
79
80     double Giph( int i)      { return giph[i]; }
81     double Gimh( int i)      { return gimh[i]; }
82     double Gjph( int i)      { return gjph[i]; }
83     double Gjmh( int i)      { return gjmh[i]; }
84
85     double Gw( int i)        { return gw[i]; }
86
87     int Ntotal()             { return nr * nz; }
88     int Nr()                 { return nr; }
89     int Nz()                 { return nz; }
90     int Nt()                 { return time.size(); }
91
92     double Alpha()           { return alpha; }
93     double Omega()            { return omega; }
94     double DTheta()           { return dtheta; }
95 };
96 #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
4   , CWell* well) {
5     nr = discretization->Nr();
6     nz = discretization->Nz();
7     dtheta = reservoir->Theta();
8     createPosicoes(reservoir);
9     createVolumes(reservoir, discretization->Ac());
10    createPermeabilidades(reservoir, discretization);
11    createFatorGeometricos(reservoir, discretization, well);
12    createTime(discretization, well);
13
14 void CGrid::createPosicoes(CReservoir* reservoir) {
15   alpha = pow(reservoir->Re() / reservoir->Rw(), 1.0 / nr);
16   omega = log((alpha - 1) / log(alpha)) / log(alpha);
17
18   //----- raio -----

```

```

19         r.push_back(reservoir->Rw() * log(alpha) / (1-(1/alpha)));
20         rmh.push_back(reservoir->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->Re());
27
28         /// ----- profundidade -----
29         std::vector<double> dz = reservoir->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
37 void CGrid::createVolumes(CReservoir* reservoir, double Ac) {
38     double _vb;
39     std::vector<double> dz = reservoir->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],
43                 2)) * reservoir->Theta() * dz[k];
44             vb.push_back(_vb );
45             vb_ac.push_back(_vb * Ac);
46         }
47     }
48
49 void CGrid::createPermeabilidades(CReservoir* reservoir,
50     CDiscretization* discretization) {
51     int nrs = discretization->Nrs();
52
53     double k0r = reservoir->K0r();
54     double k0z = reservoir->K0z();
55
56     double krs = k0r / (reservoir->S() / log(rph[nrs] /
57         reservoir->Rw()) + 1.0);
58     double kzs = k0z / (reservoir->S() / log(rph[nrs] /
59         reservoir->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->InfVol() && k == nz-1) {
73                 kr[i + k * nr] = kr[i + k * nr] *
74                     1.0e4; // multiplicador
75                     arbitrario, valor deve ser
76                     grande
77                     kz[i + k * nr] = kz[i + k * nr] *
78                     1.0e4; // multiplicador
79                     arbitrario, valor deve ser
80                     grande
81             }
82         }
83     }
84 }
85 }
86
87 void CGrid::createFatorGeometricos(CReservoir* reservoir,
88                                     CDiscretization* discretization, CWell* well) {
89     double fatorGeometrico;
90     std::vector<double> partial = well->Partial();
91     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->Bc() * rmh[i]
97                               * kimh[i + k * nr] * reservoir->Theta
98                               () * (zph[k] - zmh[k]) / (r[i] - r[i -
99                               1]));
100
101            /// fator geometrico em i - 1/2
102            if (i == nr-1)
103                giph.push_back(0.0);
104            else
105                giph.push_back(discretization->Bc() * rph[i]
106                               * kiph[i + k * nr] * reservoir->Theta
107                               () * (zph[k] - zmh[k]) / (r[i+1] - r[i])
108                               );
109
110            /// fator geometrico em j - 1/2
111            if (k == 0)
112                gjmh.push_back(0.0);
113            else
114                gjmh.push_back(discretization->Bc() * (pow(
115                    rph[i], 2) - pow(rmh[i], 2)) * kjmh[i +
116                    k * nr] * reservoir->Theta() / (2 * (z[k]
117                    - z[k - 1])));
118
119            /// fator geometrico em j + 1/2
120            if (k == nz-1)
121                gjph.push_back(0.0);
122            else
123                gjph.push_back(discretization->Bc() * (pow(
124                    rph[i], 2) - pow(rmh[i], 2)) * kjph[i +
125                    k * nr] * reservoir->Theta() / (2 * (z[k]
126                    +1] - z[k])));
127
128            /// fator geometrico do poco
129            if (i == 0)
130                gw.push_back(partial[k] * discretization->
131                             Bc() * rmh[0] * kimh[k * nr] * reservoir

```

```

120         }
121     }
122 }
123
124 void CGrid::createTime(CDiscretization* discretization, CWell* well
125 ) {
126     std::vector<double> tp = well->Tp();
127     double dtmin = discretization->DTmin();
128     time.push_back(tp[0]);
129     for (int i = 1; i < tp.size(); i++) {
130         for (int t = 0; t < discretization->Ntp(); t++) {
131             time.push_back(tp[i-1] + pow(10, t * (log10
132                 (tp[i] - tp[i - 1]) - log10(dtmin)) / (
133                 discretization->Ntp() - 1.0)) * dtmin);
134         }
135     }
136 }
```

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"
12     -p"
13 #else
14 #define GNUPLOT_NAME "gnuplot"
15 #endif
16 class CGnuplot {
17 public:
18     CGnuplot() {}
19 }
```

```

20     static void plot(std::string name, std::string xlabel, std
21         ::string ylabel, std::string saveName);
22     static void semilogx(std::string name, std::string xlabel,
23         std::string ylabel, std::string saveName);
24     static void semilogy(std::string name, std::string xlabel,
25         std::string ylabel, std::string saveName);
26     static void surfacePlot(std::string name, std::string
27         saveName);
28 };
29 #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
3 using namespace std;
4
5 void CGnuplot::plot(string name, string xlabel, string ylabel,
6     string saveName) {
7 #ifdef _WIN32
8     FILE* pipe = _popen(GNU_PLOT_NAME, "w");
9 #else
10    FILE* pipe = popen(GNU_PLOT_NAME, "w");
11 #endif
12    fprintf(pipe, ("set xlabel '" + xlabel + "'\n").c_str());
13    fprintf(pipe, ("set ylabel '" + ylabel + "'\n").c_str());
14    fprintf(pipe, "unset key\n");
15    fprintf(pipe, ("plot '" + name + "' with linespoints"
16        " linestyle 1\n").c_str());
17    fprintf(pipe, "set term pngcairo\n");
18    fprintf(pipe, ("set output '" + saveName + "'\n").c_str());
19    fprintf(pipe, "replot\n");
20 //fprintf(pipe, "set term win\n");
21    fflush(pipe);
22 }
23
24 void CGnuplot::semilogy(string name, string xlabel, string ylabel,
25     string saveName) {
26 #ifdef _WIN32
27     FILE* pipe = _popen(GNU_PLOT_NAME, "w");
28 #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"
33                 " linestyle 1\n").c_str());
34     fprintf(pipe, "set term pngcairo\n");
35     fprintf(pipe, ("set output'" + saveName + "'\n").c_str());
36     fprintf(pipe, "replot\n");
37 //fprintf(pipe, "set term win\n");
38     fflush(pipe);
39 }
40 void CGnuplot::semilogx(string name, string xlabel, string ylabel,
41                         string saveName) {
42 #ifdef _WIN32
43     FILE* pipe = _popen(GNUPLOT_NAME, "w");
44 #else
45     FILE* pipe = popen(GNUPLOT_NAME, "w");
46 #endif
47     fprintf(pipe, ("set xlabel'" + xlabel + "'\n").c_str());
48     fprintf(pipe, ("set ylabel'" + ylabel + "'\n").c_str());
49     fprintf(pipe, ("set logscale x\n"));
50     fprintf(pipe, "unset key\n");
51     fprintf(pipe, ("plot'" + name + "' with linespoints"
52                 " linestyle 1\n").c_str());
53     fprintf(pipe, "set term pngcairo\n");
54     fprintf(pipe, ("set output'" + saveName + "'\n").c_str());
55     fprintf(pipe, "replot\n");
56 //fprintf(pipe, "set term win\n");
57     fflush(pipe);
58 }
59 void CGnuplot::surfacePlot(string name, string saveName) {
60 #ifdef _WIN32
61     FILE* pipe = _popen(GNUPLOT_NAME, "w");
62 #else
63     FILE* pipe = popen(GNUPLOT_NAME, "w");
64 #endif
```

```

65     fprintf(pipe, "set palette rgbformulae 33,13,10\n");
66     fprintf(pipe, "set dgrid3d 40,40,40\n");
67     fprintf(pipe, "set logscale x\n");
68     fprintf(pipe, "set view map\n");
69     fprintf(pipe, "unset key\n");
70     fprintf(pipe, ("splot " + name + "' with pm3d\n").c_str())
71         ;
72     fprintf(pipe, ("set output '" + saveName + "'\n").c_str());
73     fprintf(pipe, "replot\n");
74     //fprintf(pipe, "set term png\n");
75     std::cin.get();
76     fflush(pipe);
77 }

```

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(double _cf, double _p0, double _mu, double _T0, double
14           _T, double _Tpc, double _Ppc, double _Ma):
15         cf{ _cf }, p0{ _p0 }, mu{ _mu }, T0{ _T0 }, T{ _T
16         }, Tpc{ _Tpc }, Ppc{ _Ppc }, Ma{ _Ma }{}
17     CGas() {}
18
19 public:
20     const double DELTA = 1.0e-5;
21     double calc_b(double p);
22     double calc_rho(double p);
23     double calc_dbdp(double p);
24     double calc_mu(double p);
25     double calc_dmudp(double p);

```

```

24
25         std::string get_type() { return type; }
26
27 private:
28
29         double Z_KIAM(double p);
30         double mu_LGE(double rho);
31
32         std::string type = "gas";
33
34         double cf{ 0.00215094 };           /// compressibilidade do
35             fluido na condição inicial [cm^2/kgf]
36         double p0{ 1.0335123 };          /// pressão padrão [kgf/cm
37             ^2]
38         double mu{ 0.0262317 };          /// viscosidade na condição
39             inicial [cp]
40         double T0{ 288.75 };            /// temperatura absoluta
41             padrão [K]
42         double T{ 353.15 };              /// temperatura do
43             fluido no reservatório [K]
44         double Tpc{ 216.32 };            /// temperatura
45             pseudocritica [K]
46         double Ppc{ 46.34 };            /// pressão pseudocritica [
47             kgf/cm^2]
48         double Ma{ 20.3 };              /// massa molecular
49             aparente [kg/kg-mol]
50
51         const double R = 0.08478;        ///constante universal dos
52             gases (ANP) [(kgf/cm^2)*(m^3)/(kg-mol*K)]
53     };
54 #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) /
13     Z_KIAM(p * (1.0 + DELTA))) - (p / Z_KIAM(p)));
14 }
15
16 double CGas::calc_mu(double p) {
17     return mu_LGE(0.001 * calc_rho(p));
18 }
19
20 double CGas::calc_dmudp(double p) {
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))
      / ((D * ppr + E * pow(y, 2) - F * pow(y, G)) * pow
      (1 - y, 3));
66
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 classe CFluido.

Listing 6.21: Arquivo de implementação da classe CFluido.

```
#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
8             volumes na altura
9     int nr{ 50 };                               /// qtd de
10            volumes na largura
11     int nrs{ 1 };                               /// qtd de
12             volumes na regiao danificada
13     int nt{ 100 };                             /// qtd de
14             tempos
15     int ntp{ 100 };                            /// qtd de
16             tempos
17     int max_iter{ 24 };                         /// numero
18             maximo de iteracoes
19     double dtmin{ 1.0 / 3600 };                /// passo de tempo
20             minimo [h]

```

```

14     double eps_NR{ 1.0e-6 };                                /// tolerancia de
15     convergencia dos resíduos
16     double eps_MB{ 1.0e-8 };                                /// tolerancia de
17     convergencia do balanco de materiais
18     double ac{ 24 };                                         ///
19     constante de conversao de unidades acumulo (ANP)
20     double bc{ 0.0083621472 };                            /// constante de
21     conversao de unidades fluxo (ANP)

22
23     /// funções get
24     int Nz()          { return nz; }
25     int Nr()          { return nr; }
26     int Nrs()         { return nrs; }
27     int Nt()          { return nt; }
28     int Ntp()         { return ntp; }
29     int Max_iter()   { return max_iter; }
30     double Eps_NR() { return eps_NR; }
31     double Eps_MB() { return eps_MB; }
32     double DTmin()  { return dtmin; }
33     double Ac()      { return ac; }
34     double 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.

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

Capítulo 7

Teste

Todo projeto de engenharia passa por uma etapa de testes. Neste capítulo apresentamos alguns testes do software desenvolvido. Na sua validação foi utilizado o artigo [Bilhartz and Ramey, 1977]. Para efeitos de simplificação, este artigo mostra como se dá o efeito da penetração parcial em um poço. A seguir, algumas definições importantes definidas pelos autores:

- 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: $P_D = (p_i - p) \frac{b k_r h}{C_2 q \mu}$.

Transformando os resultados do simulador para o tempo e pressão adimensional apresentados acima, foi 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, foram utilizadas as seguintes propriedades:

- $h = 10m$;
- $r_w = 0.0447m$;
- $h_w = 5m$;
- $k_r = 500md$;
- $k_z = 100md$;
- $\phi = 0.2$;

- $q = 1000 \text{m}^3/\text{dia};$

- $p_i = 350 \text{kgf/cm}^2;$

Na Figura 7.1 abaixo tem-se o comportamento da pressão ao longo do reservatório para o Teste #1.

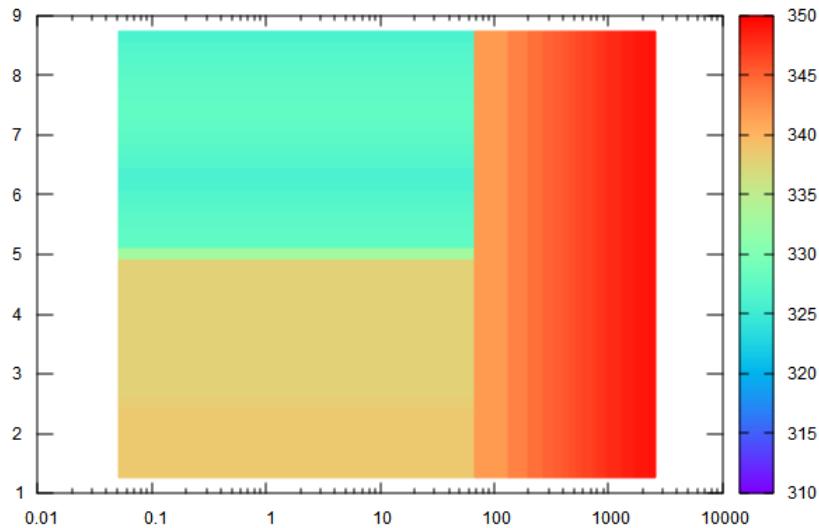


Figura 7.1: Teste 1: pressão ao longo do reservatório. A pressão é menor na região destacada em verde, visto que está aberta à produção

Na Figura 7.2 foi feito um match entre simulador e artigo.

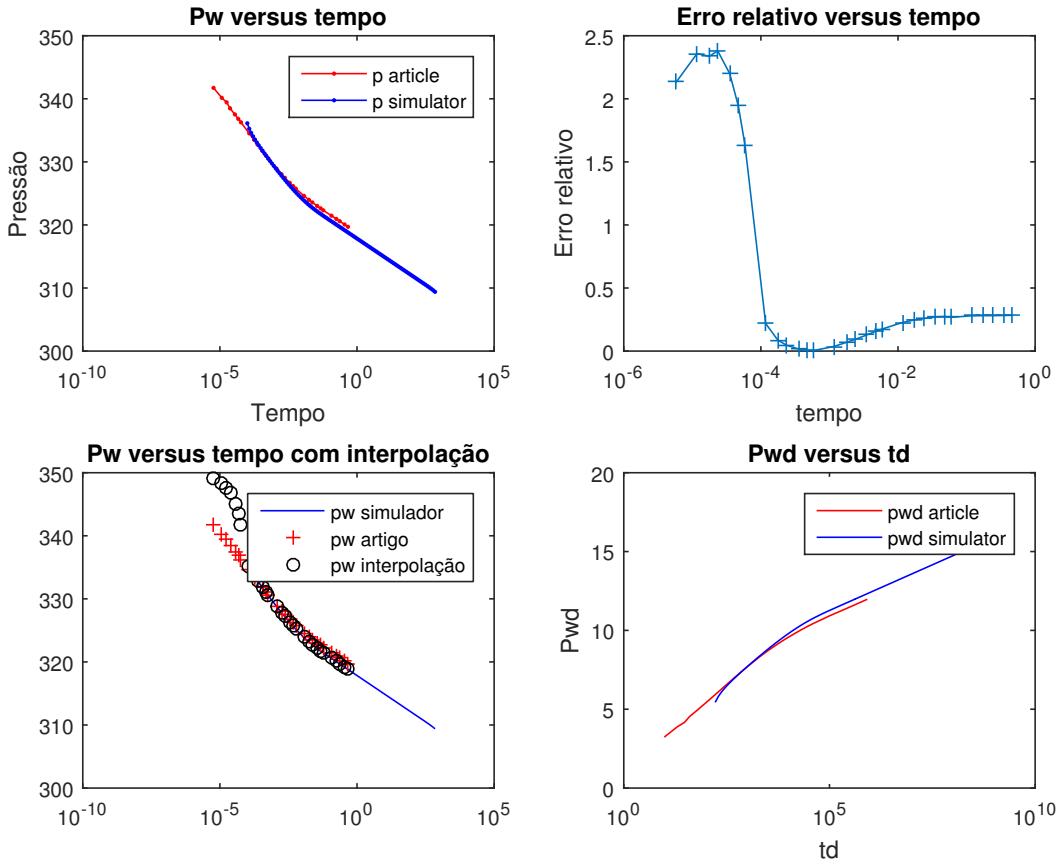


Figura 7.2: No canto superior esquerdo, é apresentada a comparação da pressão do poço do simulador com o artigo. À direta, o erro ao longo do tempo. Por sua vez, no canto inferior esquerdo, é apresentado a comparação com interpolação do artigo, ao passo que na direita, a comparação com valores adimensionais

Como pode ser observado, ambos os dados encontram-se bem próximos. Desconsiderando os primeiros pontos de resultado do simulador, o erro ficou próximo à 0,4%, o que é excelente.

7.2 Teste 2

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

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

Na Figura 7.3 abaixo tem-se o comportamento da pressão ao longo do reservatório para o Teste #2.

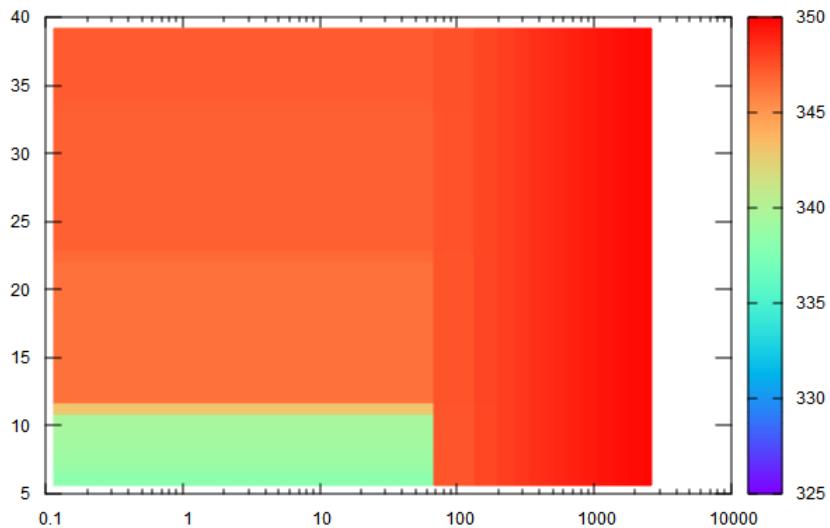


Figura 7.3: Teste 2: pressão ao longo do reservatório. A pressão é menor (região verde), por ser a região aberta à produção

Na Figura 7.2 foi feito um match entre simulador e artigo, tal como feito na Figura anterior.

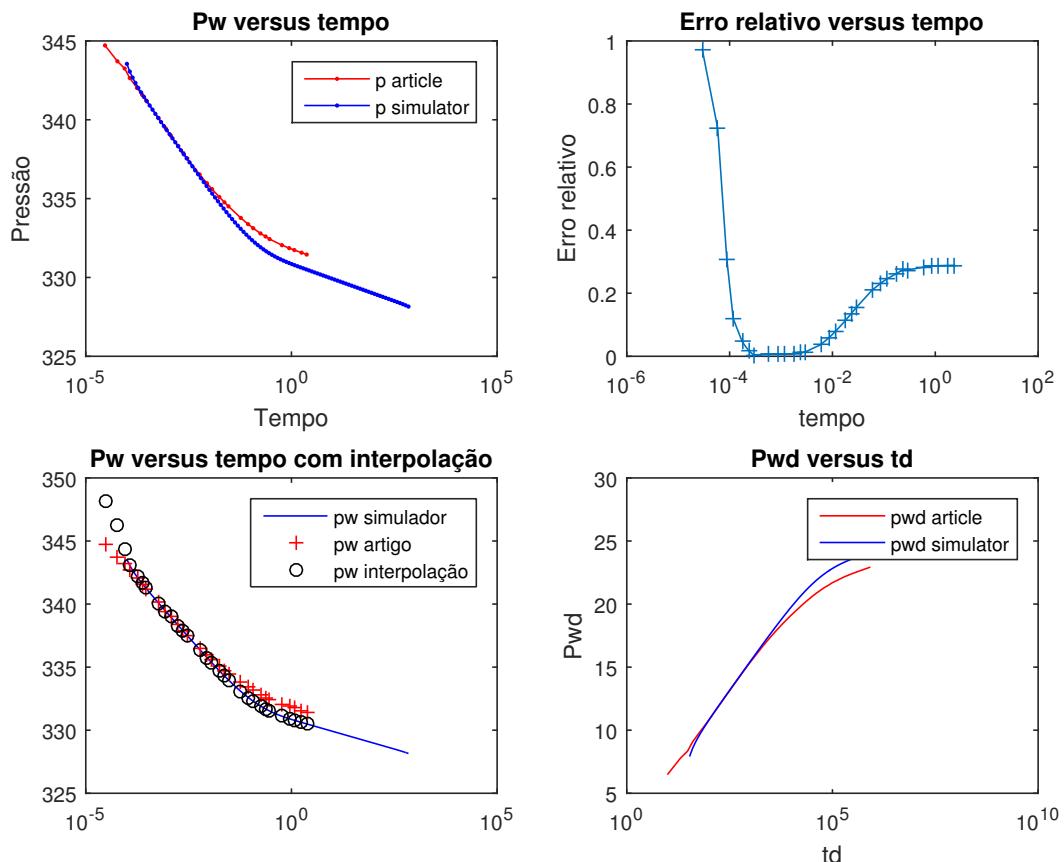


Figura 7.4: No canto superior esquerdo, é apresentada a comparação da pressão do poço do simulador com o artigo. À direita, o erro ao longo do tempo. Por sua vez, no canto inferior esquerdo, é apresentada a comparação com interpolação do artigo, e na direita, a comparação com valores adimensionais

Desconsiderando os primeiros pontos de resultado, 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 mais 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 sobre o uso do "Simulador Monofásico 2D". Esta documentação trás um passo a passo de como usar o software.

8.1 Documentação do usuário

Descreve-se aqui o manual do usuário, um guia que para instalação e uso do software Simulador 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 abrí-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 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 `sudo 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 mudança [ $m^3 std/dia$ ]
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   /// compressibilidade 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)

```

Líquido

```

cf: 14.7e-5    /// compressibilidade do fluido[cm2/kgf]
b0: 1.0        /// inverso do fator volume formação na pressão p0 [m3std/m3]
p0: 1.0335123  /// pressao de referência [kgf/cm2]
mu: 1.0        /// viscosidade [cp]
cmu: 0.0

```

8.2.2 Como gerar a documentação usando doxygen

A documentação do código 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 e um relatório no formato rtf (*Rich Text Format*) com todos os arquivos apresentados 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.

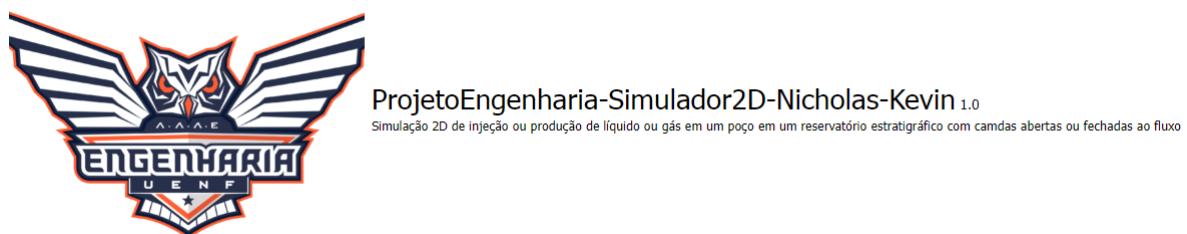


Figura 8.1: Cabeçalho do projeto - doxygen

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

C CDiscretization
C CFluido
C CGas
C CGnuplot
C CGrid
C CLiquido
C CMatrix
C CProps
C CReservoir
C CSimuladorMonofasico2D
C CWell

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.

```

ProgramacaoPratica2021
└── ProjetoEngenharia-Simulador2D-Nicholas-Kevin
    └── listagens
        ├── CDiscretization.cpp
        ├── CDiscretization.hpp
        ├── CFluido.cpp
        ├── CFluido.hpp
        ├── CGas.cpp
        ├── CGas.hpp
        ├── CGnuplot.cpp
        ├── CGnuplot.hpp
        ├── CGrid.cpp
        ├── CGrid.hpp
        ├── CLiquido.cpp
        ├── CLiquido.hpp
        ├── CProps.cpp
        ├── CProps.hpp
        ├── CReservoir.cpp
        ├── CReservoir.hpp
        ├── CSimulador.cpp
        ├── CSimulador.hpp
        ├── CWell.cpp
        ├── CWell.hpp
        └── main.cpp

```

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 CSimuladorMonofasico2D.hpp.

```
1 #ifndef CSimuladorMonofasico2D_HPP
2 #define CSimuladorMonofasico2D_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 "CLiquid.hpp"
19 #include "CReservoir.hpp"
20 #include "CDiscretization.hpp"
21
22 class CSimuladorMonofasico2D {
23 public:
24     CSimuladorMonofasico2D();
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(CProps* props_n, CProps* props_nu, double dt);
40     std::vector<double> calc_Q(double time);
41     std::vector<double> calc_X(double pw, std::vector<double> p);
42     std::vector<std::vector<double>> calc_T(CProps* props_n);
43
44     std::vector<std::vector<double>> calc_eta(CProps* props_nu, double dt);
45     std::vector<std::vector<double>> calc_tau(CProps* props_nu, std::vector<double> p_nu, double pw_nu);
46
47     void plot(std::vector<double> time, std::vector<double> Pw);
48     void plotSurface();
49
50     bool isErrorNotAcceptable(double dt, std::vector<double> R, CProps* props_nu, double q);
51
52     void read_data_and_start_objects(std::string nameFile);
53 };
54 #endif
```

Figura 8.4: Código fonte CSimuladorMonofasico2D - 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*. 80
- [K. Aziz, 1979] K. Aziz, A. S. (1979). *Petroleum Reservoir Simulation*. Applied Science Publishers, 1 edition. 13, 15
- [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. 13
- [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. 13
- [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. 13
- [P., 2014] P., D. L. (2014). *Engenharia de reservatórios : fundamentos*. Elsevier, Rio de Janeiro. 9
- [Pico, 2018] Pico, C. E. (2018). *Simulação numérica de reservatórios*. Notas de aula LNEP/CCT/UENF, 1. 1, 10, 13
- [ROSA, 2006] ROSA, Adalberto Jose; CARVALHO, R. d. S. X. J. A. D. (2006). Engenharia de Reservatórios de Petróleo. *Interciência*, Rio de Janeiro. 1, 9
- [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*. 15

Capítulo 9

Como modificar o arquivo inputdata

Neste anexo, é apresentado como o usuário deve modificar o arquivo com as propriedades iniciais da simulação.

9.1 Modificando o tipo de fluido

Abaixo é mostrado o arquivo com inputs para o caso do líquido, com 5 camadas de 1 metro cada, e somente a primeira aberta à produção. Para escolher a opção de líquido, é importante modificar a variável do Reservatório “isLiquid” para 1. O líquido recebe 5 propriedades, conforme mostrado abaixo.

```
----- 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 mudança [ $m^3 std/dia$ ]
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 // angulo estudado do reservatorio
k0r: 500 // permeabilidade horizontal
k0z: 100 // permeabilidade vertical
cphi: 1.0e-4 // compressibilidade da formacao
phi0: 0.2 // porosidade inicial
p0: 1.033512 // pressao de referencia
p_i: 350.0 // pressao inicial
S: 0 // fator de pelicula
Temperature: 353.15 // temperatura do reservatorio

```

Discretização

```

nr: 20 // quantidade de volumes na largura
nrs: 1 // quantidade de volumes na regiao danificada
nt: 100 // quantidade de tempos
ntp: 100 // quantidade de tempos
max_iter: 24 // numero maximo de iteracoes
dtmin: 1.0e-4 // passo de tempo minimo[h]
eps_NR: 1.0e-6 // tolerancia de convergencia dos resíduos
eps_MB: 1.0e-8 // tolerancia de convergencia do balanco de materiais
Ac: 24 // constante de conversao de unidades acúmulo (ANP)
Bc: 0.0083621472 // constante de conversao de unidades fluxo (ANP)

```

Líquido

```

cf: 14.7e-5 // compressibilidade do fluido[cm2/kgf]
b0: 1.0 // inverso do fator volume formacao na pressao p0 [m3std/m3]
p0: 1.0335123 // pressao de referencia [kgf/cm2]
mu: 1.0 // viscosidade [cp]
cmu: 0.0

```

Abaixo é mostrado o caso de gás, que recebe 7 propriedades.

----- Simulador Monofásico 2D -----

----- Nicholas e Kevin -----

Poço periodos: 2

```

tp: 0 1000 // tempos de mudança na vazao na superficie [h]
qsc: 0 -500 // vazoes nos tempos de mudanca[m3std/dia]
nz: 5 // numero de camadas do reservatorio
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: 0.523598666666    /// ângulo estudado do reservatorio
k0r: 500    /// permeabilidade horizontal
k0z: 100    /// permeabilidade vertical
cphi: 1.0e-4    /// compressibilidade 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)

```

Gas

```

cf: 0.00215094    /// compressibilidade do fluido[cm2/kgf]
p0: 1.08335123    /// pressão padrão [kgf/cm2]
mu: 0.0262317    /// viscosidade na condição inicial [cp]
T0: 288.75    /// temperatura absoluta padrão [K]
Tpc: 216.32    /// temperatura pseudocrítica [K]
Ppc: 46.34    /// pressão pseudocrítica [kgf/cm2]

```

Ma: 20.3 /// massa molecular aparente [kg/kg-mol]
--

9.2 Modificando as camadas do reservatório

Para modificar as camadas do reservatório, é necessário alterar os valores de “nz” e “dz | aberto/fechado”, onde “nz” é o número de camadas, e “dz | aberto/fechado” são valores adicionados nas linhas posteriores, com a espessura e se está aberto ou fechado.

Por exemplo, um reservatório com 2 camadas, a primeira fechada com 1 metro, e a segunda aberta com 3 metros, fica:

nz: 2 /// número de camadas do reservatório dz aberto/fechado /// indicação da camada / aberto = 1; fechado = 0 1 0 3 1
--

O simulador recebe o valor de “aberto/fechado” e aceita valores entre 0 e 1, onde 0 é totalmente fechado, e 1 totalmente aberto à produção.

Índice Remissivo

A

- Análise orientada a objeto, 19
- AOO, 19
- Associações, 29
- atributos, 28

C

- Casos de uso, 5
- colaboração, 23
- comunicação, 23
- Controle, 27

D

- Diagrama de colaboração, 23
- Diagrama de componentes, 29
- Diagrama de execução, 30
- Diagrama de máquina de estado, 24
- Diagrama de sequência, 21

E

- Efeitos do projeto nas associações, 29
- Efeitos do projeto nas heranças, 29
- Efeitos do projeto nos métodos, 28
- Elaboração, 8
- estado, 24
- Eventos, 21

H

- Heranças, 29
- heranças, 29

I

- Implementação, 32

M

- Mensagens, 21

métodos, 28

modelo, 28

O

otimizações, 29

P

- Plataformas, 27
- POO, 27
- Projeto do sistema, 26
- Projeto orientado a objeto, 27
- Protocolos, 26

R

Recursos, 26