POLI/UFRJ

MODELO NUMÉRICO PARA ANÁLISE DA INFLUÊNCIA DE INTRUSÕES MAGMÁTICAS NA ESTRUTURA TÉRMICA E NA MATURAÇÃO DA MATÉRIA ORGÂNICA DE BACIAS SEDIMENTARES

Daniel Francisco Maia Vasconcelos

Projeto de Graduação apresentado ao corpo docente do curso de Engenharia de Petróleo da escola politécnica, POLI, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Bacharel em Engenharia de Petróleo.

Orientadores: Luiz Landau
Jaci Maria Bernardo da Silva Guigon

Rio de Janeiro
Dezembro de 2010

MODELO NUMÉRICO PARA ANÁLISE DA INFLUÊNCIA DE INTRUSÕES MAGMÁTICAS NA ESTRUTURA TÉRMICA E NA MATURAÇÃO DA MATÉRIA ORGÂNICA DE BACIAS SEDIMENTARES

Daniel Francisco Maia Vasconcelos

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE PETRÓLEO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIAS EM ENGENHARIA DE PETRÓLEO.

Examinada por:

Prof. Luiz Landau, D.Sc.

Prof. Jaci Maria Bernardo da Silva Guigon, D.Sc.

Prof. Paulo Couto, D.Sc.

Geól. Luís Maurício Salgado Alves Corrêa, M.Sc.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2010

Maia Vasconcelos, Daniel Francisco

Modelo Numérico para Análise da Influência de Intrusões Magmáticas na Estrutura Térmica e na Maturação da Matéria Orgânica de Bacias Sedimentares/Daniel F. M. Vasconcelos, – Rio de Janeiro: UFRJ/ESCOLA POLITÉCNICA, 2010.

XVI, 112 p.: il.; 29, 7cm.

Orientadores: Luiz Landau

Jaci Maria Bernardo da Silva Guigon

Projeto de Graduação (bacharel) – UFRJ/ESCOLA POLITÉCNICA/Engenharia de Petróleo, 2010.

Referências Bibliográficas: p. 78 – 80.

1. Sistemas Petrolíferos. 2. Rochas ígneas. 3. Óleo e Gás. 4. Palavra Chave. I. Landau, Luiz *et al.* II. Universidade Federal do Rio de Janeiro, POLI, Engenharia de Petróleo. III. Título.

A experiência do assombro, de maravilhar-se diante dos fenômenos circundantes e diante de nosso próprio ser, supõe um dos maiores estímulos de nossa vida, ao mesmo tempo em que nos proporciona um contínuo prazer. Para Aristóteles, este é o fim último de nossa vida, o mais capaz de satisfazer nossas expectativas.
*(*Adela Cortina*)*

Agradecimentos

Gostaria de agradecer à vida, pois esta sim é o fenômeno natural mais interessante e desafiador, aos meus pais por sempre terem me dado todo o suporte que necessitei, à minha avó e irmão e a todas as pessoas que de um modo ou outro ajudaram, seja efetivamente neste trabalho, ou dando um pouco de sua amizade e confiança.

Resumo da Projeto de Graduação apresentado à POLI/UFRJ como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências (B.Sc.)

MODELO NUMÉRICO PARA ANÁLISE DA INFLUÊNCIA DE INTRUSÕES MAGMÁTICAS NA ESTRUTURA TÉRMICA E NA MATURAÇÃO DA MATÉRIA ORGÂNICA DE BACIAS SEDIMENTARES

Daniel Francisco Maia Vasconcelos

Dezembro/2010

Orientadores: Luiz Landau
Jaci Maria Bernardo da Silva Guigon

Departamento: Engenharia de Petróleo

Sistemas petrolíferos ígneo-sedimentares são sistemas mistos nos quais um ou mais elementos essenciais ou processos envolvidos estão relacionados a eventos magmáticos. Dentre as várias peculiaridades desse tipo de sistema, diques e soleiras de diabásio e derrames de basalto podem atuar como rochas-reservatório e selantes, na ausência de elementos convencionais, ou formar trapas estruturais ou combinadas para o aprisionamento de petróleo. Entretanto, uma das características mais marcantes é a de poder também fornecer calor extra para a geração e expulsão de hidrocarbonetos em bacias rasas e frias, tal e qual a Bacia do Solimões, que é um exemplo típico da relação magmatismo-petróleo. Assim, um estudo mais aprofundado acerca da influência na estrutura térmica da bacia e na maturação da matéria orgânica devido a presença de intrusões magmáticas é relevante.

Abstract of Graduation's Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Bachelor of Science (B.Sc.)

NUMERICAL MODEL TO THE ANALYSIS OF THE INFLUENCE OF MAGMATIC INTRUSIONS AT THE THERMAL STRUCTURE AND AT THE MATURATION OF THE ORGANIC MATTER OF SEDIMENTARY BASINS

Daniel Francisco Maia Vasconcelos

December/2010

Advisors: Luiz Landau
Jaci Maria Bernardo da Silva Guigon

Department: Petroleum Engineering

Igneous-sedimentary petroleum systems are mixed systems in which one or more essential elements or processes involved are related to magmatic events. Among the many peculiarities of this type of system, dikes and sills of diabase and basalt flows can act as reservoir rocks and seals, in the absence of conventional elements, or form structural or combined traps for the trapping of the oil. However, the most striking feature is the ability to also provide extra heat to the transformation of organic matter in shallow basins and cold ones, just like the Solimões Basin, brazilian example of the relationship between magmatism and petroleum. Thus, further study concerning the effects on the thermal structure of the basin and on the maturation of organic matter due to the presence of magmatic intrusions is necessary.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

A	fator de frequência, p. 40
$A_n u_i$	operador laplaciano n aplicado em u, p. 18
A_{nh}	operador laplaciano discreto n, p. 19
C_p	calor específico da rocha, p. 12
C_{p_p}	calor específico do fluido no poro, p. 12
E	energia de ativação, p. 40
H_j	taxa de aquecimento, p. 43
L_x	comprimento do domínio na direção x, p. 14
L_y	comprimento do domínio na direção y, p. 14
M	número de subintervalos de tempo, p. 25
Q_r	densidade de energia produzida devido radiação, p. 12
R	constante universal dos gases, p. 40
T_N	temperatura prescrita no bordo Norte, p. 14
T_S	temperatura prescrita no bordo Sul, p. 14
Δt	passo de tempo, p. 18
Γ	contorno de um domínio, p. 18
Ω	domínio de definição de uma equação diferencial, p. 18
β	fator de expansão ou contração do passo de tempo, p. 25
δ^2	operador diferencial de segunda ordem, p. 22
ϵ	perturbação numérica, p. 29

η	dimensão espacial adimensional de cada partição, p. 22
γ	razão de estabilidade, p. 30
κ	difusividade térmica, p. 18
λ	média, p. 47
$\mathbb{R}$	conjunto dos números reais, p. 18
∇	operador gradiente, p. 12, 18
$\overline{k}$	tensor de condutividades térmicas da rocha, p. 12
∂	operador do contorno, p. 12, 18
ρ	densidade da rocha, p. 12, 18
ρ_p	densidade do fluido no poro, p. 12
σ	região limitada do espaço, p. 33
τ	difusividade térmica adimensional, p. 22
θ	temperatura adimensional, p. 22
ξ	passo de tempo decorrente da subdivisão de um passo de tempo cheio, p. 25
f	anisotropia, p. 28
f_i	fator estequiométrico, p. 43
g_i	fator de escala, p. 22
h_x	dimensões espacial de cada partição na direção x, p. 19
h_y	dimensões espacial de cada partição na direção y, p. 19
i	$\sqrt{-1}$, p. 30
k	taxa de reação, p. 40
k_h	condutividade térmica horizontal, p. 28
k_i	condutividade térmica calculada na interface i, p. 26
k_v	condutividade térmica vertical, p. 28
p	ordem de convergência, p. 33

p_i	fração de massa, p. 47
q_E	fluxo prescrito no bordo Leste, p. 14
q_W	fluxo prescrito no bordo Oeste, p. 14
r_i	número de fourier, p. 22
v_p	velocidade do fluido no poro, p. 12
w	quantidade não reagida de um componente, p. 43
w_0	concentração inicial total de reagente, p. 43
$w_{0,i}$	concentração inicial do $\mathrm{i}_{\text{ésimo}}$ componente, p. 43

Lista de Abreviaturas

ADI	alternating direction implicit, p. 3
EDPs	equações diferenciais parciais, p. 17
MDF	método das diferenças finitas, p. 3
TDMA	tridiagonal matrix algorithm, p. 17

Capítulo 1

Introdução

À medida em que a taxa de descoberta de novas acumulações de petróleo entra em declínio mais acentuado, começa-se a buscar novas fronteiras exploratórias para que a demanda mundial de petróleo seja atendida. Regiões que outrora não eram consideradas viáveis tanto economicamente quanto tecnicamente, agora recebem uma atenção redobrada. Dentre essas fronteiras pode-se destacar os sistemas petrolíferos não convencionais, como é o caso dos ígneo-sedimentares.

Tal situação outrora de abandono, ou melhor dizendo, de pouca atividade exploratória, deve-se em parte, à grande facilidade de explotar as acumulações de petróleo em sistemas petrolíferos convencionais, tais como reservatórios areníticos localizados *onshore* e *offshore*, com trapas estratigráficas bem definidas e síncronas, dentre outras situações usuais. Até então, a presença de rochas ígneas no interior de bacias sedimentares era tida como um empecilho à ocorrência de petróleo e à pesquisa petrolífera. Era de comum acordo entre os especialistas, que intrusões e extrusões de material magmático nas bacias sedimentares "destruíam" a matéria orgânica e o petróleo gerado anteriormente, além de reduzir a porosidade e até mesmo tamponar a garganta de poros das rochas reservatório. Alegava-se também que o tectonismo associado aos eventos magmáticos abriria as trapas, causando remobilização e a consequente exudação do petróleo existente. Todavia, com o aumento das pesquisas na área e o avanço em direção às novas fronteiras exploratórias, esse quadro mudou e o estudo dos sistemas petrolíferos ígneos-sedimentares vem mostrando que novas descobertas poderão ocorrer em tais regiões.

Um "sistemas petrolífero" é um sistema geológico que engloba a rocha geradora de hidrocarboneto e e a rocha reservatório, além dos demais elementos e processos geológicos que são essenciais à existência de um acumulação de hidrocarbonetos [6]. Sistemas petrolíferos ígneo-sedimentares são sistemas mistos, nos quais um ou mais elementos essenciais ou processos envolvidos estão relacionados a eventos

magmáticos. Dentre as várias peculiaridades desse tipo de sistema, diques e soleiras de diabásio e derrames de basalto podem atuar como rochas-reservatório e selantes, na ausência de elementos convencionais, ou formar trapas estruturais ou combinadas, para o aprisionamento de petróleo. Entretanto, uma das características mais marcantes é a de poder também fornecer calor extra para a transformação da matéria orgânica em bacias rasas e frias, tal e qual a Bacia do Solimões, exemplo brasileiro da relação magmatismo-petróleo.

Conforme mencionado anteriormente, o calor emanado de derrames de basalto ou de soleiras de diabásio pode compensar o baixo fluxo térmico em bacias rasas e frias, deste modo proporcionando a energia necessária para que haja o craqueamento do querogênio. MAGOON e DOW [6] classificam sistemas petrolíferos com essas características como "atípicos". A análise dos efeitos térmicos decorrentes da presença de instrusões magmáticas na maturação da matéria orgânica em sistemas petrolíferos ígneo-sedimentares passa então a ser fundamental para a estimativa de potencial petrolífero de bacias com tais características.

Assim, a proposta deste trabalho é que de posse de um modelo matemático que descreva a evolução térmica da bacia e conhecendo-se a história de soterramento da mesma, seja possível reconstruir sua história térmica. Essa informação é suficiente para o cálculo da maturação da matéria orgânica uma vez que segundo HANTSCHEL e KAUERAUF [5] ela é essencialmente função do tempo, tipo de matéria orgânica e da história térmica da bacia. Modelos baseados na equação de primeira ordem de Arrhenius são amplamente utilizados no cálculo da maturação da matéria orgânica.

Portanto, feita a validação do modelo numérico implementado, realizar-se-ão os cálculos da estrutura térmica da bacia devido a presença de intrusões magmáticas. De posse desses dados, calcula-se, então, o índice de reflectância da vitrinita para toda uma seção da bacia por uso do modelo Easy%Ro [2]. Assim, uma vez realizados os cálculos descritos, obtem-se as janelas de óleo e gás para a bacia.

O ideal, contudo, seria o de realizar a calibração do modelo desenvolvido por comparação com dados de vitrinita obtidos *in situ* (dados de poço(s) interceptando a seção) assim, validando o modelo geológico proposto para a bacia e o modelo matemático desenvolvido. Entretanto, diante a dificuldade de obtenção destes, fica como sugestão para trabalhos futuros a realização deste estudo comparativo.

1.1 Motivação

A motivação deste presente trabalho é justamente a de realizar a modelagem térmica e o estudo numérico desse fenômeno aplicado à uma seção bidimensional da Bacia

do Solimões, de forma a avaliar o potencial gerador de óleo através das janelas de óleo e gás, o grau de maturação do querogênio por meio do cálculo do índice de reflectância da vitrinita e, deste modo, como resultante, realizando em sua completude, a análise da maturação da matéria orgânica. Por fim, esse estudo procura confirmar de maneira embasada cientificamente, o potencial petrolífero de sistemas petrolíferos ígneo-sedimentares, até o momento pouco estudados.

1.2 Objetivos

O objetivo deste presente trabalho é o de se aplicar o Método das Diferenças Finitas (MDF), utilizando o esquema *Alternating Direction Implicit Method(ADI)* ou, na literatura lusófona, método das direções alternadas, para a modelagem numérica de uma seção real da Bacia do Solimões, esta contendo intrusões magmáticas; e, deste modo, determinando a sua estrutura térmica devido:

1. aos gradientes geotérmicos da bacia;
2. às temperaturas anômalas correspondentes à presença dos corpos ígneos.

Utilizando-se as temperaturas obtidas nos itens 1 e 2, propõe-se a utilização do modelo Easy%Ro para a determinação teórica dos índices de reflectância da vitrinita. Tais índices permitem traçar os limites das janelas de óleo e gás na seção, que por sua vez poderiam ser calibrados com valores reais de vitrinita, obtidos a partir de poços. Com base nesses, pode-se preparar as janelas de óleo e gás para a Bacia do Solimões.

1.3 Metodologia

Este presente texto estará estruturado metodologicamente da seguinte maneira:

- introdução;
- estudo geológico da região de interesse;
- obtenção da seção geológica 2D;
- discretização e implementação de modelo numérico semi-implícito (ADI);
- determinação dos índices de maturação do querogênio com base na estrutura térmica obtida mediante implementação do modelo Easy%Ro;
- determinação das janelas de óleo e gás para a bacia;

- elaboração de gráficos;
- análise de resultados;
- comentários finais.

Em termos de pseudocódigo, tem-se a seguinte estruturação metodológica a ser implementada computacionalmente:

- pré-processamento;
- inicialização das propriedades físicas;
- resolver as temperaturas;
- calcular Easy%Ro;
- pós-processamento.

A linguagem utilizada na implementação numérica foi *C++* devido a sua flexibilidade e eficiência, além de ser amplamente utilizada na computação científica de propósito geral. Para o pós-processamento e análise dos dados gerados foi realizada a implementação de um *script* em *Matlab*. As referências utilizadas para consulta foram: [7], [8] e [9].

1.4 Organização do trabalho

Neste trabalho apresenta-se inicialmente no Capítulo 2, um revisão bibliográfica acerca dos estudos e pesquisas realizadas em torno dos sistemas petrolíferos ígneo-sedimentares de maneira geral. A seguir, no Capítulo 3, é apresentado o caso estudado e o estado da arte no que diz respeito à sua contextualização geológica e sua importância no cenário nacional do petróleo e gás.

Definida a região do problema de estudo, parte-se no Capítulo 4 para a modelagem matemática do fenômeno físico de interesse, no caso, o problema de condução de calor e, deste modo, possibilitando no Capítulo 5 o desenvolvimento numérico do problema, com seus detalhamentos e desdobramentos subsequentes.

No Capítulo posterior (7) é realizada a aplicação do método numérico desenvolvido na bacia sedimentar estudada, culminando no Capítulo 8 com a análise dos resultados. Por fim, no Capítulo 9, é feita uma súmula conclusiva acerca dos trabalhos desenvolvidos.

Capítulo 2

Estado da Arte

A Petrobras desenvolve internamente projetos sobre as bacias do Solimões e Amazonas, envolvendo modelagem termomecânica, sistemas petrolíferos, diagênese de reservatórios e imageamento das rochas magmáticas. Em paralelo, instituições de pesquisa, representadas por algumas universidades, vêm também desenvolvendo pesquisas que possam contribuir para o melhor entendimento e fornecer subsídios para a predição do potencial de óleo e gás em bacias sedimentares e sistemas petrolíferos atípicos.

No entanto, de acordo com BARATA e CAPUTO [10], são poucos os estudos de domínio público que analisam de forma integrada os sistemas petrolíferos nas grandes bacias paleozóicas brasileiras, citando como exemplos para a Bacia de Solimões, apenas os trabalhos de MELLO *et al.* [11] e EIRAS [12, 13], os quais mostram análises detalhadas dos sistemas petrolíferos identificados nesta bacia.

A modelagem computacional dos processos térmicos podem fornecer informações importantes para tal análise integrada, identificando a história termal dentro do processo evolutivo da bacia e os valores percentuais relacionados com a maturação da matéria orgânica, possibilizando caracterizar as janelas de óleo e gás no interior das bacias sedimentares.

De acordo com a pesquisa bibliográfica, foram encontrados poucos trabalhos na literatura relacionados com a quantificação da influência das intrusões ígneas na evolução térmica de sistemas petrolíferos não convencionais, e particularmente, da Bacia do Solimões. Além disto, a maior parte dos estudos, mesmo empregando técnicas numéricas de aproximação, apresenta soluções para problema unidimensional, baseados em dados de poços (LUTHI e O'BRIEN [14]; CORRÊA [15]), ou desconsidera a história de temperaturas da bacia (BAUDINO *et al.* [16]).

Um estudo mais recente (FJELDSKAAR *et al.* [17]) apresenta resultados baseados em um modelo numérico 2D aplicado a dados reais de uma região de configuração estrutural complexa, localizada no Mar do Norte (Voring Basin), considerando também, a história térmica da bacia, obtida a partir de modelos de descom-

pactação.

No presente trabalho, o modelo numérico proposto está sendo validado e aplicado a uma malha 2D, representativa de uma seção geológica da bacia do Solimões. Posteriormente, é empregado o modelo Easy%Ro [2], utilizado para calcular os valores da reflectância da vitrinita, a partir da história térmica resultante do gradiente geotérmico da bacia, somado aos valores anômalos de temperatura obtidos numericamente, a partir das rochas intrusivas presentes em uma seção bidimensional.

Capítulo 3

Caracterização da Região de Estudo

A Bacia do Solimões, anteriormente denominada Bacia do Alto Amazonas (CAPUTO [18]), situa-se no Estado do Amazonas, sendo classificada como uma bacia paleozóica intracratônica, possuindo espessura sedimentar máxima em torno de 3.800 m. É delimitada ao norte pelo Escudo das Guianas, ao sul pelo Escudo Brasileiro, a leste pelo Arco de Purus e a oeste pelo Arco de Iquitos. Está subdividida em duas sub-bacias (Juruá, à leste e Jandiatuba, à oeste), separadas pelo Alto de Carauari, possuindo cerca de 950.000 km^2 de área sedimentar total, com praticamente metade desta área correspondente à região prospectável para gás, óleo e condensado.

A primeira pesquisa para estimar o potencial petrolífero da Bacia do Solimões foi realizada em 1976, através do levantamento de sísmica de reflexão na região e já em 1978, descobria-se a província de gás e condensado do Juruá, o que intensificou a pesquisa de petróleo na região. Em 1986, descobria-se a província de óleo, gás e condensado de Urucu, localizada a 600 km de Manaus. Em 1998 teve início a operação do poliduto, com 285 km de extensão, entre Urucu e Coari, cidade mais próxima da base petrolífera. [19]

Atualmente, a Bacia de Solimões é a terceira bacia em produção de óleo no Brasil, sendo o petróleo de Urucu considerado o de melhor qualidade no país, além de possuir grandes reservas de gás natural ([19]).

Por se tratar de uma bacia paleozóica e de pouca espessura sedimentar, possui gradiente geotérmico reduzido, o que poderia comprometer a maturação da matéria orgânica, se não fossem as intrusões ígneas ali contidas. Assim, de acordo com BENDER *et al.* [20], na Bacia do Solimões, a temperatura, o volume e a geometria das várias intrusões ígneas (representadas por diques e soleiras de diabásio), além da sobrecarga causada por elas, tiveram influência primordial na evolução térmica da matéria orgânica e na geração do petróleo. Além disso, segundo FILHO, WANDERLEY *et al.* [21], as soleiras de diabásio foram importantes na geração de óleo e gás

e no craqueamento do óleo das Bacias de Solimões e Amazonas, sem as quais, não haveria calor suficiente para a transformação do querogênio em petróleo.

3.1 Estratigrafia da Bacia de Solimões

De acordo com BARATA e CAPUTO [10], as grandes sequências estratigráficas da bacia foram definidas a partir da revisão litoestratigráfica realizada por EIRAS *et al.* [22] (3.1).

A seção paleozóica é composta por rochas com idades variando do Ordoviciano ao Permiano. Nesta seção, encontram-se as formações Benjamin Constant, Jutaí, Uerê, Jandiatuba e o grupo Tefé. Neste pacote paleozóico estão ainda as intrusões dadas por três corpos principais de soleiras de diabásio. O evento magmático responsável por estas formações ocorreu por volta de 220 M.a. [13], coincidente com o evento magmático Penatecaua, associado à abertura do Atlântico Norte. Já a seção Mesozóica/Cenozóica estende-se do Cretáceo ao Terciário-Quaternário, sendo representada pelas formações Alter do Chão e Solimões (3.1 e 3.2).

Figura 3.1: Seção geológica esquemática da Bacia do Solimões

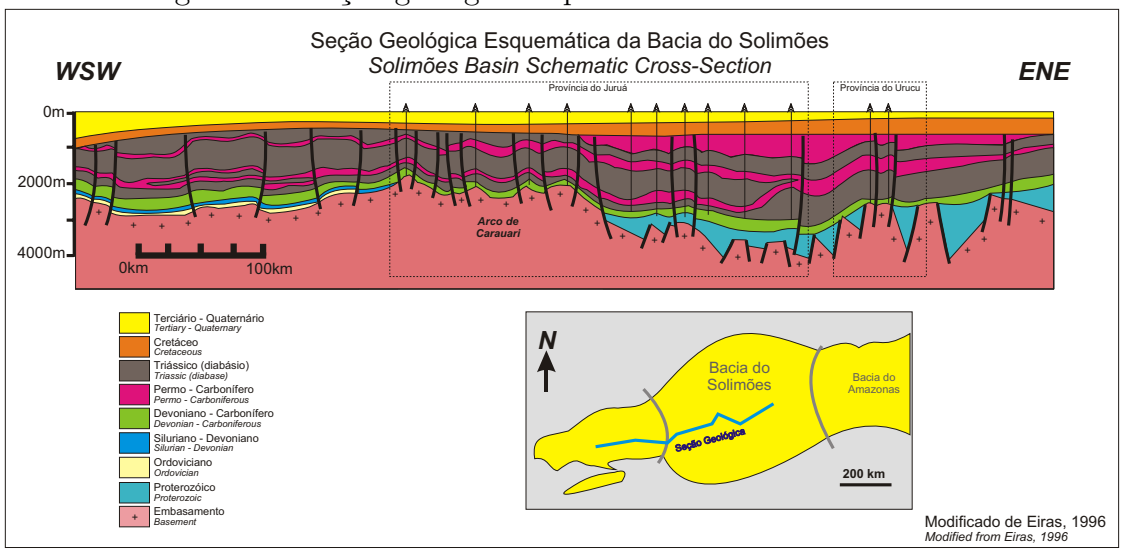
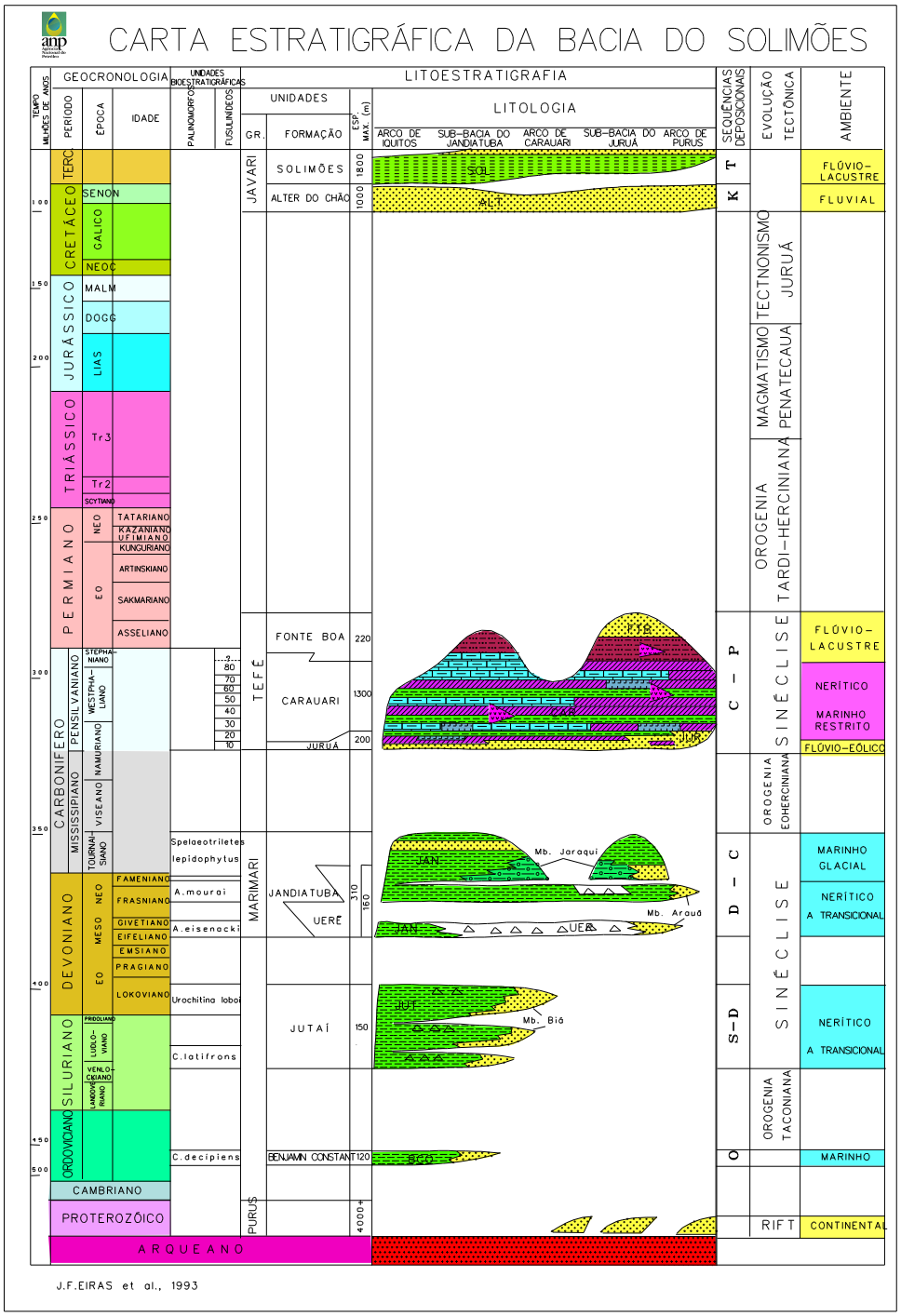


Figura 3.2: Carta estratigráfica da Bacia do Solimões



3.2 Sistemas petrolíferos

De acordo com MAGOON e DOW [6], um sistema petrolífero ativo está associado à existência de quatro elementos (rochas geradoras maturas, rochas reservatório, rochas selantes e trapas) e dois fenômenos geológicos temporais (migração e sincronismo dos eventos, além do trapeamento). Considerando, portanto, tais características, EIRAS [12, 13] propôs a existência dos sistemas petrolíferos Jandiatuba-Juruá(!) e Jandiatuba-Uerê(.).

O Sistema Petrolífero de Jandiatuba-Juruá(!) é o mais importante, uma vez que é responsável por quase que a totalidade das acumulações comerciais de petróleo da bacia. É caracterizado como atípico, com rochas geradoras compostas por folhelhos radioativos e rocha-reservatório carbonífera, ambas com mais de 40 metros de espessura; possui ainda valores médios de TOC acima dos 4,0% e reflectância da vitrinita acima de 1,35%; o sistema conta com excelentes rochas selantes evaporíticas, localizadas acima da rocha reservatório, além de trapas formadas no Paleozóico. Segundo EIRAS [12, 13], a expulsão do petróleo ocorrida no Triássico ocorreu por efeito do calor das intrusões ígneas.

Já o Sistema Petrolífero de Jandiatuba-Uerê(.) é ainda considerado como hipotético e, portanto, seu estudo mais aprofundado foge do contexto deste trabalho.

Capítulo 4

Modelo Matemático

Calor, ou energia, pode ser transferido de um ponto ao outro em uma bacia sedimentar, através de três possibilidades: condução, convecção, e radiação.

A condução de calor é definida como a transferência de energia térmica por contatos (difusão) devido a existencia de um gradiente térmico. O parâmetro que caracteriza essa modalidade de transferência é a condutividade térmica. Ela usualmente diminui de sólido para líquidos e para gases e se torna mais importante em materiais que apresentam maior densidade.

Convecção já diz respeito à energia que é transportada, ou advectada, via a movimentação de um fluido. Em bacias sedimentares é usualmente relacionada ao fluxo em meio poroso de fluidos lá contidos, por exemplo, água. A convecção pode ser mais eficiente em termos de transferência de calor do que a condução, quando há altas vazões de fluidos, por exemplo, ao longo de uma camada extremamente permeável ou em uma fratura. Contudo, note que para a situação em que a camada apresente baixa permeabilidade, não há efetivo transporte de massa e, portanto, de energia, configurando-se como uma pequena parcela do valor total transferido.

A transferência de calor por radiação é aquela feita via ondas eletromagnéticas. Contudo, como esse tipo de transferência de calor é mais atuante somente nos primeiros 300 m de profundidade da bacia e a escala utilizada para modelar a mesma é em torno de uma ordem de grandeza maior, em uma condição natural, pode ser negligenciada em sedimentos. Há ainda a questão do calor gerado pela presença de rochas radioativas, como os folhelhos geradores de hidrocarboneto e as rochas cristalinas do embasamento das bacias, que deve ser levada em conta na modelagem se for oportuno.

A seguir será apresentada a equação completa que modela o fenômeno da transferência de calor em uma bacia sedimentar e será realizada uma discussão acerca da mesma.

4.1 Equação da transferência de calor

A equação da transferência de calor é obtida mediante a realização do balanço de energia dentro de um volume de controle previamente estabelecido. Isto significa que a variação de energia interna dentro desse volume de controle é igual ao calor conduzido para dentro e fora do mesmo, mais a energia transferida por convecção e mais a energia adicionada ao sistema devido à radiação e/ou geração. O resultado segue abaixo:

$$\rho C_p \frac{\partial T}{\partial t} - \nabla \cdot \overline{k} \nabla T = \rho_p C_{p_p} \nabla \cdot (v_p T) + Q_r \tag{4.1}$$

onde ρ, $\overline{k}$ e C_p são, respectivamente, a densidade, tensor de condutividades térmicas e calor específico da rocha; ρ_p, C_{p_p}, v_p são, respectivamente, a densidade, calor específico e velocidade do fluido no poro e Q_r é uma densidade de energia produzida devido radiação.

É interessante analisar termo a termo de (4.1). Compreender a característica de cada um deles é importante, pois dão prontamente uma idéia de como abordar o problema do ponto de vista numérico e quais as dificuldades que por ventura irão ocorrer.

$$\underbrace{\rho C_p \frac{\partial T}{\partial t}}_{\text{Acumulação}} - \underbrace{\nabla \cdot \overline{k} \nabla T}_{\text{Difusão}} = \underbrace{\rho_p C_{p_p} \nabla \cdot (v_p T)}_{\text{Advecção}} + \underbrace{Q_r}_{\text{Fonte/Sumidouro}} \tag{4.2}$$

Termos advectivos impõem restrições quanto aos passos de tempo uma vez que faz-se necessário atender a algum critério, como por exemplo, o critério de CFL [23]. Termos difusivos são responsáveis pela difusão numérica dos dados, ou seja, leva a uma suavização das soluções obtidas. O termo de acumulação diz respeito ao quanto de energia fica retido em um volume de controle. Já o termo fonte/sumidouro representa um valor que provoca, ou o acréscimo, ou o decréscimo da variável primária.

Para o caso tratado neste trabalho, (4.1) passa por uma série de simplificações. Primeiramente, ρ, k e C_p são considerados constantes. O termo fonte é abandonado, justamente pelo argumento utilizado quando foi explicada a característica da transferência de calor por radiação. Em especial, para o caso da Bacia do Solimões, o efeito térmico gerado pela presença de rochar radioativas é muito menor em relação ao causado pelas intrusões magmáticas. Deste modo, o termo fonte poderia ser desconsiderado da equação de balanço geral. O termo advectivo também pode ser deixado de lado, uma vez que as intrusões magmáticas encontram-se entre camadas de folhelhos e evaporítos. Como tal, apresenta baixas permeabilidades e, deste modo, a velocidade do escoamento de fluido seria extremamente baixa. Assim, esse termo iria exercer uma influência bastante reduzida no balanço global de energia e,

como seu tratamento numérico é mais complexo, deixá-lo de lado é interessante do ponto de vista da implementação numérica e computacional.

A equação (4.1) fica, então, como:

$$\rho C_p \frac{\partial T}{\partial t} - \nabla \cdot \overline{k}\, \nabla T = 0. \tag{4.3}$$

A equação acima é totalmente difusiva e émais comumente conhecida como a equação de condução do calor.

Acerca de (4.3) pode-se fazer uma análise mais física do problema. Reescrevendo-a como:

$$\frac{\partial \left(\rho C_p T\right)}{\partial t} - \nabla q' = 0, \tag{4.4}$$

onde $q' = -k\nabla T$ é a lei de Fourrier para transferência de calor por condução.

Note que o produto $\rho C_p T$ possui unidade de energia. Fazendo, deste modo, $E = \rho C_p T$ pode-se reescrever (4.4) do seguinte modo:

$$\frac{\partial E}{\partial t} - \nabla q' = 0. \tag{4.5}$$

A inspeção de (4.5) mostra que na realidade o que se tem é a acumulação de energia e sua variação ao longo do domínio. Essa é uma abordagem mais física do problema dado que não se transfere temperatura e sim energia, no caso entalpia.

Por fim, como neste trabalho será tratada uma seção bidimensional da Bacia do Solimões, (4.3) fica

$$\rho C_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x} k_x \frac{\partial T}{\partial x} + \frac{\partial}{\partial y} k_y \frac{\partial T}{\partial y}. \tag{4.6}$$

4.2 Condições de contorno

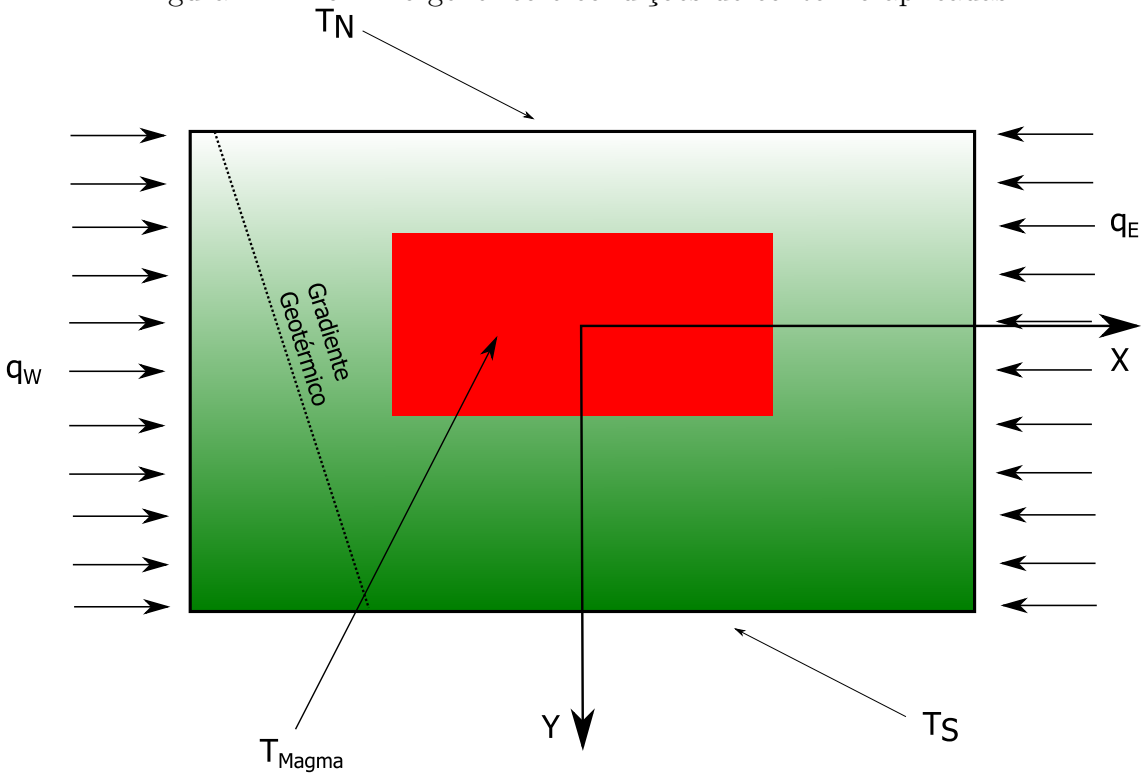
As condicções de contorno e iniciais a serem utilizadas neste trabalho podem ser assim enunciadas $\forall\, x,\, y \in \mathbb{R}$ e $t \geq 0$:

$$\begin{cases} T\left(0, x, y\right) = f(x, y) \\ k_x\, \partial T/\partial x\left(t, 0, y\right) = q_W \\ k_x\, \partial T/\partial x\left(t, L_x, y\right) = q_E \\ T\left(t, x, 0\right) = T_N \\ T\left(t, x, L_y\right) = T_S \end{cases}$$

onde $f(x, y)$ é a distribuição inicial espacial de temperaturas, q_W é um fluxo prescrito no bordo Oeste da bacia, q_E é um fluxo prescrito no bordo Leste da bacia, T_N é a

temperatura prescrita no bordo Norte da bacia e T_S é a temperatura prescrita no bordo Sul da bacia.

Figura 4.1: Domínio genérico e condições de contorno aplicadas



Capítulo 5

Modelo Numérico

Para a obtenção da solução do fenômeno físico da condução de calor em um domínio mais complexo (4.6), é necessário a aplicação de um método numérico. Para tal, neste trabalho selecionou-se o o Método das Diferenças Finitas (MDF), no caso, a classe de *métodos de direções alternadas* (ADI).

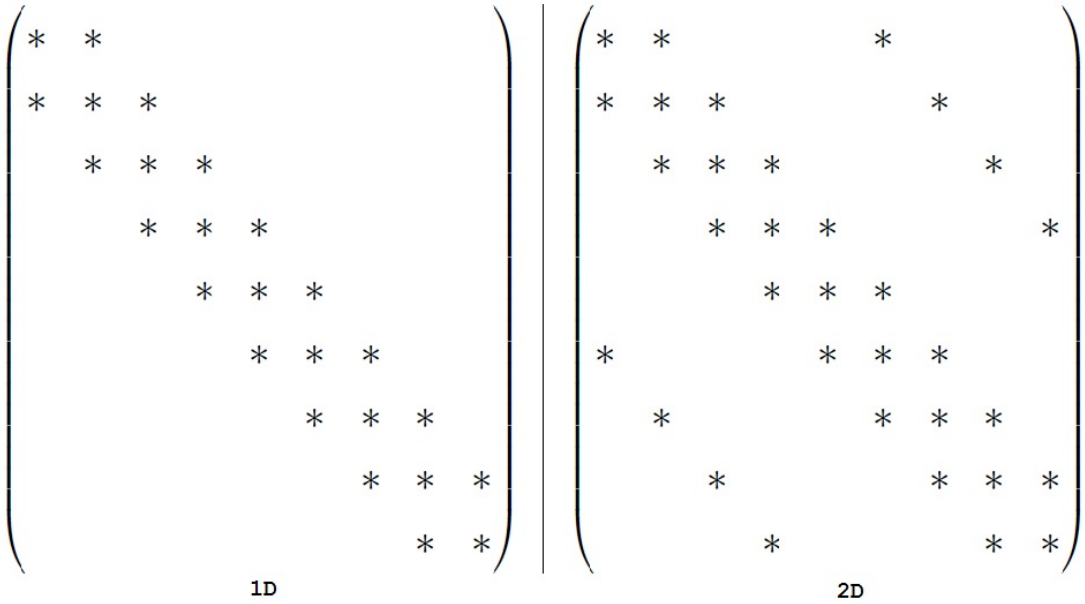
A escolha do método ADI deveu-se a uma série de questões, a saber:

- tempo de processamento;
- quantidade de memória requerida;
- facilidade de implementação;
- estabilidade do método;

Segundo MALISKA [3], a estrutura da matriz de coeficientes obtida na aproximação numérica é de fundamental importância na escolha do método de solução do sistema linear, resultando em formulações tridiagonais para problemas unidimensionais, pentadiagonal para problemas em duas dimensões, e heptadiagonal para situações tridimensionais. Deste modo, ANDERSON [1] mostra que a questão do tempo de processamento e a quantidade de memória requerida para o armazenamento dos dados estão relacionadas a uma característica bastante peculiar dessa classe de métodos: para o caso bidimensional, em uma malha igualmente dividida(nxn), a sua aplicação resulta na formação de $2n$ sistemas lineares tridiagonais (equivalente a se resolverem $2n$ problemas unidimensionais), cuja resolução é extremamente simples por permitirem a utilização do *algoritmo de Thomas* (TDMA) [24], obtendo-se deste modo a solução por meio de um método direto. Esquemas totalmente implícitos levam ao surgimento, no caso bidimensional, de sistemas lineares pentadiagonais, cujos métodos de resolução (diretos ou iterativos) são altamente custosos do ponto de vista computacional, se comparados à resolução

dos $2n$ sistemas lineares tridiagonais resultantes da aplicação do método ADI. A tabela 5.1 e a figura abaixo ajudam a corroborar o exposto.

Figura 5.1: Estrutura da matriz de coeficientes



Além do mais, a aplicação de estratégias para trabalhar com sistemas esparsos é muito mais simples em sistemas tridiagonais, uma vez que basta armazenar a diagonal principal e as *supradiagonais* superior e inferior. Deste modo, a quantidade de memória requerida para armazenamento dos dados necessário para o avanço da simulação é reduzida.

A facilidade de implementação mencionada pode ser justificada pelo enunciado nos parágrafos anteriores, uma vez que problemas bidimensionais tratados semi-implicitamente pelo método ADI levam à formação de sistemas tridiagonais, estes sendo facilmente resolvíveis pelo TDMA, que possui implementação computacional extremamente simples e que será desenvolvida posteriormente.

Tabela 5.1: Índice de esparsividade em problemas bidimensionais [3]

Malha	10 x 10	20 x 20	40 x 40
Número de elementos da matriz	10^4	16×10^4	256×10^4
Não zeros	500	2.000	8.000
% Não zeros	5%	1,25%	0,312%
Índice de esparsividade	0,95	0,9875	0,9968
Memória requerida[1] [Kbytes]	$\frac{40}{2}$	$\frac{640}{8}$	$\frac{10240}{32}$

[1]O número superior indica a quantidade de Kbytes necessária para armazenar em simples precisão a matriz completa, enquanto o número inferior indica quantos Kbytes serão necessários para armazenar somente os elementos não-nulos desta matriz

Quanto à estabilidade, o método ADI é incondicionalmente estável. Vale ressaltar que, segundo PATANKAR [25], a estabilidade aqui dita é em um sentido matemático, garantindo apenas que oscilações presentes irão eventualmente desaparecer, não garantindo, por outro lado, a obtenção de soluções fisicamente plausíveis.

Nas seções que se seguem será feito um breve resumo histórico e a análise numérica do método, com uma descrição detalhada dos procedimentos numéricos e computacionais inerentes ao mesmo.

5.1 Método ADI

Por muitos anos, métodos de separação de variáveis mostraram ser bastante úteis na solução de equações diferencias parciais (EDPs) multi-dimensionais e transientes. A idéia da separação é a de resolver problemas multi-dimensionais de um modo que somente cálculos unidimensionais sejam requeridos. Esta idéia levou ao desenvolvimento de uma enorme variedade dos denominados, na literatura inglesa, *Alternating Direction Implicit Methods* (ADI).

O método ADI foi primeiramente introduzido por PEACEMAN e RACHFORD, JR. [26] em 1955 para equações diferenciais parciais parabólicas e elípticas em duas dimensões. Os aspectos teóricos do método foram discutidos por DOUGLAS [27] em um artigo na mesma publicação. Em torno de uma década após a primeira apresentação do método ADI, DOUGLAS e GUNN [28] aprofundaram a discussão acerca da utilização de métodos de direções alternadas para problemas transientes de uma maneira geral, abordando questões de estabilidade, consistência e convergência.

THIBAULT [29] comparou 9 métodos numéricos para a solução da equação de condução do calor em um paralelepípedo. Considerando aspectos como acurácia, facilidade de programação, tempo de processamento e armazenamento computacional, o autor esboçou uma classificação, dando alta preferência aos métodos ADI, tanto o original proposto por PEACEMAN *et al.* em [26], quanto outros derivados deste.

Segundo HUNDSDORFER e VERWER [30] a idéia de separação de variáveis tem muito mais a ver com a integração no tempo, ao invés da discretização espacial. Essa afirmação dá a idéia de que as dimensões tempo e espaço são desacopladas e analisadas independentemente. Tal idéia é então o cerne do método ADI. Note, porém, que em termos gerais, o método pode envolver o desacoplamento de qualquer dimensão, seja espacial, temporal ou ainda qualquer outra que não as citadas.

5.1.1 Desenvolvimento do método

Nesta seção serão feitos dois desenvolvimentos distintos para o método. Primeiramente um, baseado em operadores diferenciais escritos em notação matricial, pelos quais a idéia do método é apresentada de modo claro e conciso, para o caso bidimensional. A seguir, então, será apresentado o desenvolvimento baseado em termos das equações de diferenças, sendo generalizado para N dimensões. Dentro do mesmo item, apresenta-se novamente o seu desdobramento natural para o caso bidimensional e uma discussão acerca do modelo físico utilizado.

Desenvolvimento baseado na teoria de operadores

Seja Ω um domínio no $\mathbb{R}^2$ com contorno $\Gamma = \partial\Omega$ e $J = (0, T]$ o intervalo de tempo, com $T > 0$. O problema a ser considerado é o mesmo descrito na seção anterior, sendo aqui enunciado da seguinte forma:

$$\frac{1}{\kappa}\partial_t T = \nabla.(\nabla T); \quad t \in J; \quad (x, y) \in \Omega \tag{5.1}$$

onde $\kappa = \frac{k}{\rho cp}$ é a difusividade térmica do meio. Por questão de simplicação, seja $\kappa = 1$. A equação (5.1) reduz-se a:

$$\partial_t T = \nabla.(\nabla T). \tag{5.2}$$

Seja definido o operador laplaciano aplicado em T como $A_n T = -(T_i)_i$, onde n é apenas uma numeração para o operador e i diz respeito à derivada na direção espacial i. Assim:

$$A_1 T = -(T_x)_x \quad \text{e} \quad A_2 T = -(T_y)_y$$

são os operadores laplaciano aplicados em T nas direções x e y respectivamente. Pode-se então reescrever (5.1) como:

$$\partial_t T + A_1 T + A_2 T = 0. \tag{5.3}$$

Suponha agora o intervalo $J = (0, T]$ particionado em n subintervalos de modo a $\{0 = t^0 < t^1 < \ldots < t^n = T\}$, onde:

$$t^n = n.\Delta t; \quad n = 0, 1, \ldots, n_t; \quad \Delta t = \frac{T}{n_t} \tag{5.4}$$

e Δt é o passo de tempo.

Para se particionar o domínio espacial, escolhem-se $n_x + 1$ e $n_y + 1$ pontos igual-

mente distribuidos nas direções x e y, respectivamente como:

$$x_i = a_x + i.h_x; \quad i = 0, 1, \ldots, n_x; \quad h_x = \frac{(b_x - a_x)}{n_x} \quad \text{e,} \tag{5.5a}$$

$$y_j = a_y + j.h_y; \quad j = 0, 1, \ldots, n_y; \quad h_y = \frac{(b_y - a_y)}{n_y} \tag{5.5b}$$

onde h_x e h_y são as dimensões de cada partição nas direções x e y, respectivamente.

A discretização temporal emprega o esquema de Cranck-Nicolson, com diferenças centradas em $t = \left(n + \frac{1}{2}\right)\Delta t$. Pela série de Taylor, (5.3) torna-se:

$$\frac{T^{n+1} - T^n}{\Delta t} + \frac{1}{2}\left(A_1 T^{n+1} + A_1 T^n\right) + \frac{1}{2}\left(A_2 T^{n+1} + A_2 T^n\right) + O\left(\Delta t^2\right) = 0$$

ou

$$\left(I + \frac{\Delta t}{2}A_1 + \frac{\Delta t}{2}A_2\right) T^{n+1} = \left(I - \frac{\Delta t}{2}A_1 - \frac{\Delta t}{2}A_2\right) T^n + O\left(\Delta t^3\right). \tag{5.6}$$

Agora, substituindo-se os operadores laplaciano A_1 e A_2 em (5.6) por, respectivamente, suas aproximações espaciais discretas A_{1h} e A_{2h}, diferenças centrais na partição definida em (5.5), tem-se:

$$\left(I + B_1 + B_2\right) T^{n+1} = \left(I - B_1 - B_2\right) T^n = 0 \tag{5.7}$$

onde

$$B_1 = \frac{\Delta t}{2}A_{1h} \quad \text{e} \quad B_2 = \frac{\Delta t}{2}A_{2h}.$$

O procedimento ADI começa propriamente adicionando-se $B_1 B_2 T^{n+1}$ a ambos os lados de (5.7). Deste modo, tem-se :

$$\begin{aligned}\left(I + B_1 + B_2 + B_1 B_2\right) T^{n+1} = \\ \left(I - B_1 - B_2 + B_1 B_2\right) T^n + B_1 B_2 \left(T^{n+1} - T^n\right).\end{aligned} \tag{5.8}$$

A equação (5.8) pode ser fatorada como:

$$\begin{aligned}\left(I + B_1\right)\left(I + B_2\right) T^{n+1} = \\ \left(I - B_1\right)\left(I - B_2\right) T^n + B_1 B_2 \left(T^{n+1} - T^n\right).\end{aligned} \tag{5.9}$$

De (5.9), pode-se chegar ao seguinte resultado:

$$T^{n+1} = T^n + O\left(\Delta t\right). \tag{5.10}$$

Repare que devido ao fator Δt^2, proveniente do produto $B_1 B_2$ no lado direito de (5.9), e por (5.10), o segundo termo do lado direito de (5.9) torna-se da $O\left(\Delta t^3\right)$, o qual é da mesma ordem que o erro de truncamento. Assim, pode-se ignorar o termo

$$G_h\left(T^{n+1}, T^n\right) = B_1 B_2\left(T^{n+1} - T^n\right) \tag{5.11}$$

permtindo-se reescrever (5.9) como:

$$\left(I + B1\right)\left(I + B2\right) T^{n+1} = \left(I - B1\right)\left(I - B2\right) T^n. \tag{5.12}$$

Por fim, para resolver (5.12) é proposto em [27], [31] e [26]

$$\left(I + B_1\right) \tilde{T}^{n+\frac{1}{2}} = \left(I - B_2\right) T^n \quad \text{e,} \tag{5.13a}$$

$$\left(I + B_2\right) T^{n+1} = \left(I - B_1\right) \tilde{T}^{n+\frac{1}{2}} \tag{5.13b}$$

onde em (5.13a)resolve-se a direção x implicitamente, mantendo-se a y explícita e, a seguir, com essa solução parcial resolve-se o inverso em (5.13b), com a direção y tratada implicitamente e a x explicitamente, deste modo, completando o passo inteiro de tempo.

Em resumo, o método se realiza em duas etapas, a saber:

$$\begin{cases} \left(t \to t + \frac{1}{2}\right) \Rightarrow \text{direção x tratada implicitamente} \\ \left(t + \frac{1}{2} \to t + 1\right) \Rightarrow \text{direção y tratada implicitamente} \end{cases}$$

Note que para cada direção resolvida, a matriz a ser invertida é tridiagonal. Vale ressaltar também que o algoritmo requer $O\left(N\right)$ flops de todos os cálculos realizados.

Figura 5.2: Resolução da direção x [1]

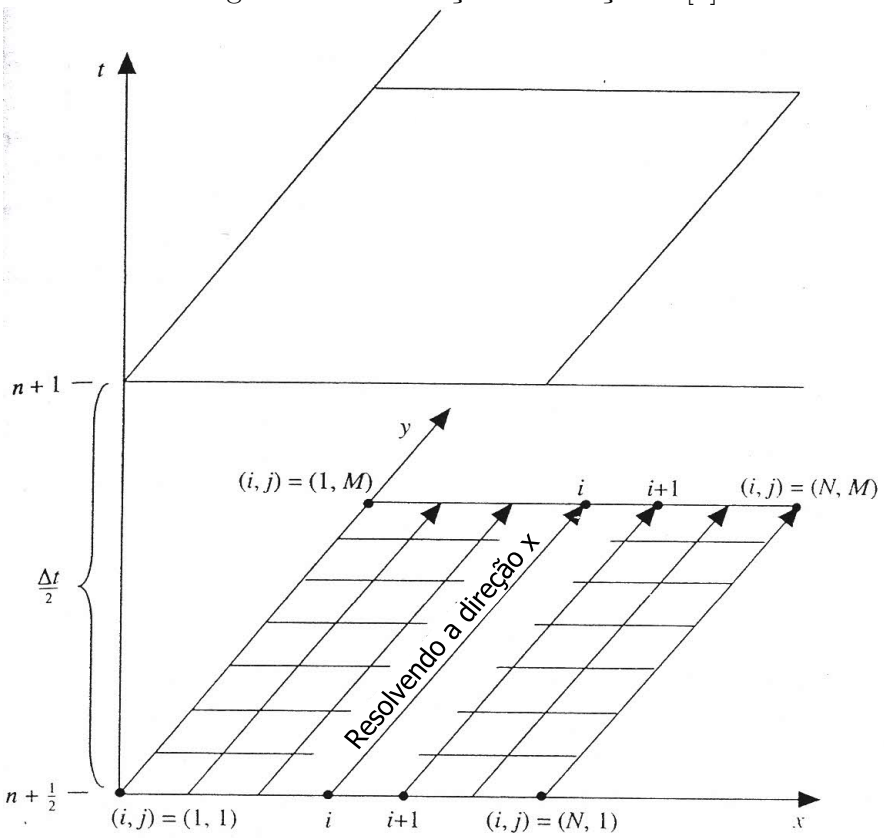
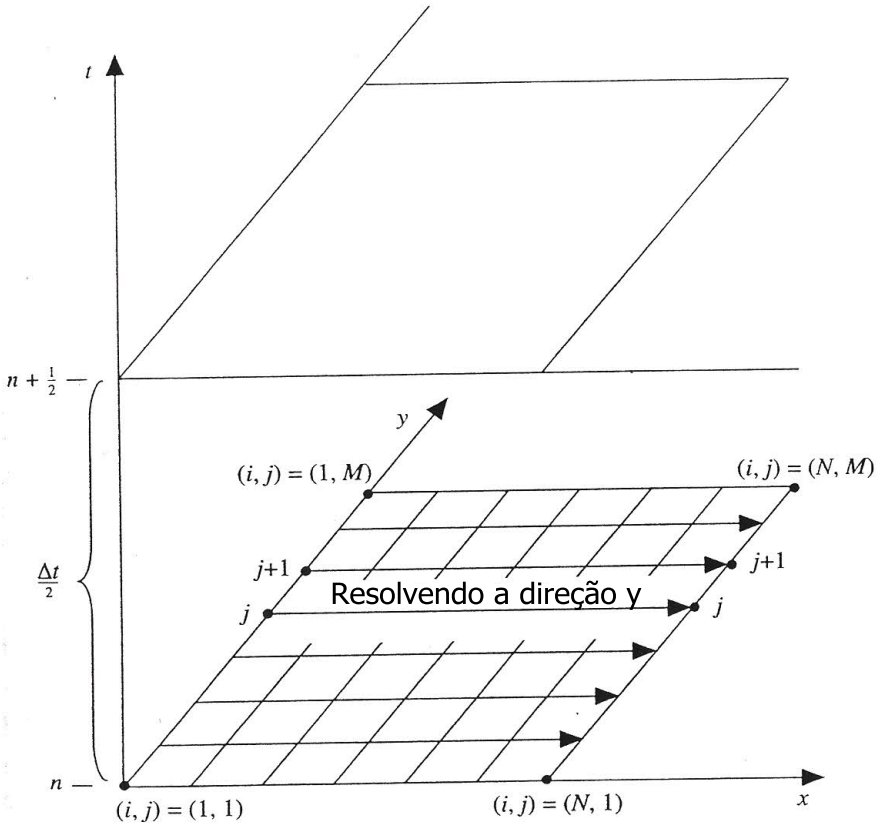


Figura 5.3: Resolução da direção y [1]



Desenvolvimento em equação de diferenças generalizadas

Retornando à equação (5.1), porém, agora a generalizando para N dimenssões em qualquer sistema de coordenadas curvilíneas ortogonais tem-se:

$$\frac{\partial T}{\partial t} = \frac{1}{g}\sum_{i=1}^{N} \kappa_i \frac{\partial}{\partial u_i}\left(\frac{g}{g_i^2}\frac{\partial T}{\partial u_i}\right); \quad (N \geq 2) \tag{5.14}$$

onde g_i é um fator de escala e $g = \prod_{i=1}^{N} g_i$. Novamente, por simplicidade, o desenvolvimento será feito para o sistema Cartesiano, no qual todos $g_i = 1$. Ao definirem-se as variáveis adimensionais

$$\theta = \frac{T - T_\infty}{T_0 - T_\infty}, \quad \tau = \frac{\kappa t}{L^2}, \quad \eta = \frac{u_i}{L}, \quad (i = 1, 2, \ldots, N) \tag{5.15}$$

(5.14) torna-se:

$$\frac{\partial \theta}{\partial \tau} = \sum_{i=1}^{N} \frac{\partial^2 \theta}{\partial \eta_i^2}. \tag{5.16}$$

Agora separando os termos difusivos no lado direito de (5.16) e alternadamente deixando cada direção espacial ser representada implicitamente enquanto as outras explicitamente no $\frac{1}{N}$ intervalo de tempo. Este procedimento leva a um esquema de N passos:
no passo 1

$$\frac{\theta^{(1)}_{k_1,k_2,\ldots,k_N} - \theta^{n}_{k_1,k_2,\ldots,k_N}}{\Delta\tau/N} = \frac{\delta^2_{\eta_1}\theta^{(1)}_{k_1,k_2,\ldots,k_N}}{\Delta\eta_1^2} + \sum_{i=2}^{N} \frac{\delta^2_{\eta_i}\theta^{n}_{k_1,k_2,\ldots,k_N}}{\Delta\eta_i^2}, \tag{5.17}$$

no passo j, $j \in (j = 2, \ldots, N-1)$,

$$\frac{\theta^{(j)}_{k_1,k_2,\ldots,k_N} - \theta^{(j-1)}_{k_1,k_2,\ldots,k_N}}{\Delta\tau/N} = \frac{\delta^2_{\eta_j}\theta^{(j)}_{k_1,k_2,\ldots,k_N}}{\Delta\eta_j^2} + \sum_{\substack{i=1 \\ i\neq j}}^{N} \frac{\delta^2_{\eta_i}\theta^{(j-1)}_{k_1,k_2,\ldots,k_N}}{\Delta\eta_i^2}, \tag{5.18}$$

e, finalmente, no passo N

$$\frac{\theta^{n+1}_{k_1,k_2,\ldots,k_N} - \theta^{(N-1)}_{k_1,k_2,\ldots,k_N}}{\Delta\tau/N} = \frac{\delta^2_{\eta_N}\theta^{n+1}_{k_1,k_2,\ldots,k_N}}{\Delta\eta_N^2} + \sum_{i=1}^{N-1} \frac{\delta^2_{\eta_i}\theta^{(N-1)}_{k_1,k_2,\ldots,k_N}}{\Delta\eta_i^2}, \tag{5.19}$$

onde $\theta^{(j)}$, $(j = 1, 2, \ldots, N-1)$ são os valores intermediários de temperatura associados ao j-*ésimo* passo, com $\theta^{(0)} = \theta^n$ e $\theta^{(N)} = \theta^{n+1}$; e δ^2 é o operador diferencial de segunda ordem. Os subscritos n e $n+1$ denotam dois passos de tempo consecutivos. Definindo-se o número de Fourier como $r_i = \frac{\Delta\tau}{\Delta\eta_i^2}$ e substituindo-se δ^2 por diferenças

centrais, que representam as derivadas segundas, as equações (5.17), (5.18) e (5.19) podem ser rearranjadas como:
no passo 1

$$\begin{aligned}
&-r_1\theta^{(1)}_{k_1-1,k_2,\ldots,k_N} + (N+2r_1)\theta^{(1)}_{k_1,k_2,\ldots,k_N} - r_1\theta^{(1)}_{k_1+1,k_2,\ldots,k_N} \\
&= \left(N - 2\sum_{i=2}^{N} r_i\right)\theta^{n}_{k_1,k_2,\ldots,k_N} + \sum_{i=2}^{N} r_i\left(\theta^{n}_{k_1,\ldots,k_i-1,\ldots,k_N} + \theta^{n}_{k_1,\ldots,k_i+1,\ldots,k_N}\right),
\end{aligned} \tag{5.20}$$

no passo j, $j \in (j = 2, \ldots, N-1)$,

$$\begin{aligned}
&-r_j\theta^{(j)}_{k_1,\ldots,k_j-1,\ldots,k_N} + (N+2r_j)\theta^{(j)}_{k_1,\ldots,k_j,\ldots,k_N} - r_j\theta^{(j)}_{k_1,\ldots,k_j+1,\ldots,k_N} \\
&= \left(N - 2\sum_{\substack{i=1\\ i\neq j}}^{N} r_i\right)\theta^{(j-1)}_{k_1,\ldots,k_j,\ldots,k_N} + \sum_{\substack{i=1\\ i\neq j}}^{N} r_i\left(\theta^{(j-1)}_{k_1,\ldots,k_i-1,\ldots,k_N} + \theta^{(j-1)}_{k_1,\ldots,k_i+1,\ldots,k_N}\right),
\end{aligned} \tag{5.21}$$

e, finalmente, no passo N

$$\begin{aligned}
&-r_N\theta^{n+1}_{k_1,k_2,\ldots,k_N-1} + (N+2r_N)\theta^{n+1}_{k_1,k_2,\ldots,k_N} - r_N\theta^{n+1}_{k_1,k_2,\ldots,k_N+1} \\
&= \left(N - 2\sum_{i=1}^{N-1} r_i\right)\theta^{(N-1)}_{k_1,k_2,\ldots,k_N} + \sum_{i=1}^{N-1} r_i\left(\theta^{(N-1)}_{k_1,\ldots,k_i-1,\ldots,k_N} + \theta^{(N-1)}_{k_1,\ldots,k_i+1,\ldots,k_N}\right).
\end{aligned} \tag{5.22}$$

Nas equações anteriores, todos os termos implícitos, com valores de temperaturas desconhecidos foram movidos para o lado esquerdo, enquanto que todos os termos explícitos, com os valores já conhecidos de temperatura, foram movidos para o lado direito. Note que as equações (5.20), (5.21) e (5.22) são válidas para os nós internos da malha utilizada. As fórmulas para os nós localizados nas fronteiras dependem do tipo de condição de contorno utilizada e não serão elaboradas para N dimensões. Para o caso bidimensional, foco deste trabalho, podem ser conferidas no apêndice A.

Ao combinar as três últimas equações para cada nó interno junto daquelas para os nós do contorno, obtém-se N equações matriciais, cada uma consistindo de $\prod_{i=1}^{N} m_i$ equações algébricas lineares, onde m_i é o número de temperaturas incógnitas no i-*ésimo* espaço.

$$\begin{aligned}
[C_1]\,\{\theta^{(1)}\} &= \{b_1\}, \\
[C_j]\,\{\theta^{(j)}\} &= \{b_j\}, \\
&\vdots \\
[C_N]\,\{\theta^{n+1}\} &= \{b_N\}.
\end{aligned}$$

Os coeficientes do lado esquerdo das equações (5.20), (5.21) e (5.22) formam os elementos tridiagonais de $[C_j]$, $(j = 1, 2, \ldots, N)$, respectivamente, enquanto os termos do lado direito dessas equações formam os vetores $\{b_j\}$, $(j = 1, 2, \ldots, N)$, respectivamente. Cada uma dessas equações são facilmente resolvíveis, conforme já citado anteriormente, pelo emprego do TDMA. Do ponto de vista da implementação computacional, o TDMA consiste de N laços, mais os procedimentos de inicialização, que possuem tempo constante. Deste modo, o tempo de processamento é da $O\left(\prod_{i=1}^{N} m_i\right)$. Como há N matrizes a serem resolvidas, o tempo total de processamento é da $O\left(N \prod_{i=1}^{N} m_i\right)$.

Por outro lado, como os coeficientes dependem essencialmente das propriedades físicas e geométricas de cada nó e, além disso, a cada passo somente é necessário o armazenamento das temperaturas intermediárias mais recentemente obtidas para o cáculo do passo presente, a capacidade de armazenamento computacional para manter toda a informação requerida é da $O\left(\prod_{i=1}^{N} m_i\right)$.

Vale ressaltar que a análise das equações (5.20), (5.21) e (5.22) revela que os coeficientes dos primeiros termos do lado direito das mesmas irão tornar-se negativos se

$$\sum_{\substack{i=1 \\ i \neq j}}^{N} r_i \geq \frac{N}{2}, \quad (j = 1, 2, \ldots, N). \tag{5.23}$$

Coeficientes negativos em equações de diferenças podem gerar contribuições de energia irrealistas fisicamente e, além disso, podem causar resultados inesperados, tais como oscilações nas soluções obtidas e levar a uma baixa acurácia. Para evitar esses problemas, deste modo, garantindo a obtenção de soluções adequadas fisicamente, deve-se garantir a positividade dos coeficientes. Assim,

$$r \leq \frac{N}{2\,(N-1)} \tag{5.24}$$

onde $r = r_{máx}$.

Existem algumas técnicas para amortecer essas oscilações. Segundo BRITZ *et al.* [4] a primeira possibilidade seria a de utilizar o método ADI conforme aqui descrito, valendo-se de passos de tempo bem pequenos. A segunda alternativa seria a de utilizar alguma variação do mesmo, mas desta vez adicionando algum tipo de correção numérica. A terceira, e aqui utilizada, é a de subdividir o primeiro passo de tempo em um grupo de subintervalos. Ao uso de subintervalos iguais dá-se o nome de método de Pearson [32]. A quarta possibilidade seria a de se resolver o primeiro passo de tempo através de um esquema totalmente implícito e os posteriores utilizando-se normalmente o método ADI. Por fim, a quinta possibilidade seria a de se aproximar o perfil térmico inicial através de alguma solução analítica. O esquema

de subdivisão é apresentado abaixo, como segue:

Para subintervalos expandidos exponecialmente utiliza-se:

$$\xi_j = \beta \xi_{j-1} \tag{5.25}$$

com ξ_1 escolhido como:

$$\xi_1 = \Delta t \frac{\beta - 1}{\beta^M - 1} \tag{5.26}$$

e assegurando-se que:

$$\sum_{j=1}^{M} \xi_j = \Delta t. \tag{5.27}$$

Para subintervalos igualmente espaçados (Pearson, $\beta = 1,00$), tem-se simplesmente:

$$\xi = \frac{\Delta t}{M}. \tag{5.28}$$

Em que M é o número de subintervalos, ξ o novo passo de tempo a ser utilizado durante o primeiro passo de tempo do método ADI e β um fator de expansão ou contração.

Tabela 5.2: Comparação entre diferentes métodos de amortecimentos de oscilações numéricas [4]

Método	N_T	CPU/s
Peaceman-Rachford	2000	5.62
Douglas-Rachford	200	0.55
Douglas-Rachford simplificado	1000	2.18
Peaceman-Rachfor (subdivisão)[2]	100	2.18
Peaceman-Rachfor (subdivisão)[3]	100	0.58
Douglas-Rachford e Peaceman-Rachford[4]	100	0.29
Totalmente implícito[5]	50	1.42

A apresentação do caso bidimensional, interesse deste trabalho, e que será utilizado para a realização da simulação da estrutura térmica da bacia sedimentar, torna-se trivial após a apresentação que se seguiu, bastando utilizar $N = 2$ nas relações anteriores. Temos então a seguinte discretização para o caso $2D$:
no passo $\left(t \to t + \frac{1}{2}\right)$

$$\begin{aligned} -r_i\theta_{j,i-1}^{n+\frac{1}{2}} + (2 + 2r_i)\theta_{j,i}^{n+\frac{1}{2}} - r_i\theta_{j,i+1}^{n+\frac{1}{2}} \\ = (2 - 2r_j)\,\theta_{j,i}^{n} + r_j\left(\theta_{j-1,i}^{n} + \theta_{j+1,i}^{n}\right), \end{aligned} \tag{5.29}$$

[2]Método de Pearson, $M = 100$

[3]Subdivisão utilizando passos de tempo exponencialmente expandidos, $M = 100$, $\beta = 1,02$

[4]2 ou 4 Douglas-Rachford pré passos de tempo

[5]Um passo totalmente implícito

no passo $\left(t+\frac{1}{2}\rightarrow t+1\right)$

$$\begin{aligned}-r_j\theta_{j-1,i}^{n+1}+(2+2r_j)\theta_{j,i}^{n+1}-r_j\theta_{j+1,i}^{n+1}\\ =(2-2r_j)\,\theta_{j,i}^{n+\frac{1}{2}}+r_j\left(\theta_{j,i-1}^{n+\frac{1}{2}}+\theta_{j,i+1}^{n+\frac{1}{2}}\right).\end{aligned}\tag{5.30}$$

Ou, em termos dimensionais e reagrupando-se os termos tem-se:
no passo $\left(t\rightarrow t+\frac{1}{2}\right)$

$$\begin{aligned}&\rho C_p h_x h_y\left(T_{j,i}^{n+\frac{1}{2}}-T_{j,i}^{n}\right)-\\ &\quad\frac{\Delta t h_y}{2}\left(k_e\frac{T_{j,i+1}^{n+\frac{1}{2}}-T_{j,i}^{n+\frac{1}{2}}}{h_x}-k_w\frac{T_{j,i}^{n+\frac{1}{2}}-T_{j,i-1}^{n+\frac{1}{2}}}{h_x}\right)-\\ &\quad\frac{\Delta t h_x}{2}\left(k_s\frac{T_{j+1,i}^{n}-T_{j,i}^{n}}{h_y}-k_n\frac{T_{j,i}^{n}-T_{j-1,i}^{n}}{h_y}\right)=0,\end{aligned}\tag{5.31}$$

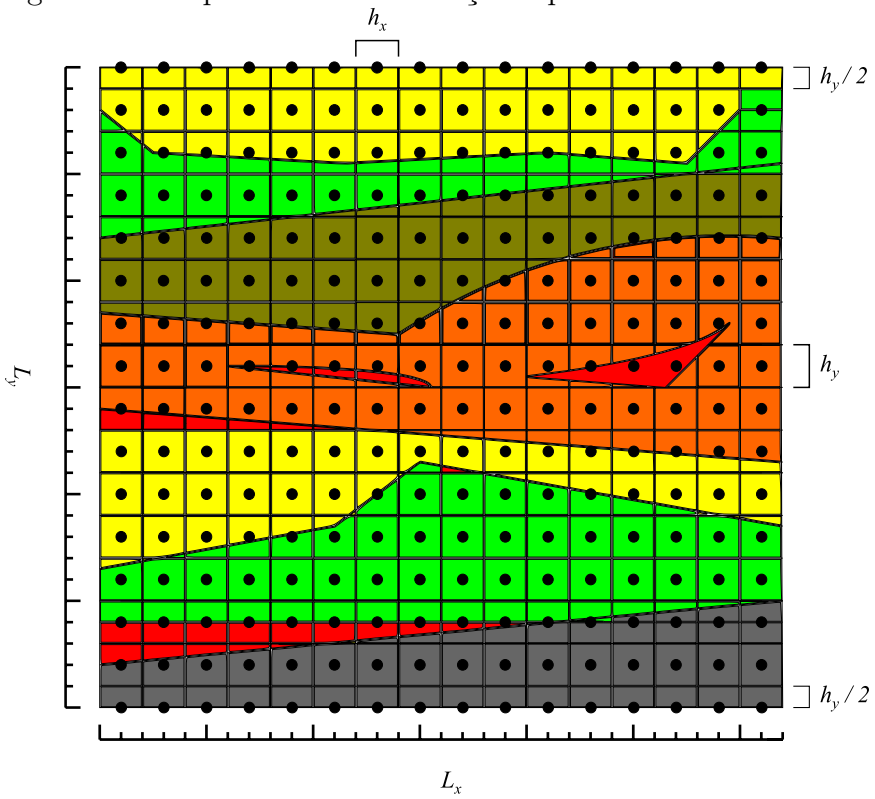
no passo $\left(t+\frac{1}{2}\rightarrow t+1\right)$

$$\begin{aligned}&\rho C_p h_x h_y\left(T_{j,i}^{n+1}-T_{j,i}^{n+\frac{1}{2}}\right)-\\ &\quad\frac{\Delta t h_x}{2}\left(k_s\frac{T_{j+1,i}^{n+1}-T_{j,i}^{n+1}}{h_y}-k_n\frac{T_{j,i}^{n+1}-T_{j-1,i}^{n+1}}{h_y}\right)-\\ &\quad\frac{\Delta t h_y}{2}\left(k_e\frac{T_{j,i+1}^{n+\frac{1}{2}}-T_{j,i}^{n+\frac{1}{2}}}{h_x}-k_w\frac{T_{j,i}^{n+\frac{1}{2}}-T_{j,i-1}^{n+\frac{1}{2}}}{h_x}\right)=0,\end{aligned}\tag{5.32}$$

onde k_s, k_n, k_w e k_e são as condutividades térmicas calculadas nas interfaces de cada bloco centrado em (j,i). Note que a formulação apresentada acima para o caso bidimensional, em termos de equações de diferenças, é análoga ao resultado alcançado em (5.13), sendo esta obtida por meio de outro desenvolvimento.

É interessante, ainda, notar que a física do problema de condução de calor é preservada mesmo após a discretização. Observe que o primeiro termo de (5.31) e (5.32) representa uma quantidade de energia, no caso representada pela entalpia, que é acumulada pela partição centrada em (j,i) devido á marcha no tempo. Os outros termos das mesmas equações representam o balanço de energia que entra e sai desse volume, calculado nas suas faces. Assim, por mais que o MDF seja um método essencialmente matemático, a física do problema é preservada de modo natural. Neste caso, o conjunto de equações (5.31) e (5.32) representa a conservação de energia calculada em cada pequeno espaço discreto, que na sua totalidade compõe o domínio físico de interesse.

Figura 5.4: Esquema da discretização espacial de um domínio Ω



Segundo PATANKAR [25], as condutividades térmicas nas interfaces são calculadas com maior exatidão física utilizando-se a média harmônica, ao invés de uma interpolação linear. Assim, estas ficam como se segue:

$$k_e = 2\frac{k_{j,i+1}k_{j,i}}{k_{j,i+1} + k_{j,i}}, \tag{5.33a}$$

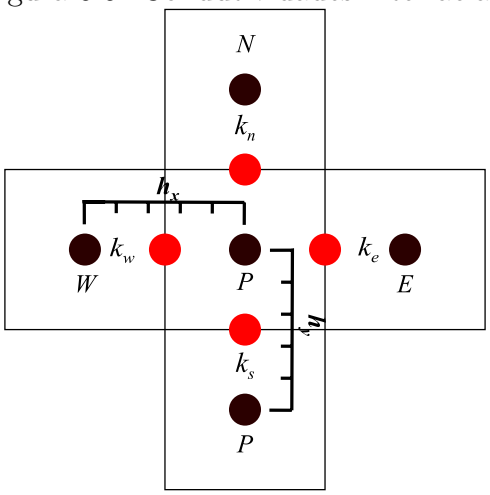
$$k_w = 2\frac{k_{j,i}k_{j,i-1}}{k_{j,i} + k_{j,i-1}}, \tag{5.33b}$$

$$k_s = 2\frac{k_{j+1,i}k_{j,i}}{k_{j+1,i} + k_{j,i}} \quad \text{e,} \tag{5.33c}$$

$$k_n = 2\frac{k_{j,i}k_{j-1,i}}{k_{j,i} + k_{j-1,i}}. \tag{5.33d}$$

Ainda no tocante às condutividades térmicas, no caso de bacias sedimentares, é importante levar em conta a questão da anisotropia do meio. Tal anisotropia é decorrente dos processos deposicionais atuantes na mesma. No geral, pode-se afirmar que sempre haverá ao menos anisotropia vertical, uma vez que devido a atuação da gravidade, os grãos de rocha tendem a se alinharem na direção desse campo, deste modo propiciando o fluxo térmico na direção vertical. Assim, define-se a anisotropia

Figura 5.5: Condutividades interfaciais



f como:

$$f = \frac{k_v}{k_h}, \tag{5.34}$$

em que k_v e k_h são, respectivamente, as condutividades térmicas na direção vertical e horizontal, e $f \geq 1,0$.

Assim, as equações (5.33) podem ser reescritas como:

$$k_e = 2\frac{k_{j,i+1}k_{j,i}}{k_{j,i+1} + k_{j,i}}, \tag{5.35a}$$

$$k_w = 2\frac{k_{j,i}k_{j,i-1}}{k_{j,i} + k_{j,i-1}}, \tag{5.35b}$$

$$k_s = 2\frac{f_{j+1,i}k_{j+1,i} \cdot f_{j,i}k_{j,i}}{f_{j+1,i}k_{j+1,i} + f_{j,i}k_{j,i}} \quad \text{e,} \tag{5.35c}$$

$$k_n = 2\frac{f_{j,i}k_{j,i} \cdot f_{j-1,i}k_{j-1,i}}{f_{j,i}k_{j,i} + f_{j-1,i}k_{j-1,i}}. \tag{5.35d}$$

Ainda tratando-se do caso em duas dimensões, o critério de não-negativadade dos coeficientes requer que:

$$máx\left\{\frac{\kappa\Delta t}{h_x^2}, \frac{\kappa\Delta t}{h_y^2}\right\} \leq 1,0 \tag{5.36}$$

5.1.2 Análise de estabilidade

Um método numérico é dito estável se o erro (ou perturbação) de uma iteração não se propaga nem se amplifica na iteração posterior. Assim, a análise de estabilidade não precisa se preocupar com a origem do erro, e sim, com suas implicações.

Uma equação de diferenças pode ter sua estabilidade testada, simplesmente ao

se introduzir uma perturbação nos valores da solução. Por exemplo:

$$T^{n}_{\epsilon j,i} = T^{n}_{j,i} + \epsilon^{n}_{j,i}, \tag{5.37}$$

onde ϵ é a perturbação introduzida na solução.

Assumindo por questão de simplificação que k, ρ e C_p são constantes ao longo de toda a malha e independentes da temperatura, $h_x = h_y$ e, em vez de andar-se meio passo de tempo a cada "iteração" do método ADI $\left(\frac{\Delta t}{2}\right)$, assumir um passo inteiro para a resolução de cada direção (Δt). O objetivo é avaliar se o método é estável ao se resolver apenas uma das direções (y ou x) implicitamente, enquanto a outra é mantida explícita, sem um passo intermediário de tempo. Ou seja, irá ser avaliado se a resolução em um passo inteiro de tempo de uma das direções, independentemente da outra, é estável em um sentido matemático.

Pode-se reescrever (5.31) como

$$\frac{\rho Cp}{k}\frac{T^{n+1}_{j,i} - T^{n}_{j,i}}{\Delta t} = \frac{T^{n+1}_{j,i+1} - 2T^{n+1}_{j,i} + T^{n+1}_{j,i-1}}{h_x^2} + \frac{T^{n}_{j+1,i} - 2T^{n}_{j,i} + T^{n}_{j-1,i}}{h_y^2} \quad \text{e,} \tag{5.38}$$

aplicando (5.37) chega-se a

$$\begin{aligned}\frac{\rho Cp}{k}\frac{\left(T^{n+1}_{j,i} + \epsilon^{n+1}_{j,i}\right) - \left(T^{n}_{j,i} + \epsilon^{n}_{j,i}\right)}{\Delta t} = \\ \frac{\left(T^{n+1}_{j,i+1} + \epsilon^{n+1}_{j,i+1}\right) - 2\left(T^{n+1}_{j,i} + \epsilon^{n+1}_{j,i}\right) + \left(T^{n+1}_{j,i-1} + \epsilon^{n+1}_{j,i-1}\right)}{h_x^2} + \\ \frac{\left(T^{n}_{j+1,i} + \epsilon^{n}_{j+1,i}\right) - 2\left(T^{n}_{j,i} + \epsilon^{n}_{j,i}\right) + \left(T^{n}_{j-1,i} + \epsilon^{n}_{j-1,i}\right)}{h_y^2}.\end{aligned} \tag{5.39}$$

Subtraindo (5.39) de (5.38) tem-se

$$\frac{\rho Cp}{k}\frac{\epsilon^{n+1}_{j,i} - \epsilon^{n}_{j,i}}{\Delta t} = \frac{\epsilon^{n+1}_{j,i+1} - 2\epsilon^{n+1}_{j,i} + \epsilon^{n+1}_{j,i-1}}{h_x^2} + \frac{\epsilon^{n}_{j+1,i} - 2\epsilon^{n}_{j,i} + \epsilon^{n}_{j-1,i}}{h_y^2}, \tag{5.40}$$

que é a equação perturbada para a resolução da direção x implicitamente, mantendo a y explícita, em um passo completo de tempo. Note que (5.40) apresenta a mesma forma de (5.38). Essa constatação somente será verdade se a equação de diferenças que a originou for linear e homogênea.

Por analogia, resolvendo-se implicitamente a direção y e mantendo-se a x explícita, para um único passo de tempo, tem-se:

$$\frac{\rho Cp}{k}\frac{\epsilon^{n+1}_{j,i} - \epsilon^{n+\frac{1}{2}}_{j,i}}{\Delta t} = \frac{\epsilon^{n+1}_{j+1,i} - 2\epsilon^{n+1}_{j,i} + \epsilon^{n+1}_{j-1,i}}{h_y^2} + \frac{\epsilon^{n}_{j,i+1} - 2\epsilon^{n}_{j,i} + \epsilon^{n}_{j,i-1}}{h_x^2}. \tag{5.41}$$

Para obter o critério de estabilidade do método ADI é comum utilizar a análise de estabilidade de von Neumann. Esse critério consiste na decomposição do erro em uma série de Fourier com a forma

$$\epsilon_{j,i}^{n} = \sum_{p} \sum_{q} \gamma^{n} e^{ipx_i} e^{iqy_j}, \tag{5.42}$$

onde $i = \sqrt{-1}$, $x_i = ih_x$ e $y_j = jh_y$.

Substituindo a série acima nas equações do erro e resolvendo-se para cada modo de oscilação, obtem-se a razão de estabilidade γ, dado por,

$$\gamma = \frac{\gamma_p^{n+1}}{\gamma_p^{n}}\cdot \tag{5.43}$$

Deste modo, o critério de estabilidade de von Neumann é então satisfeito se

$$-1 \leq \gamma \leq 1. \tag{5.44}$$

A relação (5.44) basicamente expressa que o erro não pode aumentar em módulo, ou seja, o erro deve manter-se estável ou tender a zero.

Assumindo que a solução de (5.40) e (5.41) tenha a forma

$$\epsilon_{j,i}^{n} = \gamma^{n} e^{ipih_x} e^{iqjh_y} \tag{5.45}$$

e, substituindo-a nas mesmas, obtem-se:
direção x tratada implicitamente $(n \rightarrow n+1)$

$$\begin{aligned} \frac{\rho Cp}{k}\left(\frac{\gamma^{n+1}-\gamma^{n}}{\Delta t}\right) = &-\left(4/h_x^2\right)\gamma^{n+1}\sin^2\left(ph_x/2\right) \\ &-\left(4/h_y^2\right)\gamma^{n}\sin^2\left(qh_y/2\right), \end{aligned} \tag{5.46}$$

direção y tratada implicitamente $(n \rightarrow n+1)$

$$\begin{aligned} \frac{\rho Cp}{k}\left(\frac{\gamma^{n+1}-\gamma^{n+1}}{\Delta t}\right) = &-\left(4/h_y^2\right)\gamma^{n+1}\sin^2\left(qh_y/2\right) \\ &-\left(4/h_x^2\right)\gamma^{n}\sin^2\left(ph_x/2\right)\cdot \end{aligned} \tag{5.47}$$

Seja $\psi = \frac{k\Delta t}{\rho C_p}$ e substituindo-a em (5.46) e (5.47) é possível se chegar às razões de estabilidade para cada uma das "iterações" do método ADI realizadas em um passo

inteiro de tempo:
direção x tratada implicitamente $(n \to n+1)$

$$\frac{\gamma^{n+1}}{\gamma^n} = \frac{1 - 4\psi \sin^2(ph_x/2)}{1 + 4\psi \sin^2(qh_y/2)}, \tag{5.48}$$

direção y tratada implicitamente $(n \to n+1)$

$$\frac{\gamma^{n+1}}{\gamma^n} = \frac{1 - 4\psi \sin^2(qh_y/2)}{1 + 4\psi \sin^2(ph_x/2)}. \tag{5.49}$$

A inspeção da equação (5.48) mostra que se $\psi > \frac{1}{2}$, $p = 1$ e $q = 1$ o erro se amplifica, assim, (5.40) é instável. Para (5.49) se $\psi > \frac{1}{2}$, $p = 1$ e $q = 1$ o erro novamente amplifica em módulo, deste modo, levando á conclusão de que (5.41) também é instável. Deste modo, a resolução de cada direção de forma independente para um passo inteiro de tempo é instável numericamente.

A idéia para contornar tal problema é supor um passo duplo de tempo $\left(t \to t + \frac{\Delta t}{2} \to t + \Delta t\right)$, ou seja, considera-se o primeiro passo como intermediário usando (5.40) e o passo seguinte usando (5.41). As razões de estabilidade (5.48) e (5.49) ficam, respectivamente

$$\frac{\gamma^{n+\frac{1}{2}}}{\gamma^n} = \frac{1 - 2\psi \sin^2(ph_x/2)}{1 + 2\psi \sin^2(qh_y/2)}, \tag{5.50}$$

$$\frac{\gamma^{n+1}}{\gamma^{n+\frac{1}{2}}} = \frac{1 - 2\psi \sin^2(qh_y/2)}{1 + 2\psi \sin^2(ph_x/2)}. \tag{5.51}$$

Assim, realizando a multiplicação de (5.50) com (5.51) tem-se o seguinte resultado:

$$\frac{\gamma^{n+1}}{\gamma^n} = \frac{1 - 2\psi \sin^2(ph_x/2)}{1 + 2\psi \sin^2(ph_x/2)} \cdot \frac{1 - 2\psi \sin^2(qh_y/2)}{1 + 2\psi \sin^2(qh_y/2)}, \tag{5.52}$$

onde $|\gamma| < 1$ para para qualquer Δt, p e q que sejam escolhidos.

Portanto, o uso alternado das duas equações, uma tomando x implicitamente e a outra y implicitamente, em um esquema de quebra do passo de tempo, ou seja, o uso de um passo intermediário, resulta na estabilidade matemática incondicional do método ADI, *para o caso bidimensional*.

5.1.3 Análise de acurácia

O estudo da acurácia do método pode ser feita pela eliminação do termo $T_{j,i}^{n+\frac{1}{2}}$ das equações (5.31) e (5.32), deste modo obtendo uma equação média.

Somando e subtraindo, respectivamente, (5.31) e (5.32) e reorganizando-se os termos obtem-se:

$$2\frac{\rho C_p}{k\Delta t}\left(T_{j,i}^{n+1}-T_{j,i}^{n}\right)=2\left(\frac{T_{j,i+1}^{n+\frac{1}{2}}-2T_{j,i}^{n+\frac{1}{2}}+T_{j,i-1}^{n+\frac{1}{2}}}{h_x^2}\right) + \frac{\left(T_{j+1,i}^{n+1}+T_{j+1,i}^{n}\right)-2\left(T_{j,i}^{n+1}+T_{j,i}^{n}\right)+\left(T_{j-1,i}^{n+1}+T_{j-1,i}^{n}\right)}{h_y^2} \quad \text{e} \tag{5.53}$$

$$\frac{\rho C_p}{k\Delta t}\left(2T_{j,i}^{n+\frac{1}{2}}-T_{j,i}^{n}-T_{j,i}^{n+1}\right)= \frac{\left(T_{j+1,i}^{n}-T_{j+1,i}^{n+1}\right)-2\left(T_{j,i}^{n}-T_{j,i}^{n+1}\right)+\left(T_{j-1,i}^{n}-T_{j-1,i}^{n+1}\right)}{h_y^2}. \tag{5.54}$$

Resolvendo (5.53) para $T_{j,i}^{n+\frac{1}{2}}$ e substituindo em (5.54) chega-se a:

$$\begin{aligned} T_{j,i}^{n+1} = T_{j,i}^{n} &+ \frac{k\Delta t}{\rho C_p}\left[\frac{T_{j,i+1}^{n+\frac{1}{2}}-\left(T_{j,i}^{n+1}+T_{j,i}^{n}\right)+T_{j,i-1}^{n+\frac{1}{2}}}{h_x^2}\right]+ \\ &\frac{1}{2}\frac{k\Delta t}{\rho C_p}\left[\frac{\left(T_{j+1,i}^{n+1}+T_{j+1,i}^{n}\right)-\left(T_{j,i}^{n+1}+T_{j,i}^{n}\right)+\left(T_{j-1,i}^{n+1}+T_{j-1,i}^{n}\right)}{h_y^2}\right]+ \\ &\frac{1}{4}\frac{k^2\Delta t^2}{\rho^2 C_p^2}\left[\frac{\left(T_{j+1,i}^{n}-T_{j+1,i}^{n+1}\right)-2\left(T_{j,i}^{n}-T_{j,i}^{n+1}\right)+\left(T_{j-1,i}^{n}-T_{j-1,i}^{n+1}\right)}{h_y^2 h_x^2}\right]. \end{aligned} \tag{5.55}$$

Porém, note que pode-se aproximar $T_{j,i}^{n+\frac{1}{2}}$ por

$$T_{j,i}^{n+\frac{1}{2}}=\frac{T_{j,i}^{n+1}+T_{j,i}^{n}}{2} \tag{5.56}$$

assim, (5.55) fica

$$T_{j,i}^{n+1} = T_{j,i}^{n} + \frac{k\Delta t}{\rho C_p}\left[\frac{T_{j,i+1}^{n+\frac{1}{2}} - 2T_{j,i}^{n+\frac{1}{2}} + T_{j,i-1}^{n+\frac{1}{2}}}{h_x^2}\right] + \frac{1}{2}\frac{k\Delta t}{\rho C_p}\left[\frac{\left(T_{j+1,i}^{n+1} + T_{j+1,i}^{n}\right) - 2\left(T_{j,i}^{n+1} + T_{j,i}^{n}\right) + \left(T_{j-1,i}^{n+1} + T_{j-1,i}^{n}\right)}{h_y^2}\right] + \frac{1}{4}\frac{k^2\Delta t^2}{\rho^2 C_p^2}\left[\frac{\left(T_{j+1,i}^{n} - T_{j+1,i}^{n+1}\right) - 2\left(T_{j,i}^{n} - T_{j,i}^{n+1}\right) + \left(T_{j-1,i}^{n} - T_{j-1,i}^{n+1}\right)}{h_y^2 h_x^2}\right], \quad (5.57)$$

que é uma perturbação da equação de Crank-Nicolson [33]. Deste modo o método ADI apresenta erro da $O\left(h_x^2 + h_y^2 + \Delta t^2\right)$.

5.1.4 Análise de convergência

A análise de convergência consiste na obtenção do grau de convergência p do método. Entende-se aqui por p como a velocidade com a qual uma sucessão converge a seu limite. Este conceito é, do ponto de vista prático, muito importante ao trabalhar com sequências de sucessivas aproximações, o qual é o caso da resolução numérica de EDPs.

Considere a equação (5.29). Seja m e M, respectivamente, o menor e maior valor de θ em uma região limitada σ. Se $m \leq \theta_{j,i}^{n} \leq M \ \forall \ j, i$, então:

$$m \leq \theta_{j,i}^{n+\frac{1}{2}} \leq M \quad (5.58)$$

desde que $(2 - 2r_j) \geq 0$. Ou seja, se $\|\theta_{j,i}^{n}\| = M$, então:

$$\|\theta_{j,i}^{n+\frac{1}{2}}\| \leq \|\theta_{j,i}^{n}\|. \quad (5.59)$$

Por analogia, a partir da equação (5.30),

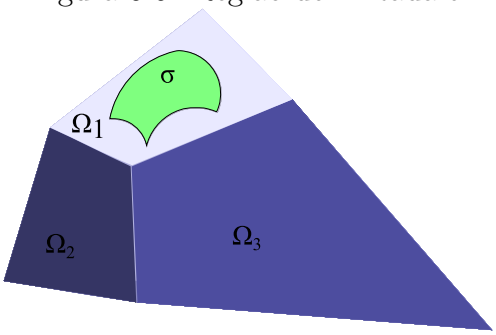
$$\|\theta_{j,i}^{n+1}\| \leq \|\theta_{j,i}^{n+\frac{1}{2}}\|, \quad (5.60)$$

se $\|\theta_{j,i}^{n+\frac{1}{2}}\| = M$.

Note que (5.29) pode ser escrita como:

$$\theta_{j,i}^{n+\frac{1}{2}} = A^{-1}B\theta_{j,i}^{n} \quad (5.61)$$

Figura 5.6: Região delimitada σ



e (5.30) como:

$$\theta_{j,i}^{n+1} = C^{-1}D\theta_{j,i}^{n+\frac{1}{2}} = C^{-1}DA^{-1}B\theta_{j,i}^{n} \tag{5.62}$$

onde A, B, C e D são matrizes de coeficientes.

Seja agora u e U, respectivamente, a solução exata e numérica do caso bidimensional de (5.16). O erro numérico pode ser definido como:

$$e = U - u. \tag{5.63}$$

Assim (5.61) e (5.62) podem ser escritas em termos dos erros numéricos do seguinte modo:

$$e_{j,i}^{n+\frac{1}{2}} = A^{-1}Be_{j,i}^{n} + \frac{\Delta t}{2}O\left(\Delta t^2 + h_x^2 + h_y^2\right) \quad \text{e,} \tag{5.64}$$

$$\begin{aligned} e_{j,i}^{n+1} &= C^{-1}De_{j,i}^{n+\frac{1}{2}} + \frac{\Delta t}{2}O\left(\Delta t^2 + h_x^2 + h_y^2\right) \\ &= C^{-1}D\left[A^{-1}Be_{j,i}^{n} + \frac{\Delta t}{2}O\left(\Delta t^2 + h_x^2 + h_y^2\right)\right] \\ &+ \frac{\Delta t}{2}O\left(\Delta t^2 + h_x^2 + h_y^2\right). \end{aligned} \tag{5.65}$$

Desprezando os termos de ordem superior da equação (5.65) e fazendo

$$M = C^{-1}DA^{-1}B$$

tem-se:

$$e_{j,i}^{n+1} = Me_{j,i}^{n} + \frac{\Delta t}{2}\underbrace{O\left(\Delta t^2 + h_x^2 + h_y^2\right)}_{\nu_{n+\frac{1}{2}}} \tag{5.66}$$

em que $\nu_k = O\left(\Delta t^2 + h_x^2 + h_y^2\right)$ é o erro no nível k. Logo,

$$e_{j,i}^{n+1} = M^n e_{j,i}^{0} + \left[M_{n+\frac{1}{2}}\nu_1 + M_n\nu_2 + \ldots + M\nu_{n-1}\right]\frac{\Delta t}{2}. \tag{5.67}$$

Se

$$\|M^k u\| \leq C\|u\| \quad \forall \quad k,$$

então:

$$\|e_{j,i}^{n+1}\| = C\left(\|e_{j,i}^{0}\| + [\|\nu_1\| + \|\nu_2\| + \ldots + \|\nu_{n-1}\|]\,\frac{\Delta t}{2}\right) \tag{5.68}$$

onde $C \geq 0$ é uma constante que independe da discretização.

Assim, pelo resultado obtido em (5.68), o método ADI converge com a mesma precisão do esquema numérico, ou seja, converge com ordem $p = 2$. Assim, para problemas lineares, convergência confunde-se com a estabilidade do método.

5.2 Tridiagonal matrix algorithm – TDMA

O método de Thomas ou TDMA é o mais conhecido método linha a linha disponível. Como se pode deduzir, os métodos linha a linha resolvem diretamente uma linha, ou seja, um problema unidimensional e, deste modo, sendo um candidato natural a ser selecionado para aplicação em conjunto com o método ADI.

Considere um sistema linear com N equações algébricas, que consiste de N incógnitas, $u_1, u_2, u_3, \ldots, u_N$, dado na forma abaixo.

$$\begin{aligned}
d_1u_1 + a_1u_2 &= c_1 && (5.69)\\
b_2u_1 + d_2u_2 + a_2u_3 &= c_2 && (5.70)\\
b_3u_2 + d_3u_3 + a_3u_4 &= c_3 && (5.71)\\
\vdots \quad \vdots \quad \vdots &\quad \vdots &&\\
b_{N-1}u_{N-2} + d_{N-1}u_{N-1} + a_{N-1}u_N &= c_{N-1} && (5.72)\\
b_Nu_{N-1} + d_Nu_N &= c_N && (5.73)
\end{aligned}$$

Este é um sistema tridiagonal, ou seja, é um sistema de equações que consiste apenas da diagonal principal (d_i's), da subdiagonal inferior (b_i's) e da subdiagonal superior (a_i's).

Um método padrão para resolver um sistema linear de equações algébricas é a aplicação da eliminação gaussiana. O TDMA é, então, essencialmente o resultado da aplicação desta em um sistema tridiagonal de equações. Especificamente, a idéia do método é o de se eliminar os termos da diagonal inferior (b_i's) e assim obter uma relação direta de substituição como se segue.

Multiplique (5.69) por b_2.

$$b_2 d_1 u_1 + b_2 a_1 u_2 = c_1 b_2 \tag{5.74}$$

Agora multiplique (5.70) por d_1.

$$d_1 b_2 u_1 + d_1 d_2 u_2 + d_1 a_2 u_3 = c_2 d_1 \tag{5.75}$$

Subtraia (5.74) de (5.75)

$$(d_1 d_2 - b_2 a_1)\, u_2 + d_1 a_2 u_3 = c_2 d_1 - c_1 b_2 \tag{5.76}$$

Divida (5.76) por d_1.

$$\left(d_2 - \frac{b_2 a_1}{d_1}\right) u_2 + a_2 u_3 = c_2 - \frac{c_1 b_2}{d_1} \tag{5.77}$$

Note que (5.77) não mais apresenta o termo referente a sua diagonal inferior. Seja:

$$d'_2 = d_2 - \frac{b_2 a_1}{d_1} \quad \text{e,} \tag{5.78}$$

$$c'_2 = c_2 - \frac{c_1 b_2}{d_1}, \tag{5.79}$$

pode-se reescrever (5.77) como:

$$d'_2 u_2 + a_2 u_3 = c'_2. \tag{5.80}$$

Continuando o processo de eliminação, multiplica-se (5.80) por b_3.

$$b_3 d'_2 u_2 + b_3 a_2 u_3 = b_3 c'_2. \tag{5.81}$$

Multiplique (5.71) por d'_2.

$$d'_2 b_3 u_2 + d'_2 d_3 a_2 u_3 + d'_2 a_3 u_4 = d'_2 c_3 \tag{5.82}$$

Subtraia (5.81) de (5.82).

$$(d'_2 d_3 - b_3 a_2)\, u_3 + d'_2 a_3 u_4 = d'_2 c_3 - b_3 c'_2 \tag{5.83}$$

Divida (5.83) por d'_2.

$$\left(d_3 - \frac{b_3 a_2}{d'_2}\right) u_3 + a_3 u_4 = c_3 - \frac{b_3 c'_2}{d'_2} \tag{5.84}$$

Observe que (5.84) não mais apresenta o seu termo da diagonal inferior; este termo foi novamente eliminado da mesma maneira que para o casa do equação (5.77).

Neste momento é possível notar que há um padrão se desenvolvendo. A equação (5.77) pode ser vista como obtida da equação (5.70) pela:

1. eliminação do primeiro termo (o termo envolvendo u_1),
2. substituição de sua diagonal principal por

$$d_2 - \frac{b_2 a_1}{d_1} \tag{5.85}$$

   no lugar de d_2,
3. não alteração do terceiro termo ($a_2 u_3$),
4. substituição do termo no lado direito da equação por

$$c_2 - \frac{c_1 b_2}{d_1} \tag{5.86}$$

   no lugar de c_2.

Comparando as equações (5.84) e (5.71) é possível encontrar exatamente o mesmo padrão, onde na equação (5.71) o primeiro termo ($b_3 u_2$) é eliminado, o coeficiente da diagonal principal é substituido por

$$d_3 - \frac{b_3 a_2}{d'_2} \tag{5.87}$$

o terceiro termo permanece o mesmo ($a_3 u_4$), e o termo no lado direito é substituido por

$$c_3 - \frac{c'_2 b_3}{d'_2}. \tag{5.88}$$

A comparação das formas apresentadas por (5.85) e (5.87) permite chegar a conclusão de que são as mesmas. O mesmo pode ser feito para (5.86) e (5.88), chegando-se a conclusão idêntica. Assim, o padrão torna-se claro e, deste modo, é possível generalizar o procedimento.

Começando do topo do sistema linear representado pelas equações (5.69) a (5.73), deixa-se (5.69) conforme está e, para as equações seguintes, elimina-se o primeiro termo, substitui-se o coeficiente da diagonal principal por

$$d'_i = d_i - \frac{b_i a_{i-1}}{d'_{i-1}} \qquad i = 2, 3, \ldots, N \tag{5.89}$$

e substituiem-se os termos do lado direito das mesmas por

$$c'_i = c_i - \frac{c'_{i-1} b_i}{d'_{i-1}} \qquad i = 2, 3, \ldots, N \tag{5.90}$$

resultando em um sistema bidiagonal triangular superior de equações dado por

$$\begin{aligned}
&d_1 u_1 + a_1 u_2 && = c_1 \\
&d'_2 u_2 + a_2 u_3 && = c'_2 \\
&d'_3 u_3 + a_3 u_4 && = c'_3 \\
&\vdots \qquad \vdots && \vdots \\
&d'_{N-1} u_{N-1} + a_{N-1} u_N && = c'_{N-1} \qquad (5.91) \\
&d'_N u_N && = c'_N \qquad (5.92)
\end{aligned}$$

Examinando o sistema de equações acima é possível notar que a última das equações , (5.92), contém apenas uma incógnita, u_N; assim:

$$u_N = \frac{c'_N}{d'_N}. \tag{5.93}$$

A solução das incógnitas restantes é obtida mediante a retro-substituição no sistema acima. Por exemplo, depois de obtido u_N de (5.93), o valor de u_{N-1} pode ser encontrado da equação (5.91) como:

$$u_{N-1} = \frac{c'_{N-1} - a_{N-1} u_N}{d'_{N-1}}. \tag{5.94}$$

Por inspeção, (5.95) pode ser substituida por uma relação recursiva generalizada dada por

$$u_i = \frac{c'_i - a_i u_{i+1}}{d'_i}. \tag{5.95}$$

para o cálculo de u_i, onde u_{i+1} é conhecido da aplicação anterior da equação (5.95). Assim, todas as incógnitas são encontradas em sequência, começando de $u_i = u_{N-1}$ e terminando com $u_i = u_1$.

Capítulo 6

Avaliação da Maturação da Matéria Orgânica

A transformação e maturação da matéria orgânica pode ser dividida em quatro grandes fases: diagênese, catagênese, metagênese e metamorfismo [5].

Durante a diagênese, a maior parte da matéria orgânica é transformada por meio de processos biológicos em querogênio, com a liberação das frações mais voláteis, tais como CH_4, NH_3 e CO_2. A atividade bacteriana provoca a reorganização celular, levando à formação de metano bioquímico e biogênico.

É durante a catagênese que a maior parte do petróleo é gerado, justamente por meio do craqueamento térmico do querogênio em hidrocarbonetos leves e pesados. A taxa de transformação depende essencialmente do tipo de matéria orgânica e da história térmica da bacia. Frações mais pesadas de petróleo são em geral geradas em um primeiro momento e, à medida que a temperatura aumenta, são gradativamente reduzidas a frações mais leves.

A continuidade do processo de aquecimento, avançando até em torno de 210°C, propicia a quebra do querogênio e sua transformação em gás leve. É a chamada metagênese.

Por fim, ultrapassada a fase anterior, a continuação do incremento de temperatura leva à degradação do hidrocarboneto gerado, deixando como remanescente, a grafita, gás carbônico e algum resíduo de gás metano, caracterizando, então, a fase de metamorfismo.

Tabela 6.1: Evolução da matéria orgânica [5]

Estágio	%Ro	Nível de Maturação
diagênese	$< 0,5$	imaturo
catagênese	$0,5 - 1,35$	zona de óleo
catagênese	$1,35 - 2,00$	matura (zona regressiva)
catagênese	$2,00 - 3,00$	zona de gás
metaênese	$> 2,0$	senil (zona de gás seco)

Um modo de quantificar essas fases e, deste modo, o grau de maturação da matéria orgânica faz-se por meio da utilização de modelos de cinética química. Tais formulações são baseadas em balanços de massa. Um esquema amplamente utilizado para o fim aqui proposto é o de fazer uso de modelos baseados na cinética de reatividades distribuidas. A idéia desse tipo de formulação é a de replicar mais fielmente a natureza do processo em questão, já que leva em consideração o fato da ocorrência de reações químicas em paralelo. Modelos desse tipo são baseados na lei de Arrhenius e apresentam em geral a forma:

$$k = Ae^{-E/RT} \tag{6.1}$$

onde k é a taxa de reação, $A = 1,0$ x $10^{13}/s$ um fator de frequência, T é a temperatura em Kelvin, E a energia de ativação e $R = 8.31447Ws/mol/K$ é a constante universal dos gases. A relação acima demonstra a forte dependência com a temperatura. O fator de frequência representa a frequência na qual as moléculas serão transformadas e a energia de ativação descreve o limiar de energia necessário para que a reação inicie espontaneamente. A lei de Arrhenius foi inicialmente desenvolvida como uma equação empírica, todavia posteriormente veio a ser confimada teoricamente [34].

Dentre as diversas opções de modelos para essa classe de métodos, optou-se por utilizar neste presente trabalho o modelo Easy%Ro desenvolvido por SWEENEY e BURNHAM [2]. Este modelo avalia o grau de maturação da matéria orgânica por meio de uma associação ao grau de reflectância da vitrinita.

A reflectância da vitrinita é um dos principais indicadores de maturação da matéria orgânica. Ela aumenta como uma função da temperatura e tempo de aproximadamente $R_o = 0{,}25\%$ até valor superiores a $R_o = 4{,}00\%$, sendo valores acima deste já um estado de meta-antracito, não sendo de interesse prático.

A vitrinita é um constituinte orgânico dos carvões e sedimentos, resultado da matéria orgânica massareal, que permite detectar o estágio de maturação dos sedimentos, sendo utilizada como parâmetro básico para as deteminações paleotermométricas. A vitrinita compreende dois tipos de substâncias: telinita (material da parede das células de plantas terrestres) colinita (substancia que preenche a cavidade das células). A refletância da luz na superficie polida desses dois submacerais é aproximadamente a mesma. Ela aumenta com a maturação, em razão do aumento da temperatura, que provoca alteração irreversível na estrutura molecular de ambos. Consequentemente, a refletância mede as mudanças de maturação provocadas pelo aumento progressivo da temperatura. Os valores de vitrinita indicam somente o nível de maturação da matéria orgânica examinada, não sendo diagnos-

tico da presença de óleo ou gás, devido aos fenômenos de migração e da qualidade da matéria orgânica presente no sedimento. A reação de governo geral assume que a vitrinita é transformada em uma forma residual (modificada ou maturada) e algum condensado.

$$\text{Vitrinita} \longrightarrow \text{Vitrinita Residual} + \text{Voláteis} \tag{6.2}$$

6.1 Easy%Ro

Introdução

Este modelo usa uma distribuição de energias de ativação para cada uma das quatro reações consideradas:

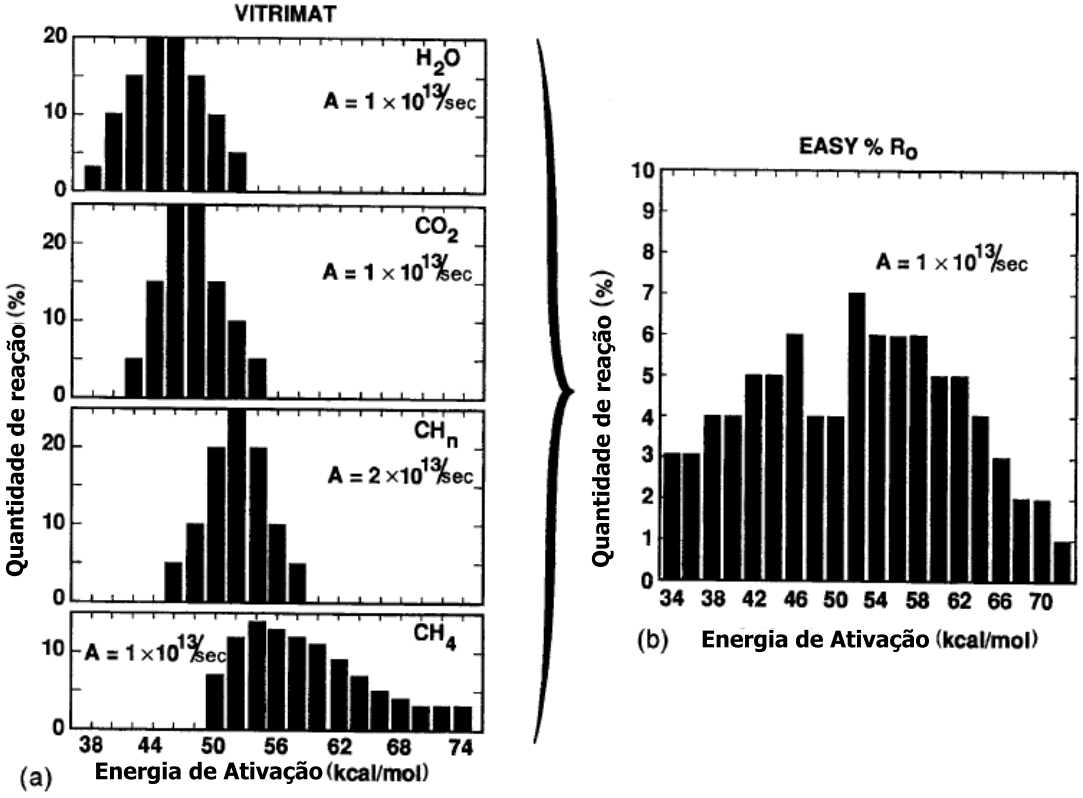
- eliminação de água;
- eliminação de dióxido de carbono;
- eliminação de metano;
- eliminação de hidrocarbonetos pesados;

Cada uma dessas reações é descrita como uma reação paralela de decomposição, que apresenta um conjunto discreto de energias de ativação

$$\begin{aligned}
\text{vitrinita} &\xrightarrow{k_{1i}} \text{vitrinita residual} + \mathrm{H_2O} \\
\text{vitrinita} &\xrightarrow{k_{2i}} \text{vitrinita residual} + \mathrm{CO_2} \\
\text{vitrinita} &\xrightarrow{k_{3i}} \text{vitrinita residual} + \mathrm{CH}_n \\
\text{vitrinita} &\xrightarrow{k_{4i}} \text{vitrinita residual} + \mathrm{CH_4}
\end{aligned} \tag{6.3}$$

A reflectância da vitrinita é então calculada com as quatro taxas de transformação das reações acima. Para o modelo Easy%Ro há ainda mais uma simplificação por meio da aproximação dessa cinética química através da superposição das quatro reações utilizando o mesmo fator de frequência.

Figura 6.1: Distribuição de energias de ativação usadas no modelo Easy%Ro [2]



Antes de iniciar propriamente a apresentação matemática do modelo faz-se necessário destacar alguns pontos acerca do mesmo:

- o modelo apresenta valores válidos para a faixa de %Ro = 0,3 – 4,5%;
- o modelo é calibrado para um modelo químico mais completo de maturação da vitrinita no qual as razões atômicas H/C e O/C são correlacionadas a uma média máxima e a valores médios aleatórios de reflectância medidos;
- pode ser aplicado a situações que apresentam taxas de aquecimentos entre a faixa de 10°C/semana – 1°C/k.a.[6];
- permite a sua utilização em conjunto com qualquer tipo de história térmica.

Modelo químico-matemático

Reações químicas podem ser descritas por uma reação de Arrhenius de primeira ordem,

$$\frac{dw}{dt} = -kw \tag{6.4}$$

[6]k.a. = 1000 anos.

onde k é dado por (6.1) e w é a quantidade não reagida de um componente.

Nesta abordagem, a natureza heterogênea da matéria é levada em conta assumindo-se que uma complexa reação é melhor representada por um conjunto de reações paralelas com o mesmo A, mas diferentes Es.

Para a reação do i$_{\text{ésimo}}$ componente sujeito a uma temperatura $T(t)$,

$$\frac{dw_i}{dt} = -w_i A e^{-E_i/RT(t)}, \tag{6.5}$$

e

$$\frac{dw}{dt} = \sum_i \frac{dw_i}{dt}. \tag{6.6}$$

A quantidade de matéria orgânica não convertida que resta do i$_{\text{ésimo}}$ componente é descrita por

$$w_i = w_{0,i} - \int_0^t [dw_i/dt] \, \mathrm{d}t, \tag{6.7}$$

e a fração convertida de reagente é dada por:

$$F = 1 - \frac{w}{w_0} = 1 - \sum_i f_i \frac{w_i}{w_{0i}}, \tag{6.8}$$

onde w_0 é a concentração inicial total de reagente, w_{0i} é a concentração inicial do i$_{\text{ésimo}}$ componente e f_i é o fator estequiométrico.

F pode ser calculado através da quebra da história térmica em uma série de segmentos com taxas de aquecimento constantes. Define-se a taxa de aquecimento entre os níveis de tempo j e $j-1$ como:

$$H_j = \frac{T_j - T_{j-1}}{t_j - t_{j-1}}. \tag{6.9}$$

A extensão da reação do i$_{\text{ésimo}}$ componente no tempo j é então dado por:

$$w_i/w_{0,i} = 1 - e^{-\Delta I_{i,j}}, \tag{6.10}$$

onde

$$\Delta I_{i,j} = \left(I_{i,j} - I_{i,j-1}\right)/H_{i,j}, \tag{6.11}$$

e

$$I_{i,j} = T_j A e^{-E_i/RT_j} \left[1 - \frac{(E_i/RT_j)^2 + a_1 (E_i/RT_j) + a_2}{(E_i/RT_j)^2 + b_1 (E_i/RT_j) + b_2}\right], \tag{6.12}$$

em que $a_1 = 2,334733$, $a_2 = 0,250621$, $b_1 = 3,330657$ e $b_2 = 1,681534$.

Por fim, calcula-se o valor de reflectância da vitrinita com:

$$\%Ro = exp\left(-1,6 + 3,7F\right). \tag{6.13}$$

Tabela 6.2: Energias de ativação e fatores estequiométricos utilizados no modelo Easy%Ro [2]

Fator Estequimétrico	Energia de Ativação (kcal/mole)
0,03	34
0,03	36
0,04	38
0,04	40
0,05	42
0,05	44
0,06	46
0,04	48
0,04	50
0,07	52
0,06	54
0,06	56
0,06	58
0,05	60
0,05	62
0,04	64
0,03	66
0,02	68
0,02	70
0,01	72

Em suma, o modelo Easy%Ro utiliza uma única distribuição de reações paralelas com um fator de frequência comum a ambas e uma simples relação exponencial entre a extensão da reação e o valor máximo de reflectância da vitrinita.

6.2 História térmica da bacia sedimentar

A história térmica da bacia sedimentar desempenha importante papel na maturação da matéria orgânica. Conforme visto anteriormente, a transformação desta é fortemente função da temperatura. Assim, é intuitivo supor que diferentes histórias térmicas irão levar a diferentes graus de maturação do querogênio, sendo estas sutilezas facilmente identificáveis nos mapas de reflectância de vitrinita (o qual será elaborado valendo-se do modelo Easy%Ro).

Essa é uma característica importante, pois permite por meio da avaliação dos valores de reflectância da vitrinita identificar qual a história térmica da bacia e, deste modo, calibrar o modelo térmico determinado para a mesma. Por ser considerada paleotermômetro, a vitrinita permite identificar o momento em que a bacia atingiu os níveis necessários para a transformação da matéria orgânica em óleo e gás. Assim, para uma avaliação mais precisa, é necessário considerar os fatores responsáveis pelas variações térmicas na bacia.

As intrusões ígneas, que são o objeto de estudo neste trabalho, tem um papel importante na modificação das condições térmicas da bacia, no momento em que ocorrem. No entanto, a história térmica total da bacia é dada pela adição das mudanças térmicas causadas pela história do soterramento aos efeitos térmicos devido a presença de diques e soleiras.

Assim, a história térmica do soterramento pode ser obtida:

1 através de modelos diretos, que podem quantificar a deformação litosférica por meio de formulações matemáticas, que correlacionam a geração de espaço de acomodação e o fluxo térmico basal, durante o processo evolutivo da bacia. Estas variações térmicas são em geral, dadas em função dos intervalos de subsidência inicial e de resfriamento térmico;

2 através de técnicas de descompactação, que permitem restabelecer a profundidade do embasamento no momento anterior à deposição de cada camada sedimentar, fornecendo assim, um gráfico da geo-história da bacia.

Uma vez determinada a história do fluxo térmico basal, estas poderão ser utilizados como condições de contorno na parte inferior da bacia, para que se possa determinar os gradientes térmicos no interior da mesma, em função da profundidade e das propriedades litológicas de cada camada sedimentar. [5]

Capítulo 7

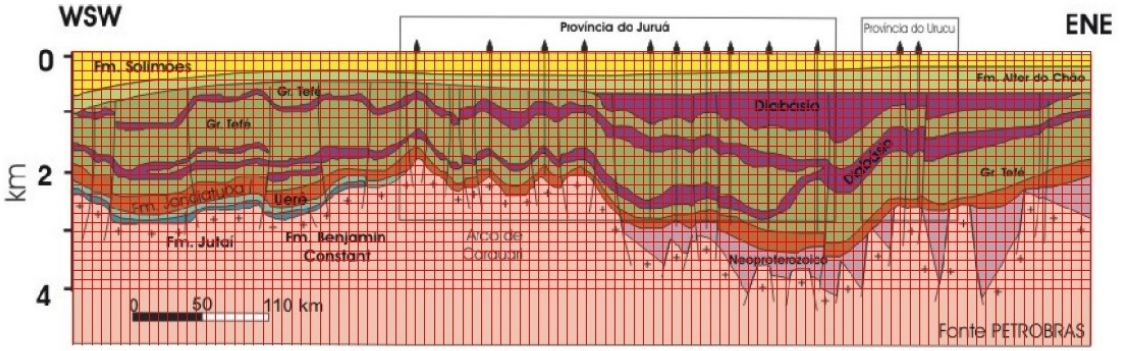
Aplicação

Neste capítulo será apresentada a seção geológica que serviu de base para o estudo aqui realizado e a elaboração da malha utilizada na simulação térmica e no cálculo da reflectância da vitrinita.

Além disso, serão também apresentadas as distribuições das propriedades físicas ao longo do domínio discreto. Optou-se pela utilização de uma malha composta de 2675 blocos (107 x 25).

A seção utilizada é a que segue abaixo:

Figura 7.1: Seção geológica da Bacia do Solimões



Como neste trabalho realiza-se a simulação com base em unidades litológicas, faz-se fundamental realizar um upscaling das propriedades. Para propriedades extensivas optou-se por utilizar a média aritmética da forma

$$\lambda = \sum_{i=1}^{n} p_i \lambda_i \tag{7.1}$$

com

$$\sum_{i=1}^{n} p_i = 1, \tag{7.2}$$

em que λ_i é o valor da propriedade, p_i é a fração de massa e λ é o valor da média.

Por outro lado, as conditividades térmicas são melhor representadas utilizando-se uma variação da média geométrica denominada média geométrica quadrática. Ela é enunciada da seguinte maneira:

$$\sqrt{\lambda} = \sum_{i=1}^{n} p_i \sqrt{\lambda_i}. \tag{7.3}$$

Já as anisotropias, conforme definidas no capítulo 5 são bem representadas pela média geométrica simples:

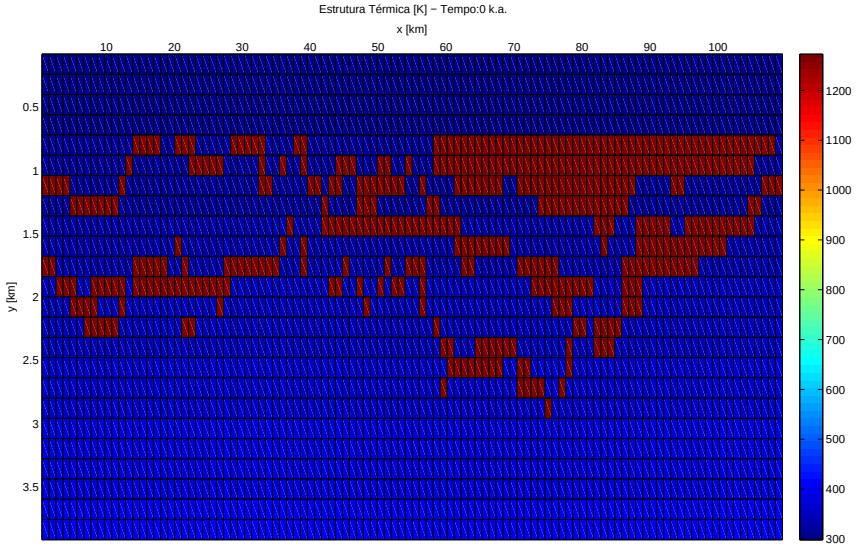
$$\lambda = \prod_{i=1}^{n} \lambda_{ii}^{p}. \tag{7.4}$$

Tabela 7.1: Propriedades das litologias utilizadas [5]

Litologia	$f\,[-]$	$k\,[w/mK]$	$C_p\,[ws/kgK]$	$\rho\,[kg/m^3]$
Arenito	1.15	3.95	855.0	2720.0
Folhelho	1.60	1.64	860.0	2700.0
Calcarenito	1.70	2.18	850.0	2730.0
Granito	1.15	2.18	800.0	2650.0
Magma	1.17	2.10	790.0	2870.0
Diabásio	1.00	2.60	800.0	2800.0
Sal	1.01	6.50	860.0	2740.0

A condição inicial utilizada para a simulação da Bacia do Solimões pode ser vista abaixo. Utilizou-se um gradiente geotérmico de 0.0270833 K/m [14].

Figura 7.2: Distribuição inicial de temperaturas[K]



Por fim, abaixo seguem as distribuições de densidade, condutividade térmica, anisotropia e calor específico, respectivamente.

Figura 7.3: Distribuição de densidades [kg/m^3]

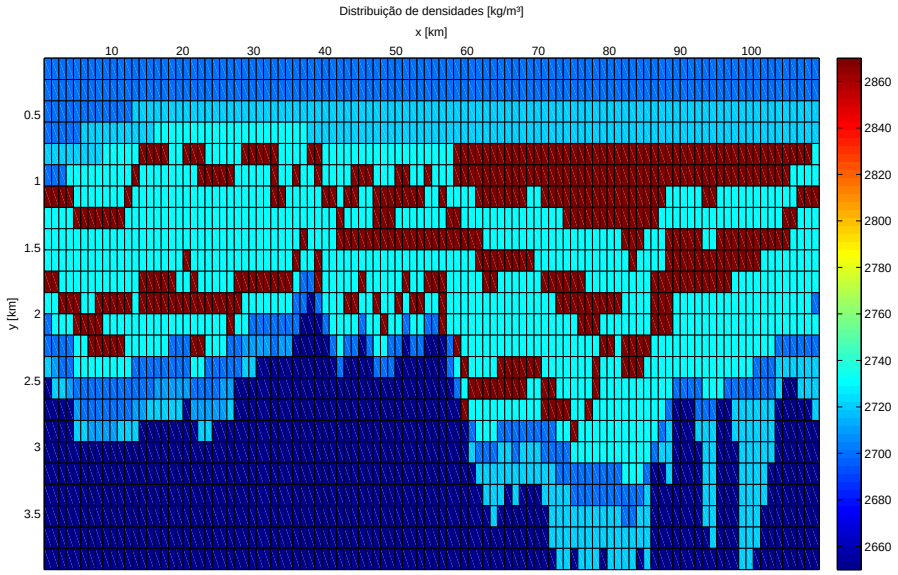


Figura 7.4: Distribuição de condutividades térmicas [w/mK]

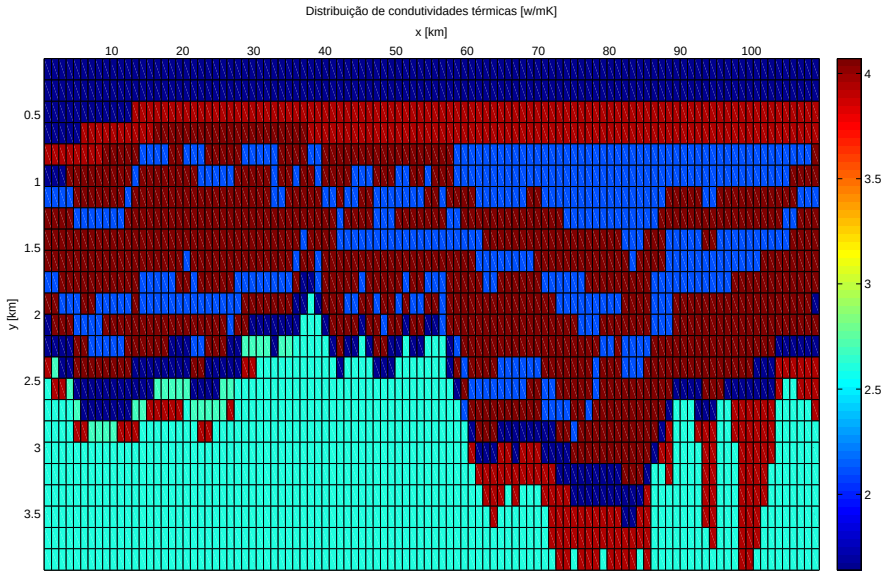


Figura 7.5: Distribuição de anisotropias [-]

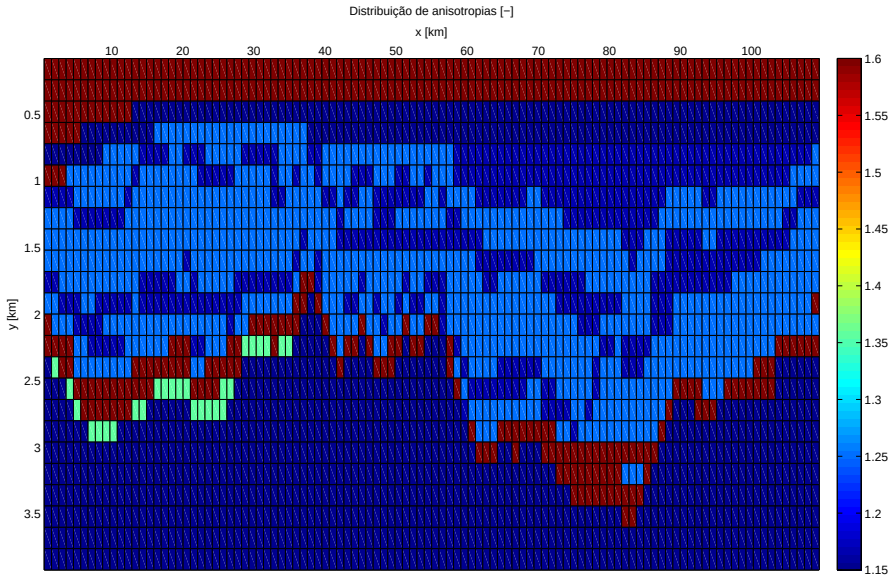
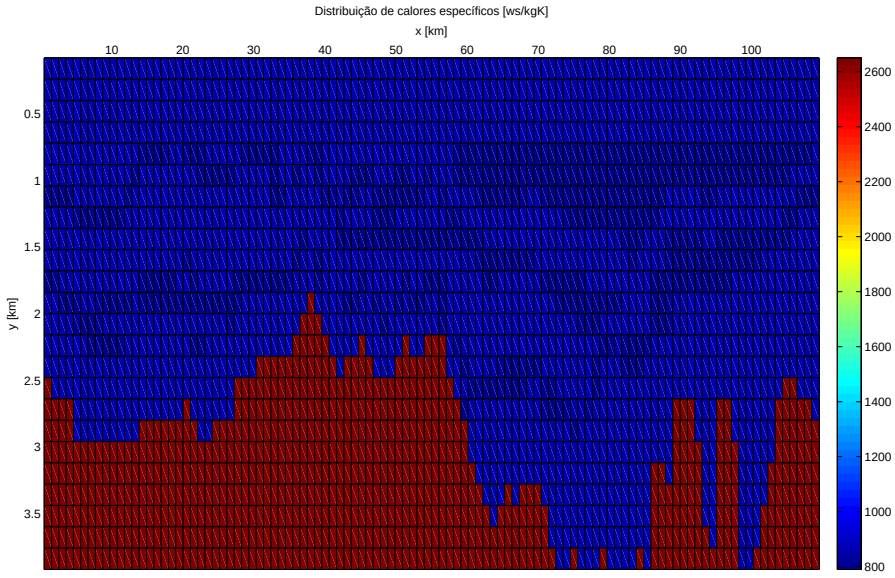


Figura 7.6: Distribuição de calores específicos [ws/kgK]



Capítulo 8

Resultados

8.1 Validação do modelo numérico

Neste tópico serão exibidos resultados que buscam validar os modelos numéricos aqui desenvolvidos. A primeira análise realizada foi baseada em um caso simples, que consiste em uma malha 50 x 50 (2500 blocos) com uma intrusão magmática em seu interior, posicionada simetricamente em relação aos bordos da mesma.

As propriedades térmicas utilizadas foram: as do arenito, para regiões que não as da intrusão, do magma (enquanto este ainda não atingiu o ponto de mudança de fase) e a do diabásio, após a solidificação do mesmo. O objetivo principal é o de se analisar a simetria do problema, conforme pode-se esperar, tanto para o perfil térmico, quanto para o de %Ro.

Serão apresentados quatro diferentes testes, a saber:

- considerando um gradiente térmico;
- desconsiderando a existência de uma gradiente térmico;
- desconsiderando a existência de uma intrusão magmática e gradiente geotérmico;
- comparação entre os dados tabelados de %Ro em [2] com os calculados pelo modelo implementado;

Primeiramente seguem os resultados considerando-se um gradiente térmico :

Figura 8.1: Perfil térmico para t = 0 k.a.

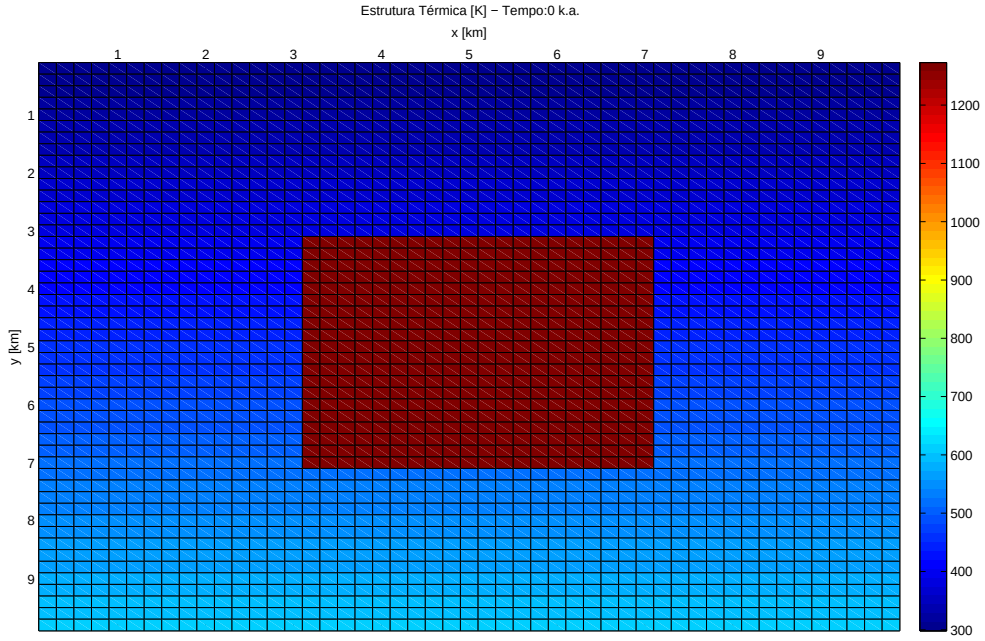


Figura 8.2: %Ro para t = 0 k.a.

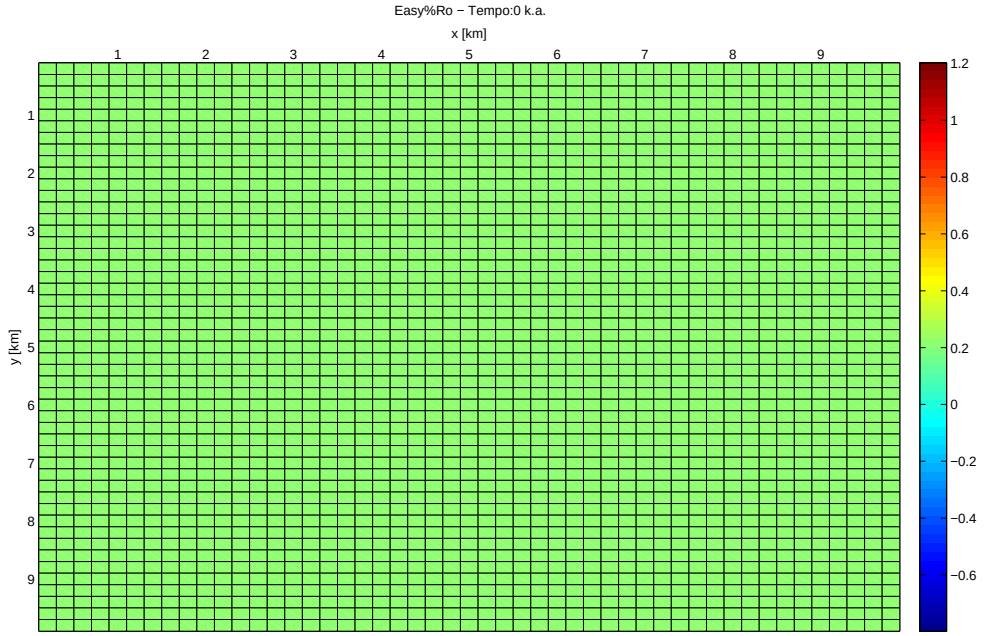


Figura 8.3: Perfil térmico para t = 12,5 k.a.

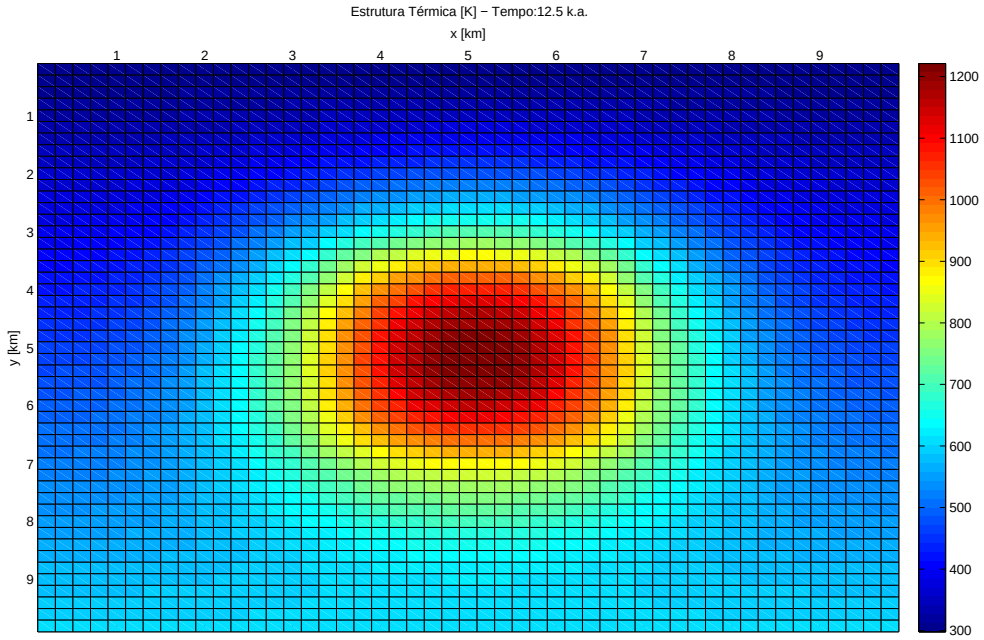


Figura 8.4: %Ro para t = 12,5 k.a.

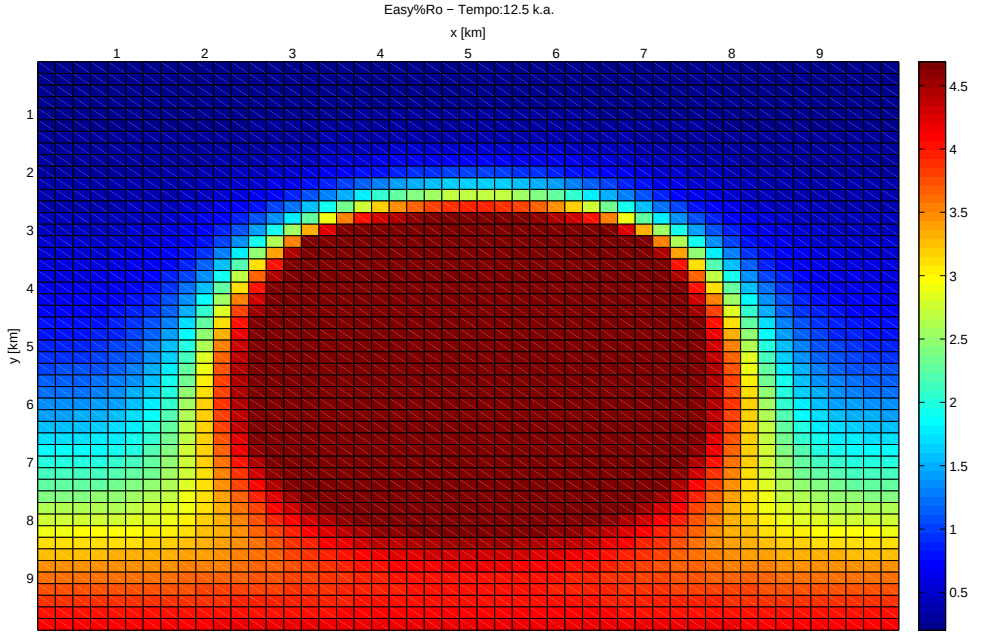


Figura 8.5: Perfil térmico para t = 25 k.a.

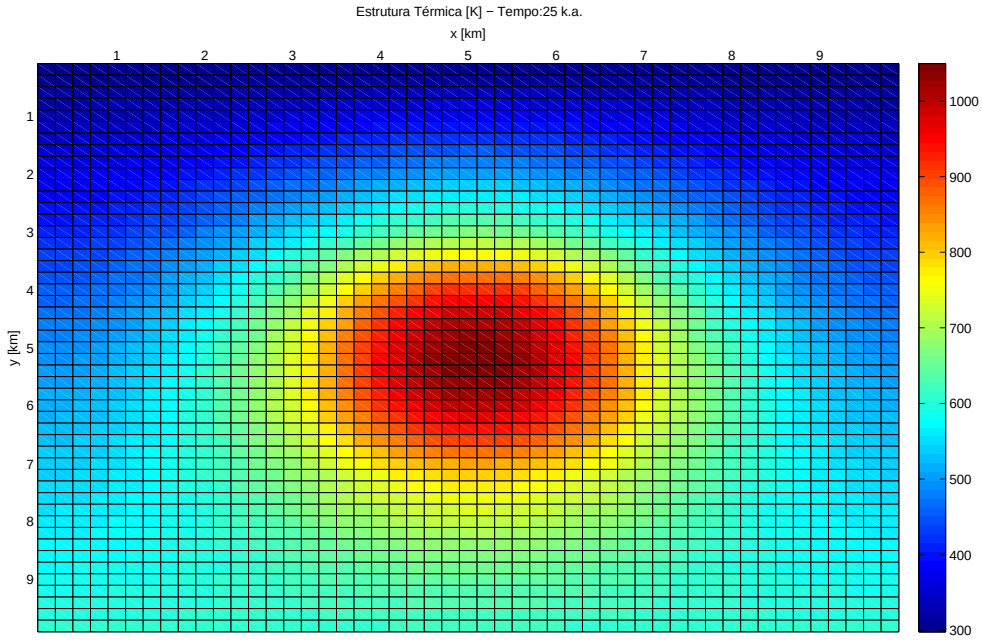
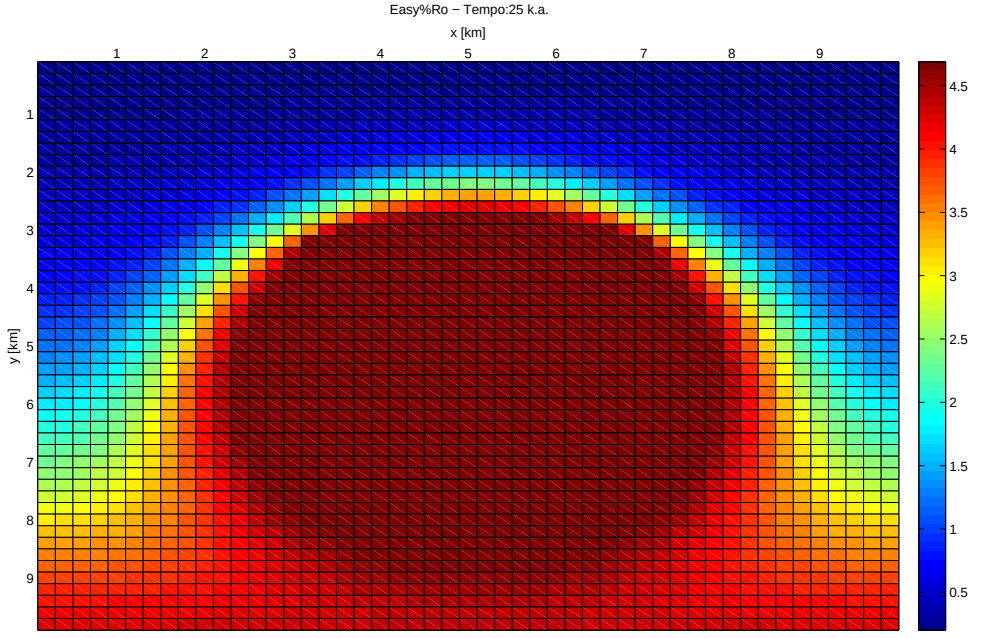


Figura 8.6: %Ro para t = 25 k.a.



Agora desconsiderando-se a existência de uma gradiente térmico:

Figura 8.7: Perfil térmico para t = 0 k.a.

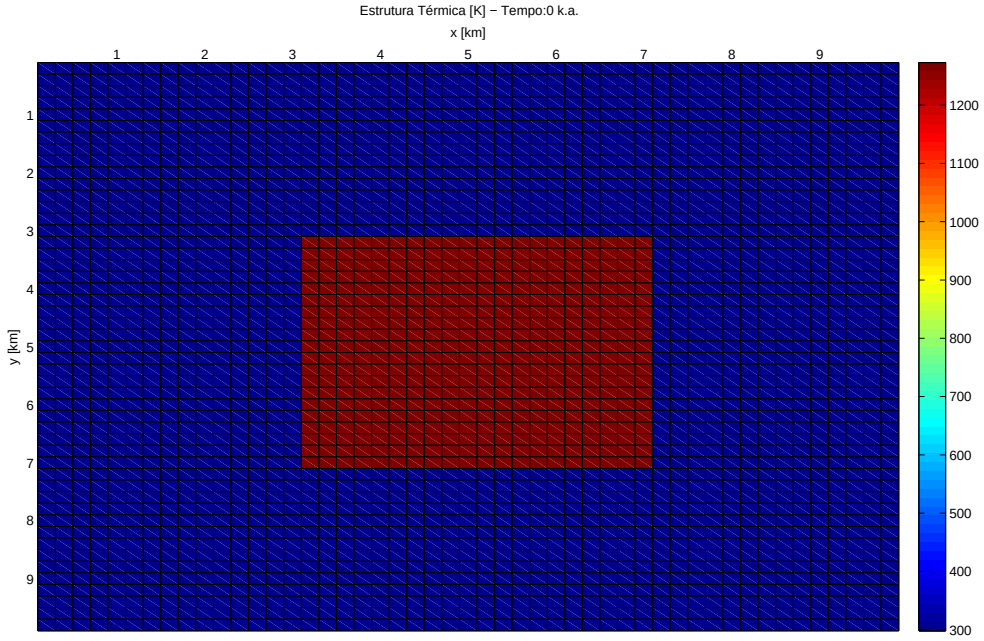


Figura 8.8: %Ro para t = 0 k.a.

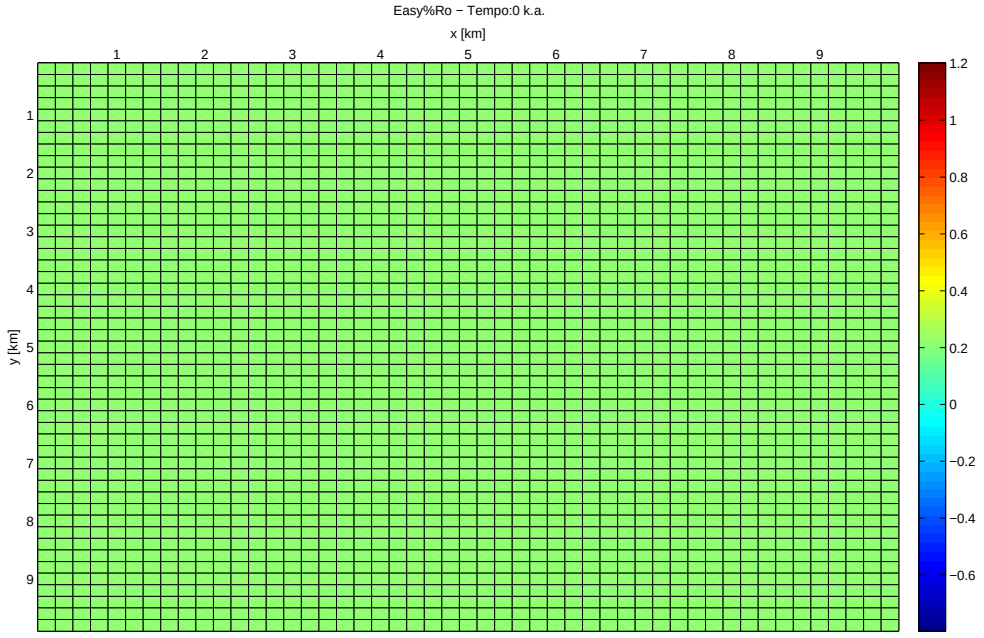


Figura 8.9: Perfil térmico para t = 12,5 k.a.

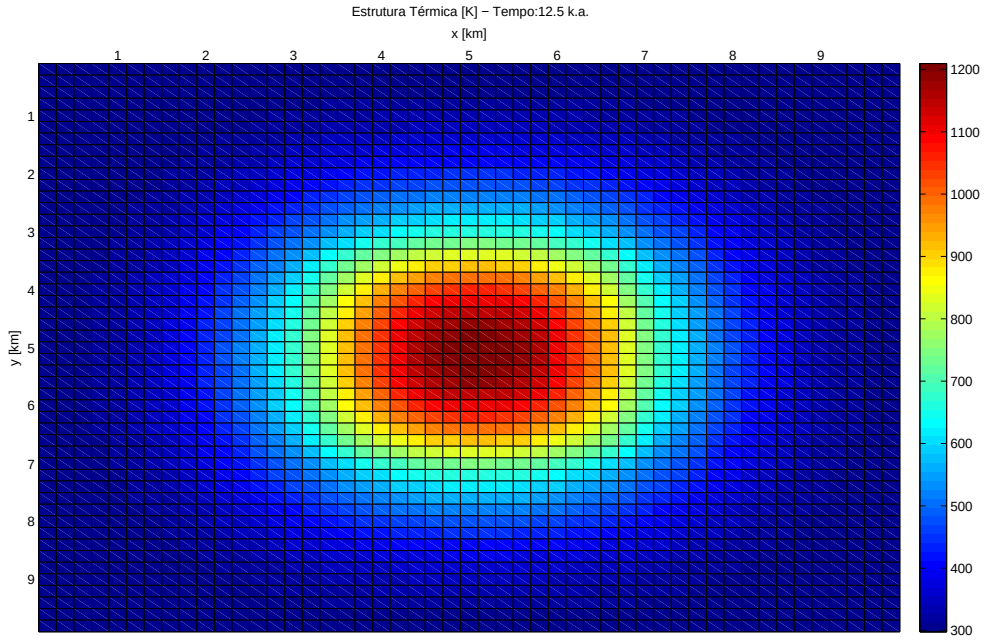


Figura 8.10: %Ro para t = 12,5 k.a.

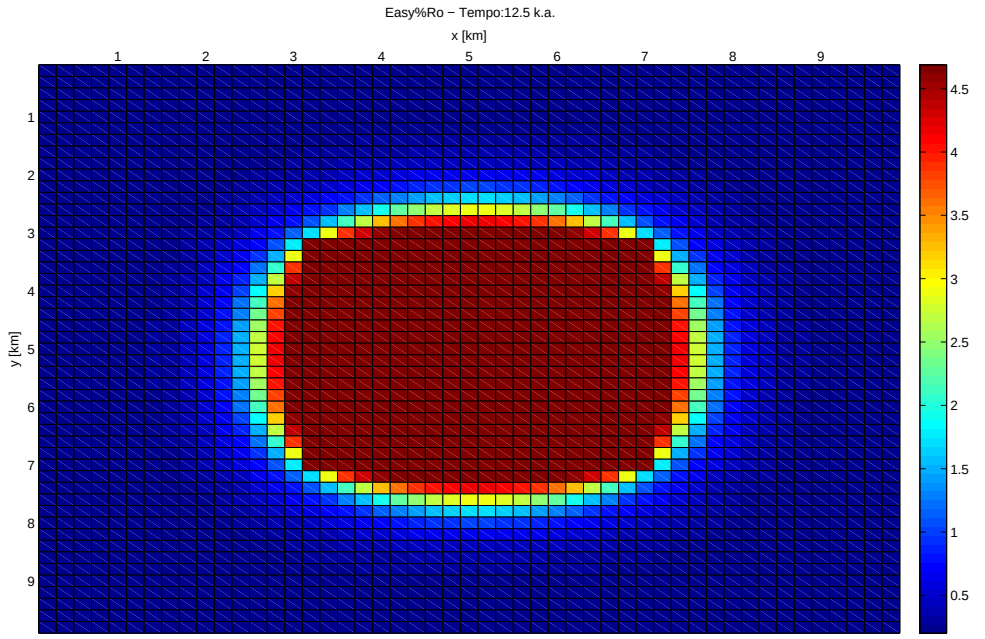


Figura 8.11: Perfil térmico para t = 25 k.a.

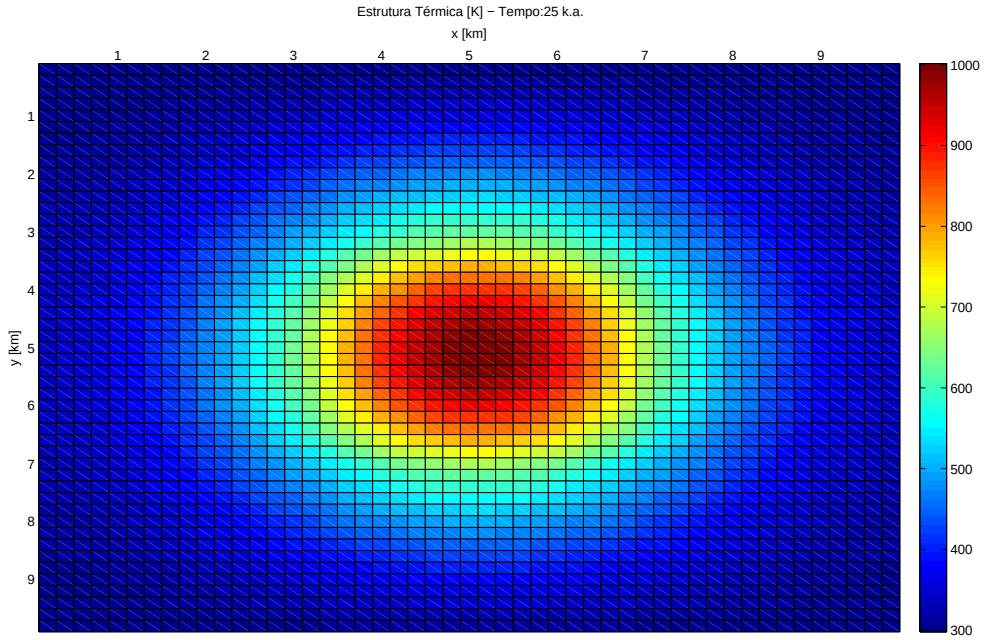
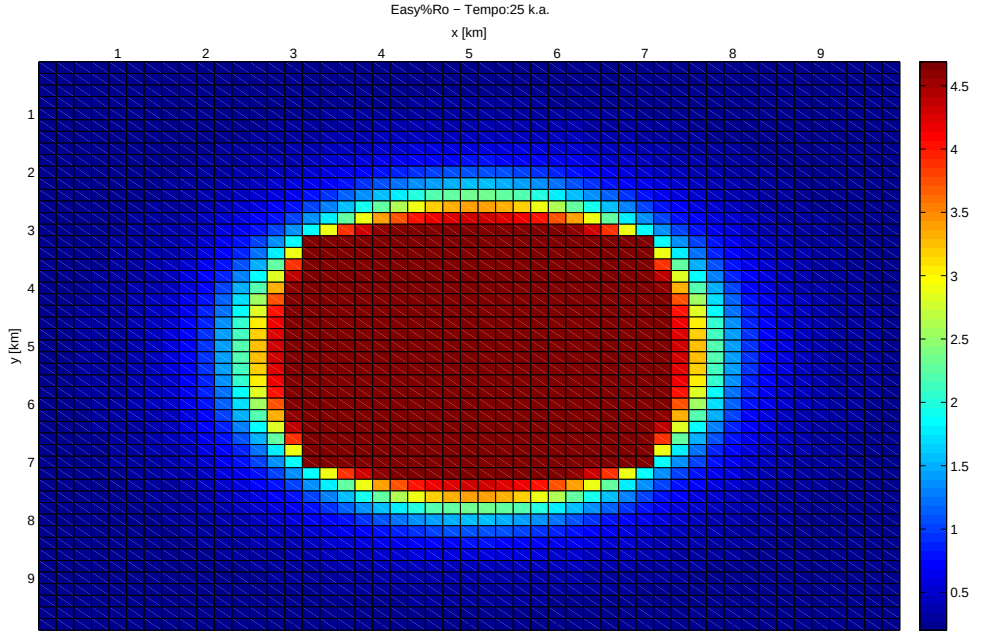


Figura 8.12: %Ro para t = 25 k.a.



Desconsiderando-se a existência de um gradiente térmico e intrusão:

Figura 8.13: Perfil térmico para t = 0 k.a.

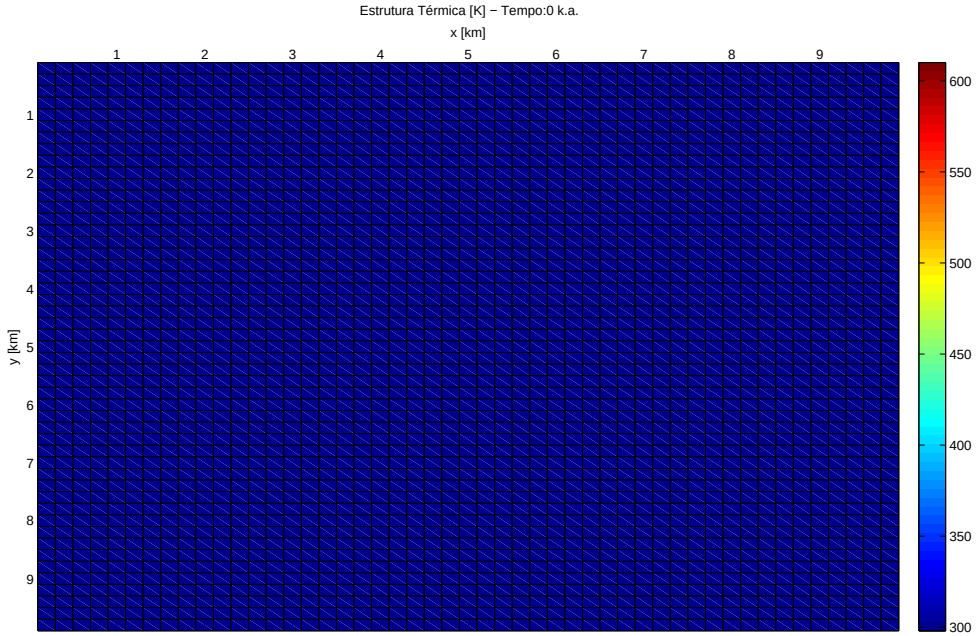


Figura 8.14: %Ro para t = 0 k.a.

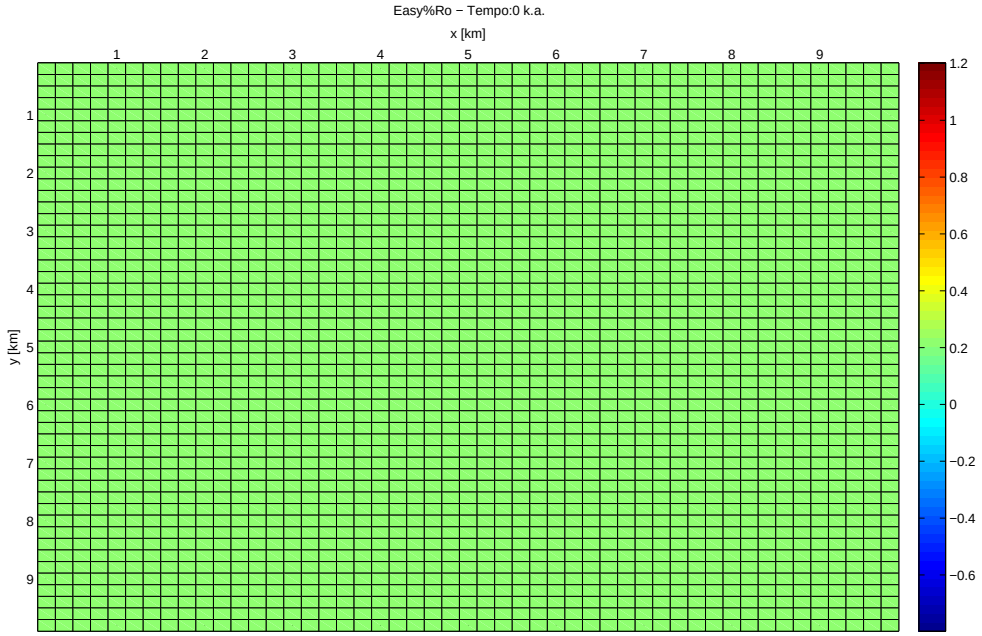


Figura 8.15: Perfil térmico para t = 12,5 k.a.

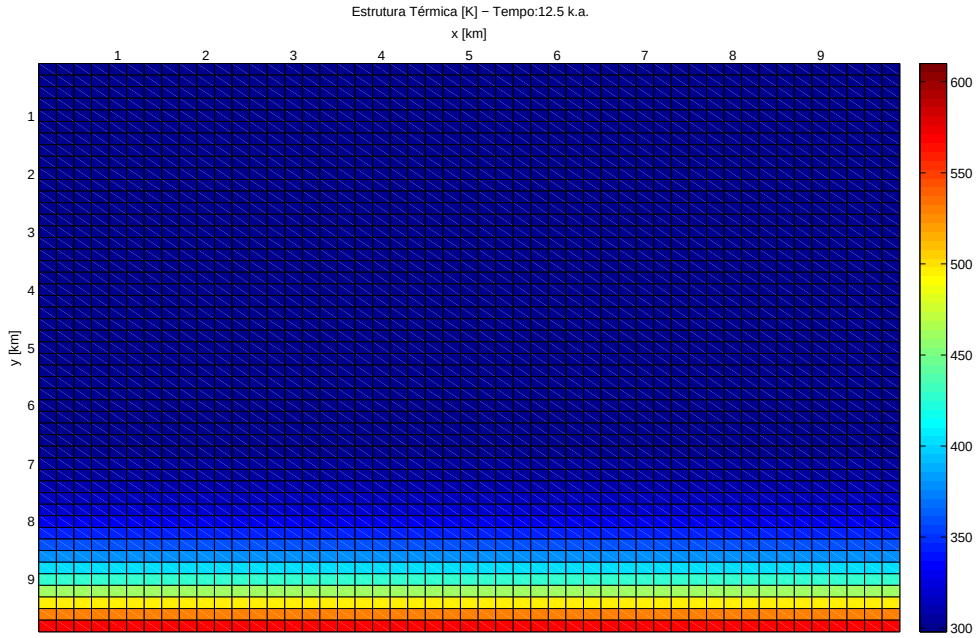


Figura 8.16: %Ro para t = 12,5 k.a.

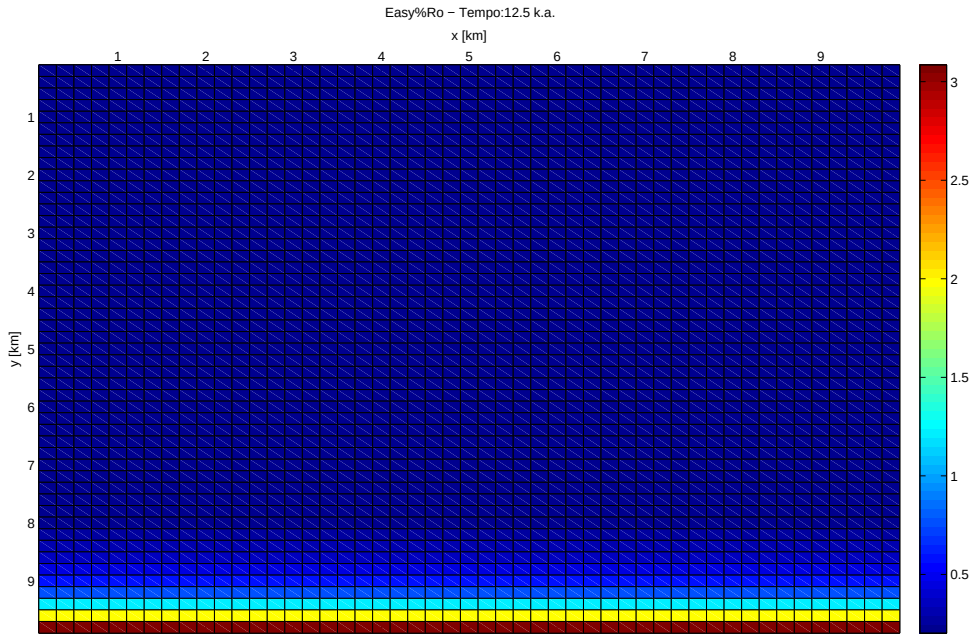


Figura 8.17: Perfil térmico para t = 25 k.a.

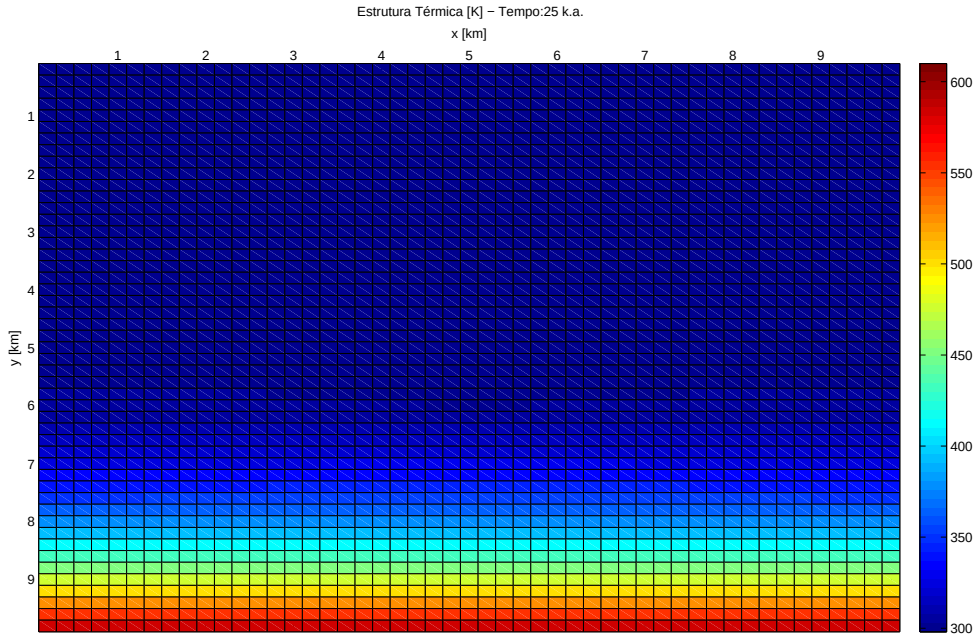
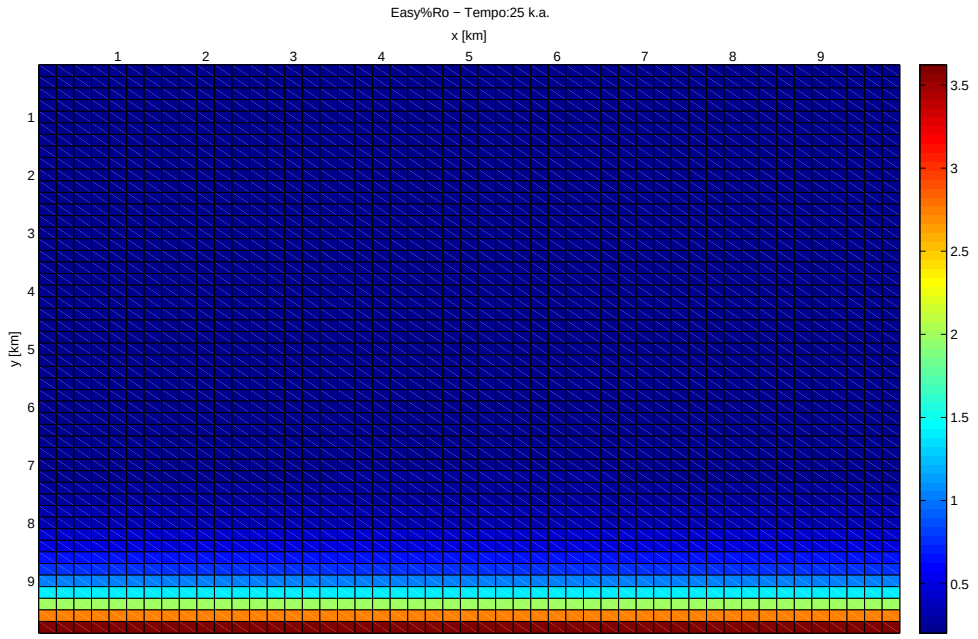
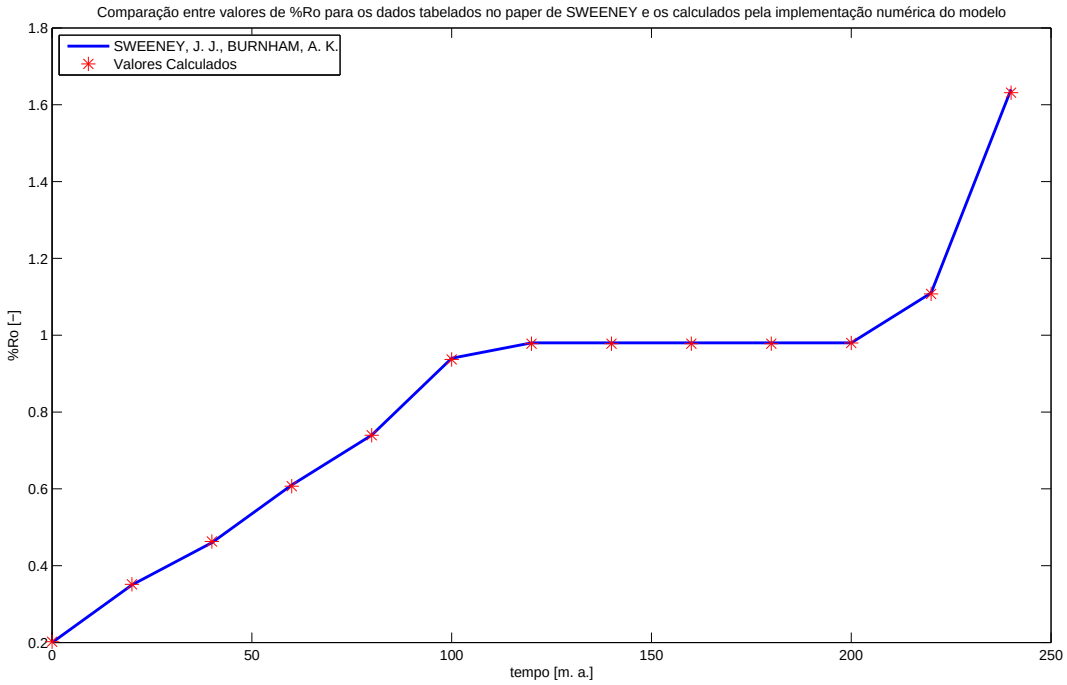


Figura 8.18: %Ro para t = 25 k.a.



Por fim, o último teste:

Figura 8.19: Comparação entre valores de %Ro para os dados tabelados em [2] e os calculados pela implementação numérica do modelo



Conforme pode ser visto nas imagens acima, as implementações numéricas realizadas apresentam as características esperadas, ajudando a confirmar a sua validade. Além disso, a imagem 8.19, demonstra que o modelo implementado para o Easy%Ro neste trabalho está de acordo com os valores apresentados pelos autores do mesmo em [2], o que confirma a robustez da implementação computacional.

8.2 Bacia do Solimões

A maturação da matéria orgânica nessa bacia foi fortemente controlada pela presença de intrusões magmáticas de grande extensão. Como era de esperar, regiões próximas às mesmas tendem a apresentar valores de reflectância da vitrinita mais altos. A casos como esses, dá-se o nome de supermaturação da matéria orgânica. Essa é uma situação em que o mais provável é de ter havido a formação de gás. Serão apresentados resultados simulados por 30 k.a.(dt = 5 k.a.). Tempos acima deste não faz-se mais necessário a análise, pois os valores de %Ro tende a um valor fixo, uma vez que as taxas de aquecimento pouco mudam.

Figura 8.20: Perfil térmico [K] para t = 0 k.a.

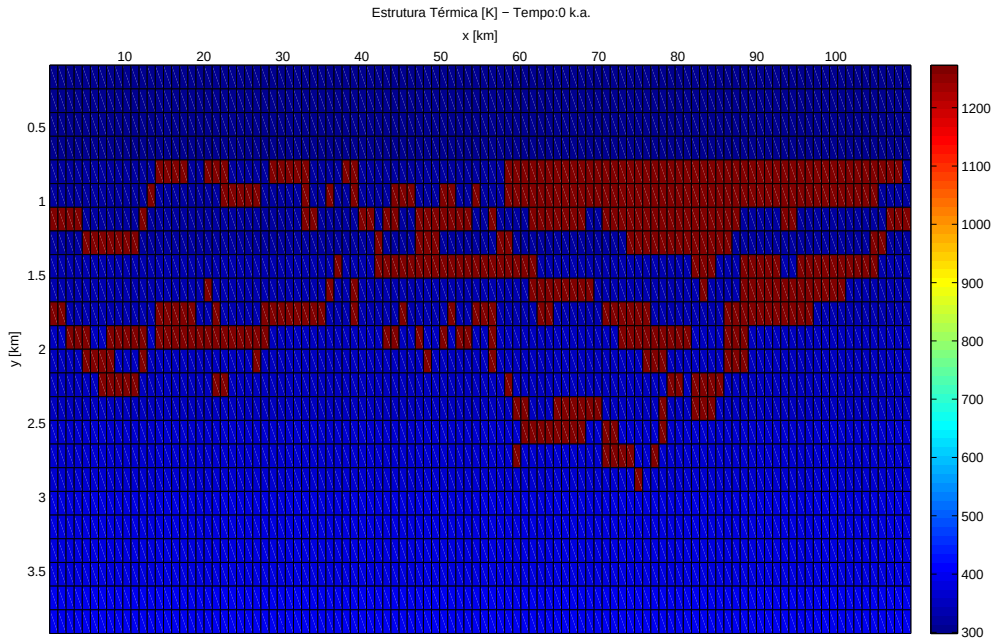


Figura 8.21: %Ro para t = 0 k.a.

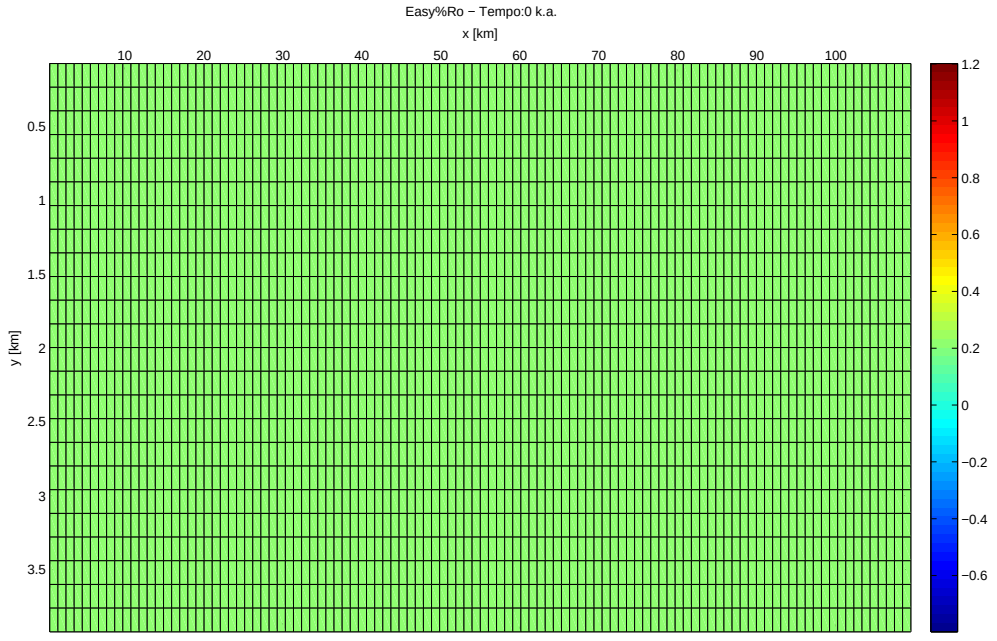


Figura 8.22: Janela de óleo [%] para t = 0 k.a.

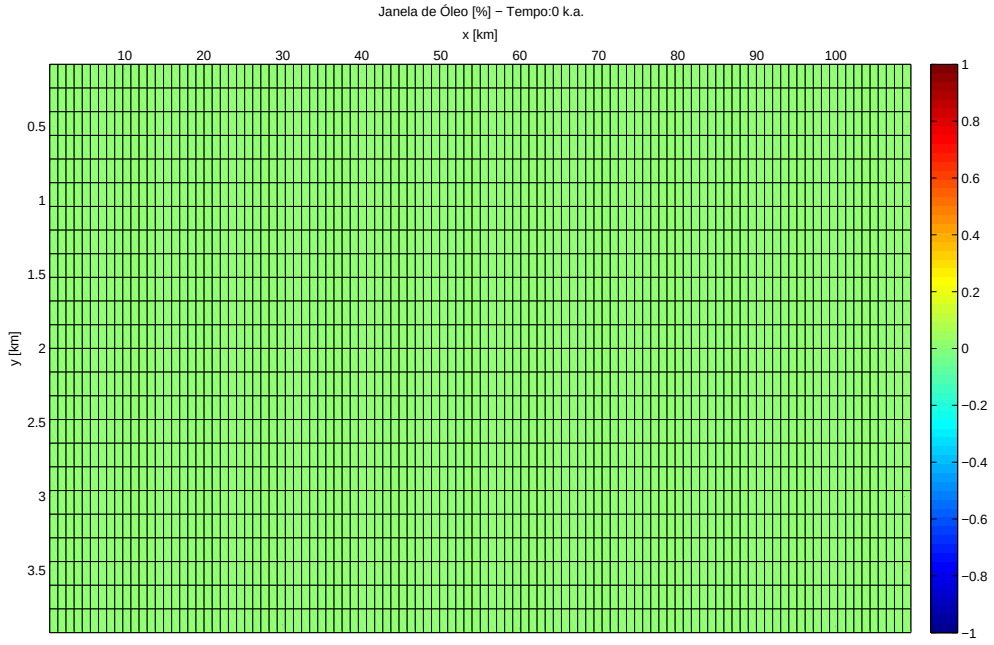


Figura 8.23: Janela de gás [%] para t = 0 k.a.

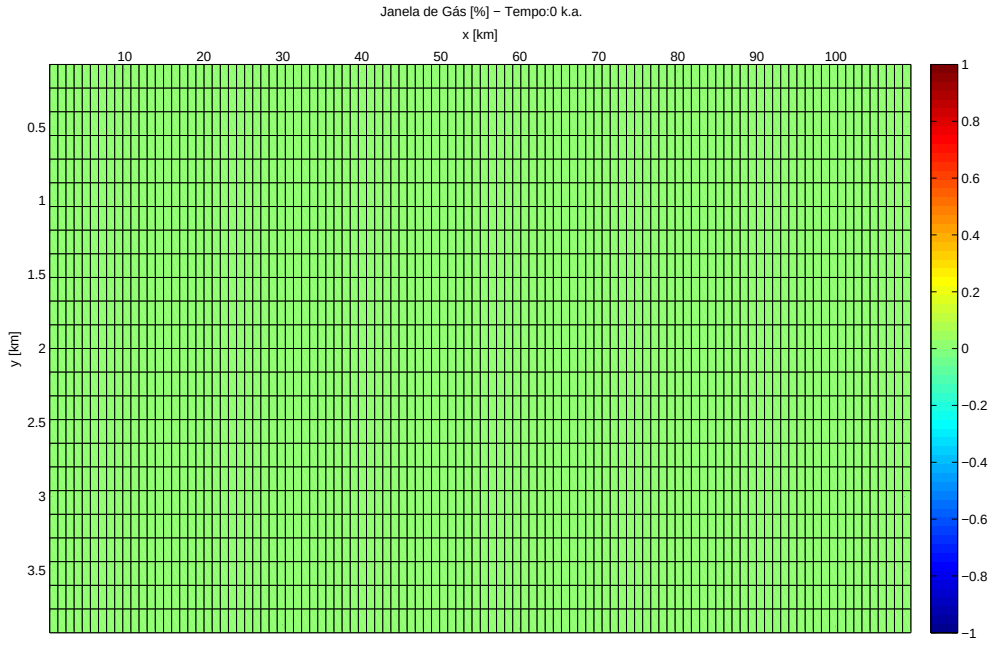


Figura 8.24: Perfil térmico [K] para t = 5 k.a.

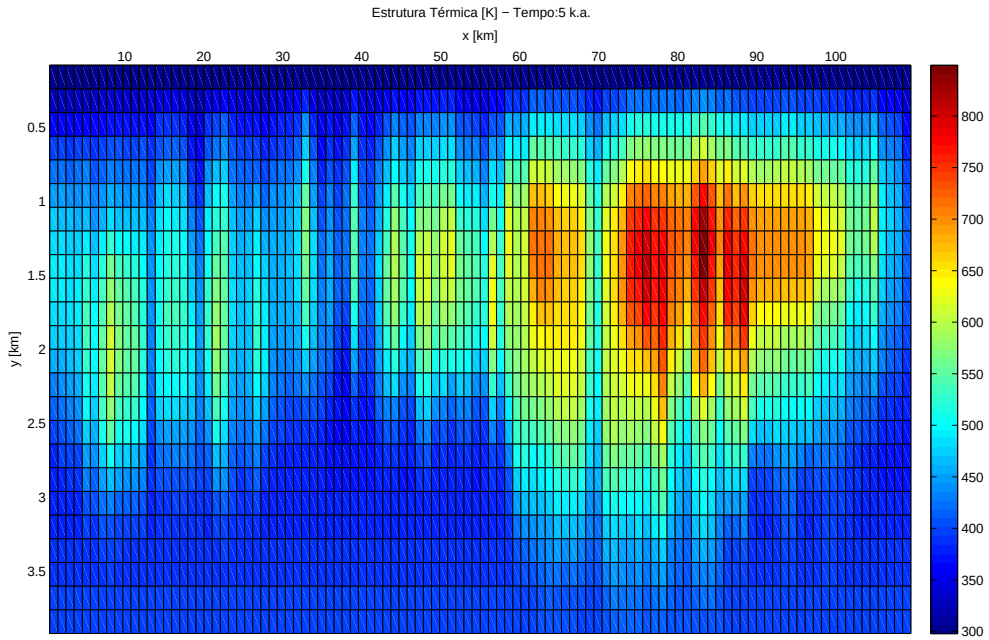


Figura 8.25: %Ro para t = 5 k.a.

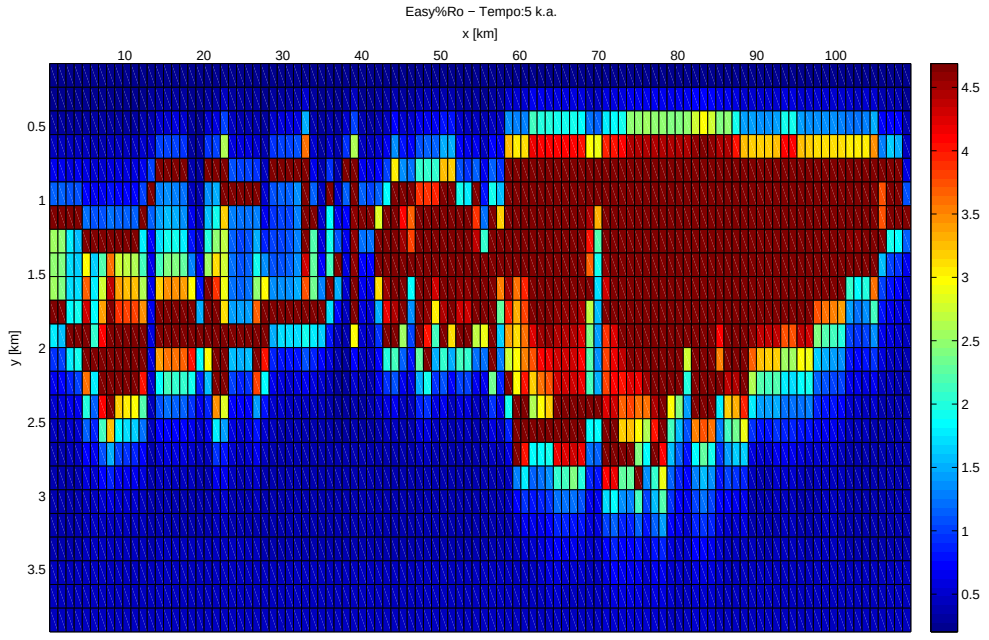


Figura 8.26: Janela de óleo [%] para t = 5 k.a.

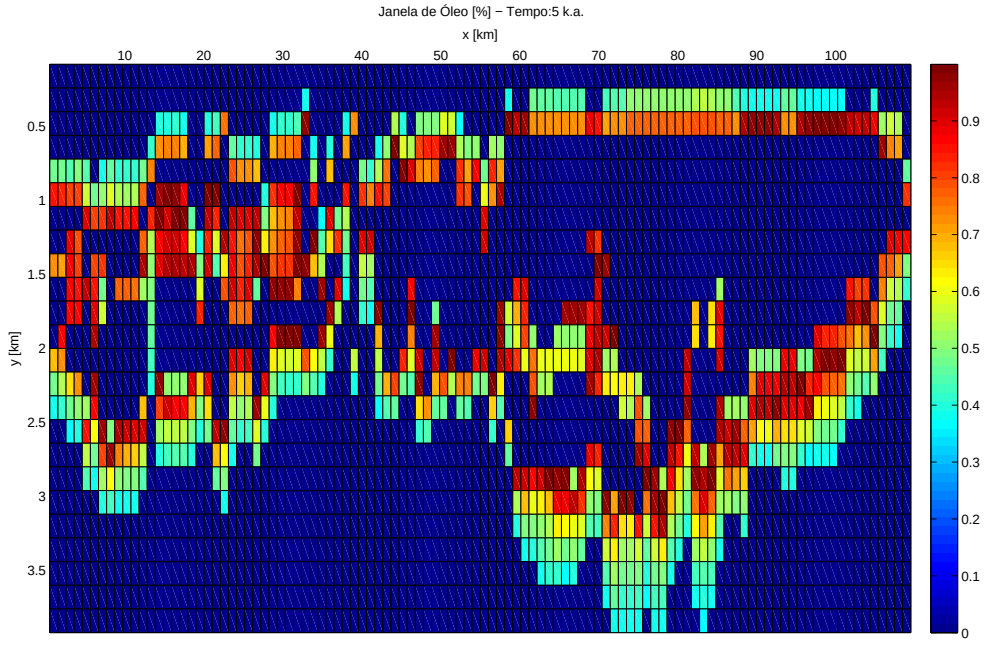


Figura 8.27: Janela de gás [%] para t = 3 k.a.

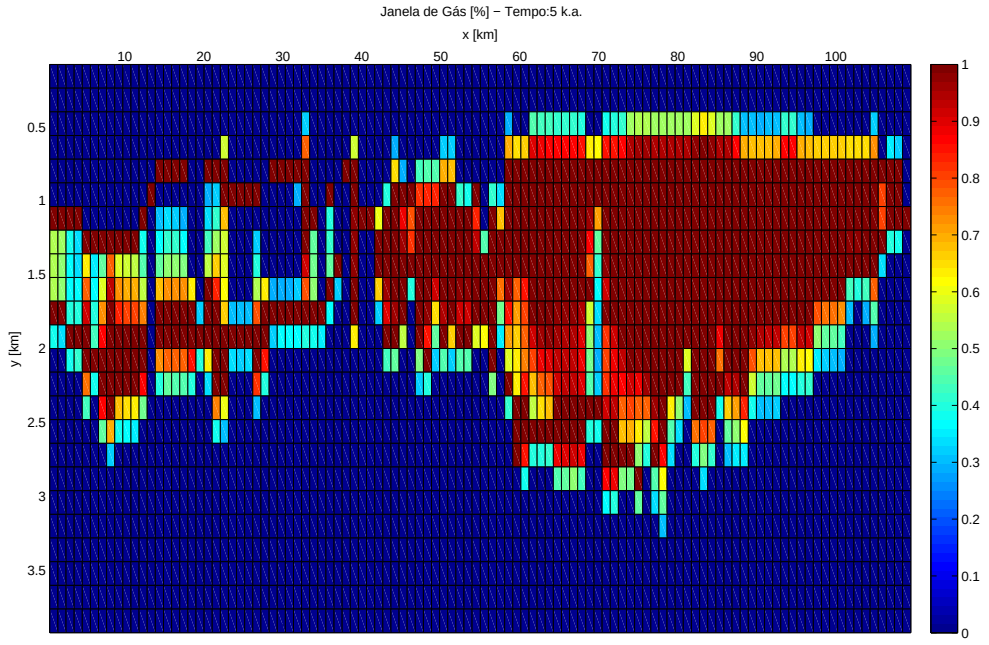


Figura 8.28: Perfil térmico [K] para t = 10 k.a.

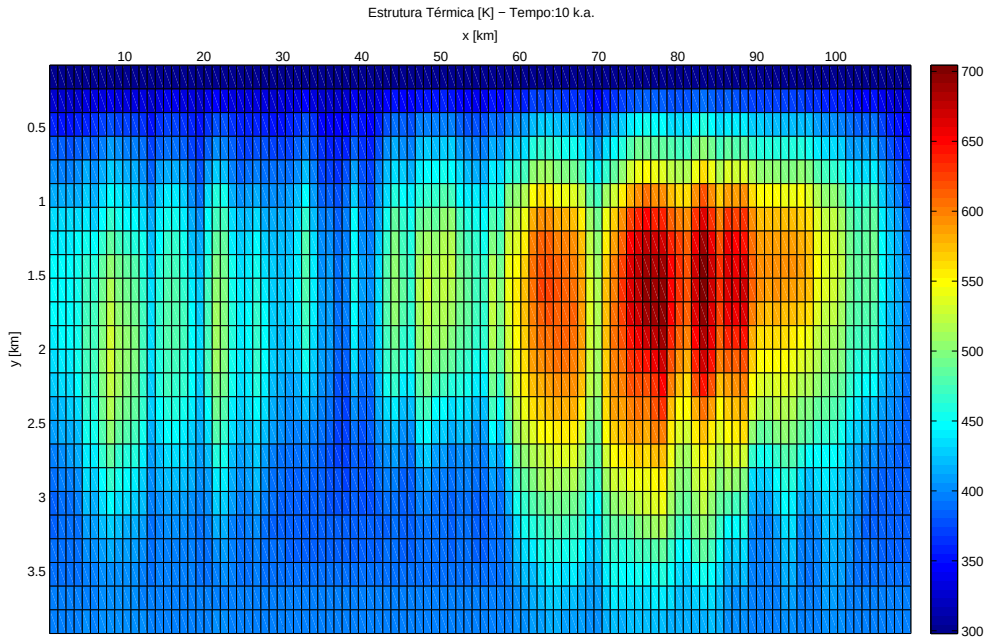


Figura 8.29: %Ro para t = 10 k.a.

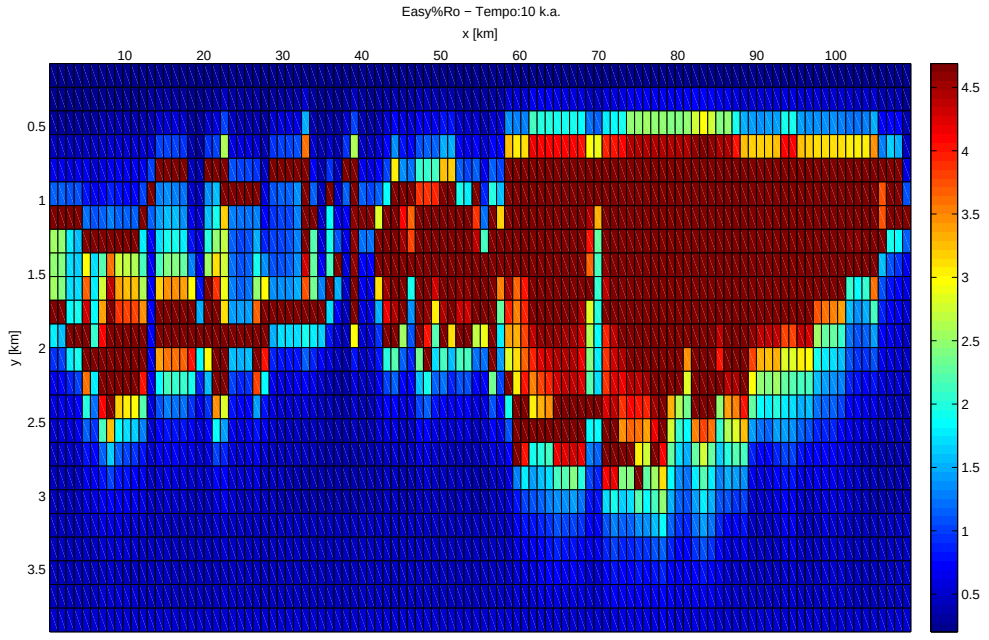


Figura 8.30: Janela de óleo [%] para t = 10 k.a.

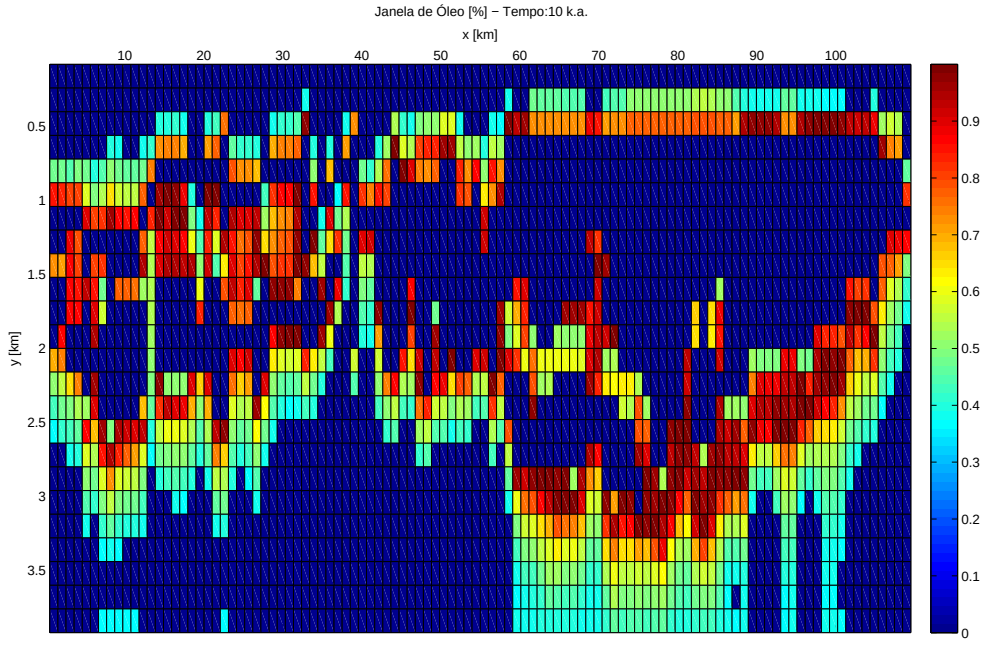


Figura 8.31: Janela de gás [%] para t = 10 k.a.

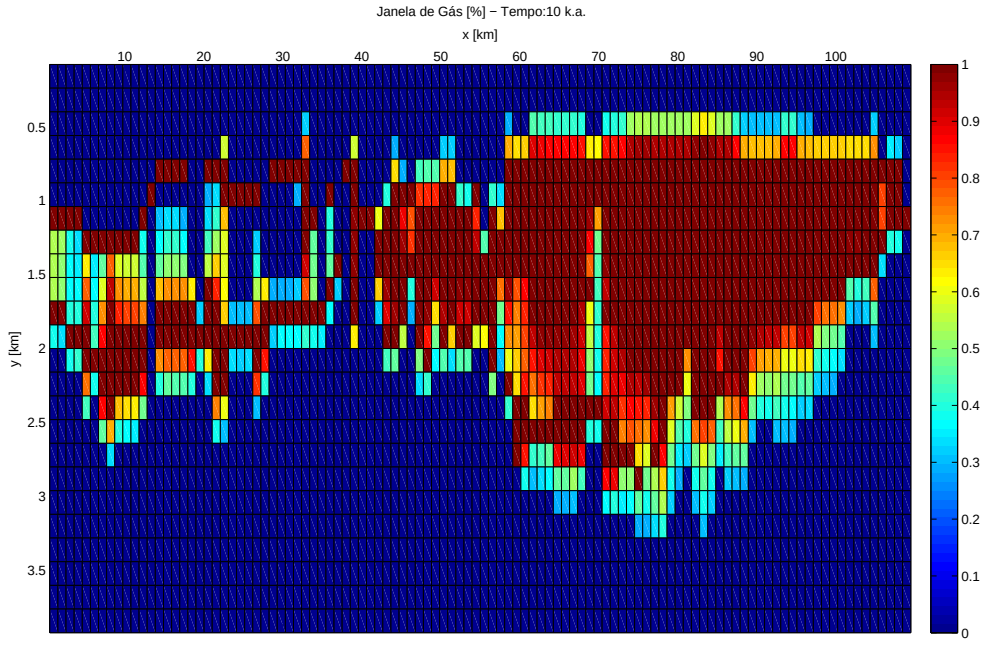


Figura 8.32: Perfil térmico [K] para t = 15 k.a.

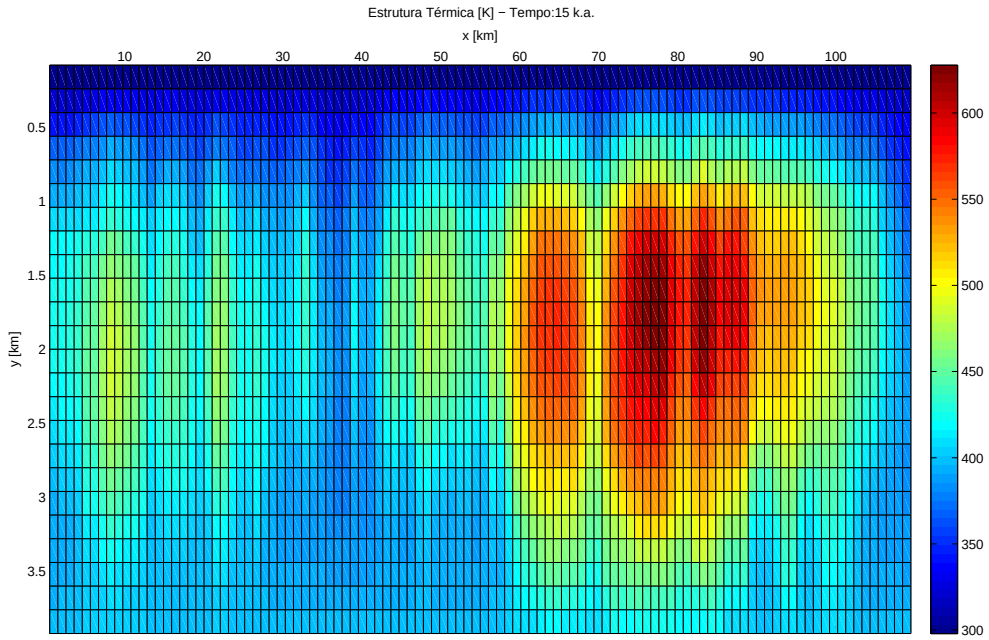


Figura 8.33: %Ro para t = 15 k.a.

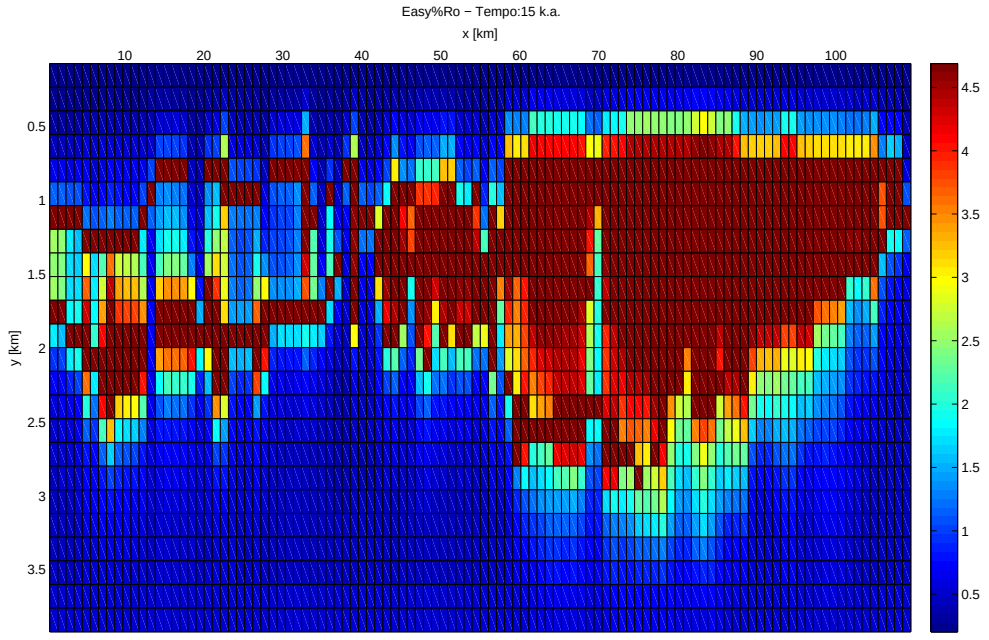


Figura 8.34: Janela de óleo [%] para t = 15 k.a.

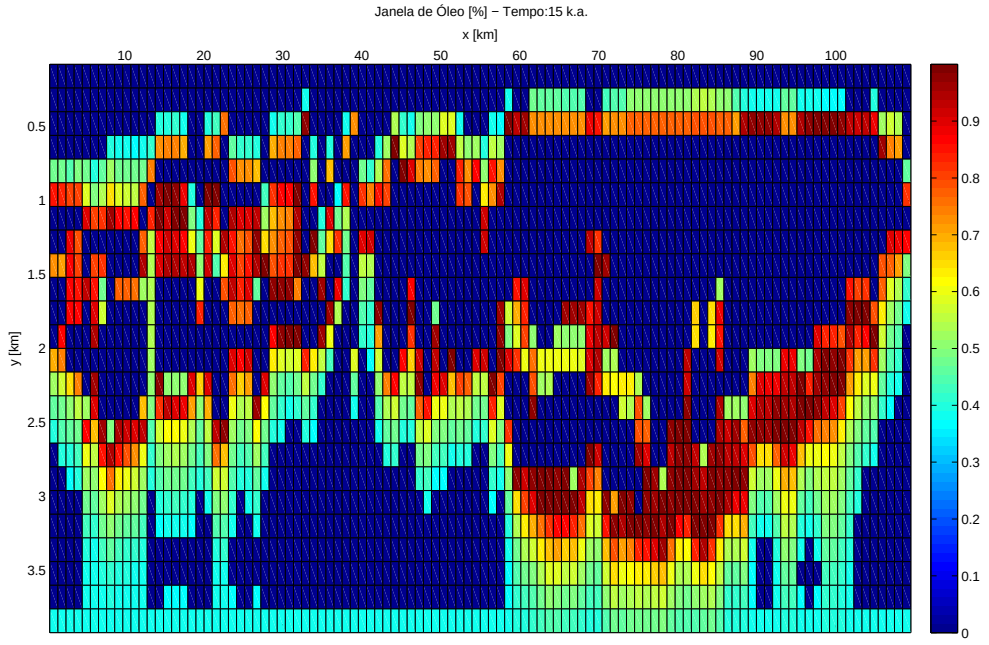


Figura 8.35: Janela de gás [%] para t = 15 k.a.

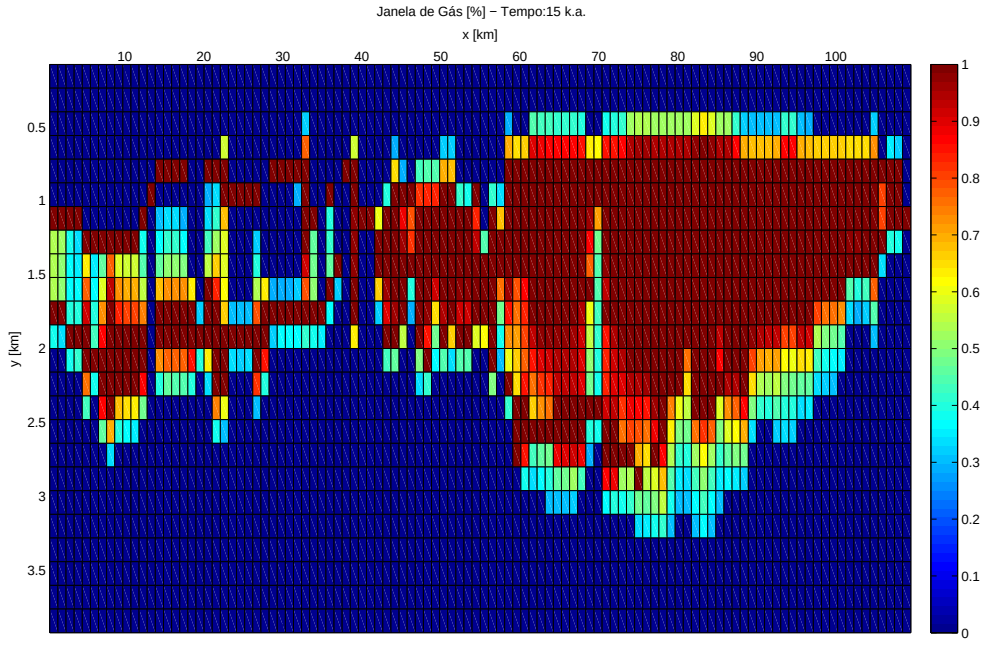


Figura 8.36: Perfil térmico [K] para t = 20 k.a.

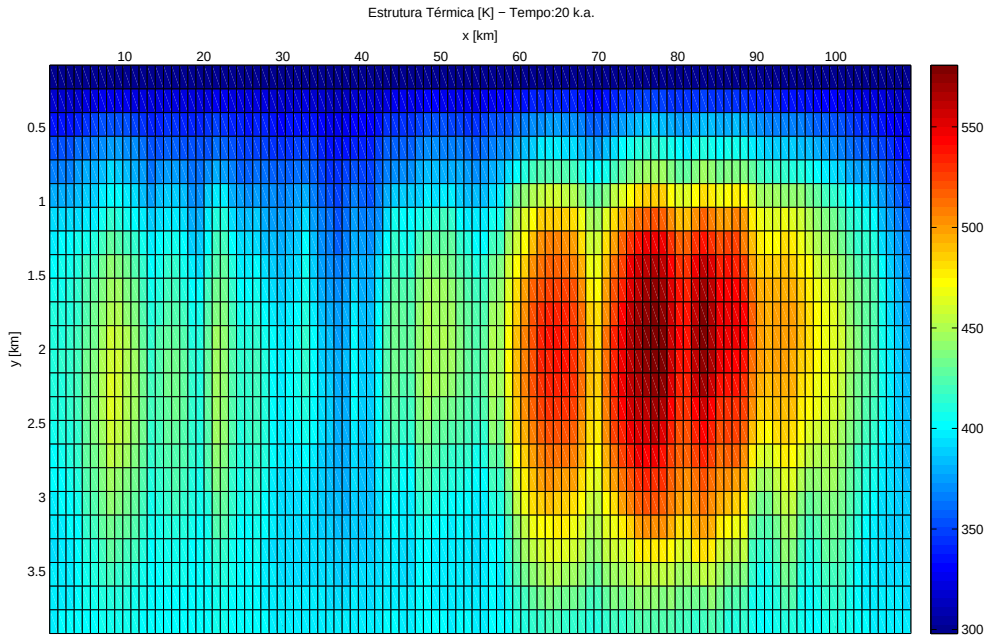


Figura 8.37: %Ro para t = 20 k.a.

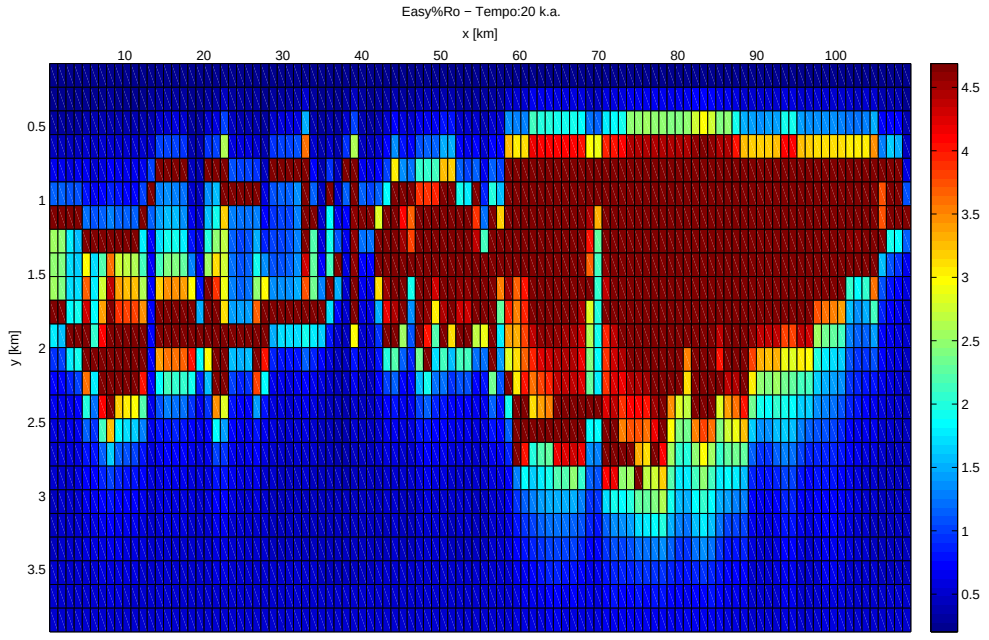


Figura 8.38: Janela de óleo [%] para t = 20 k.a.

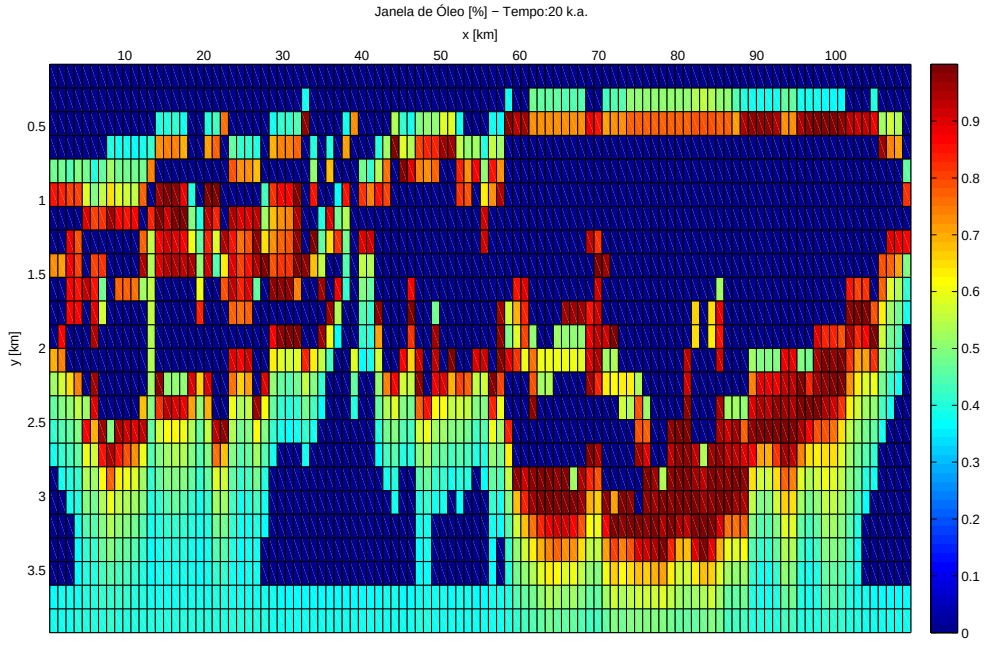


Figura 8.39: Janela de gás [%] para t = 20 k.a.

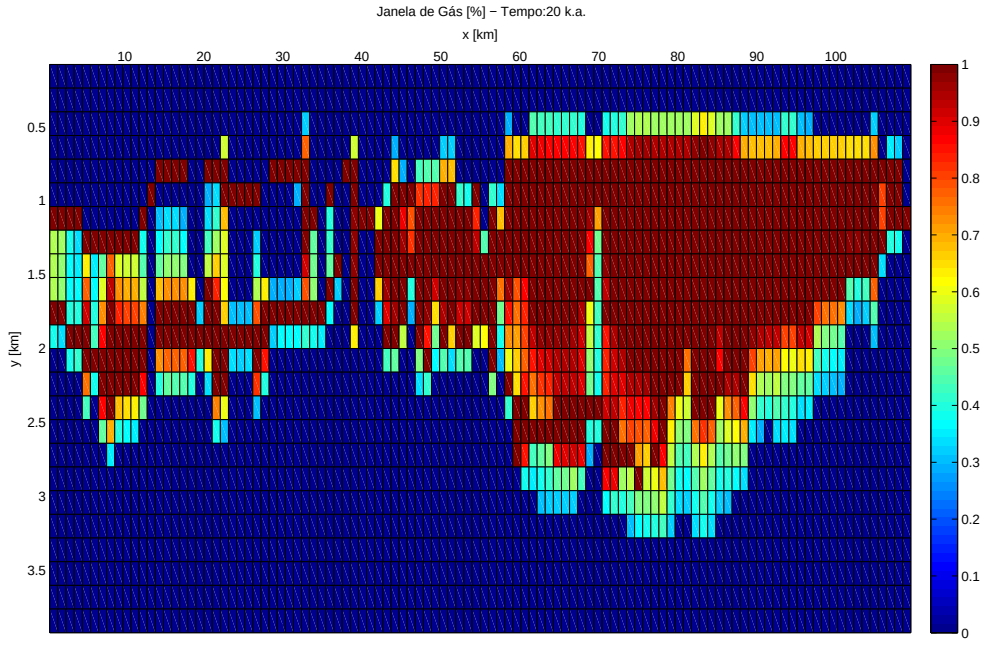


Figura 8.40: Perfil térmico [K] para t = 25 k.a.

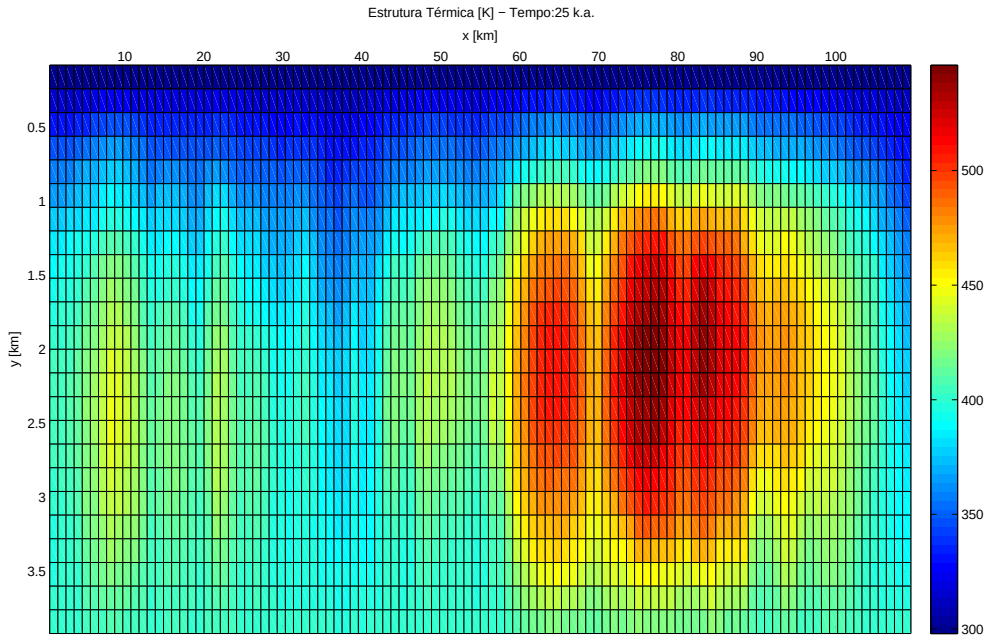


Figura 8.41: %Ro para t = 25 k.a.

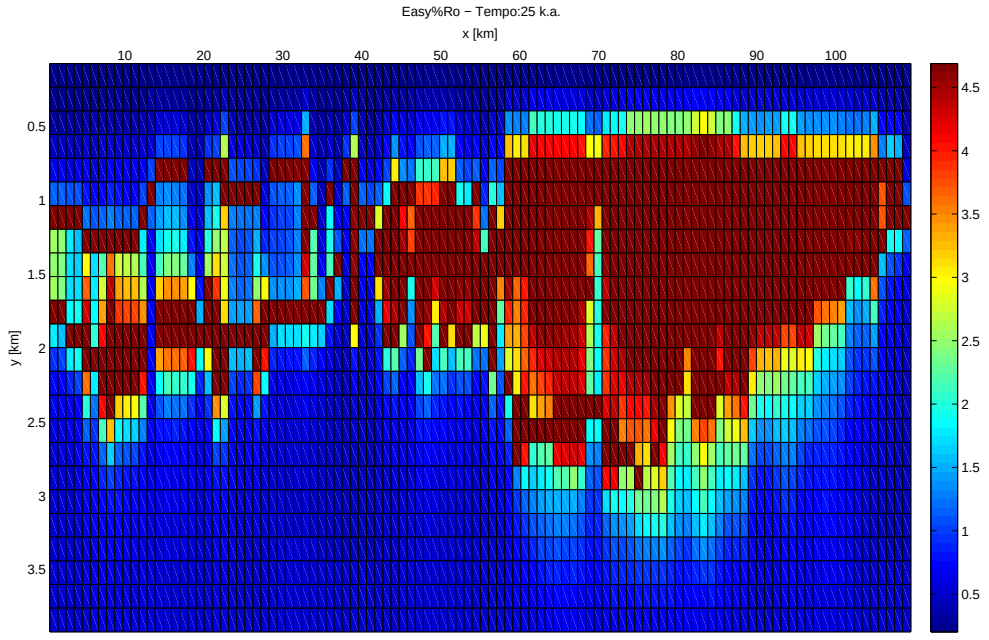


Figura 8.42: Janela de óleo [%] para t = 25 k.a.

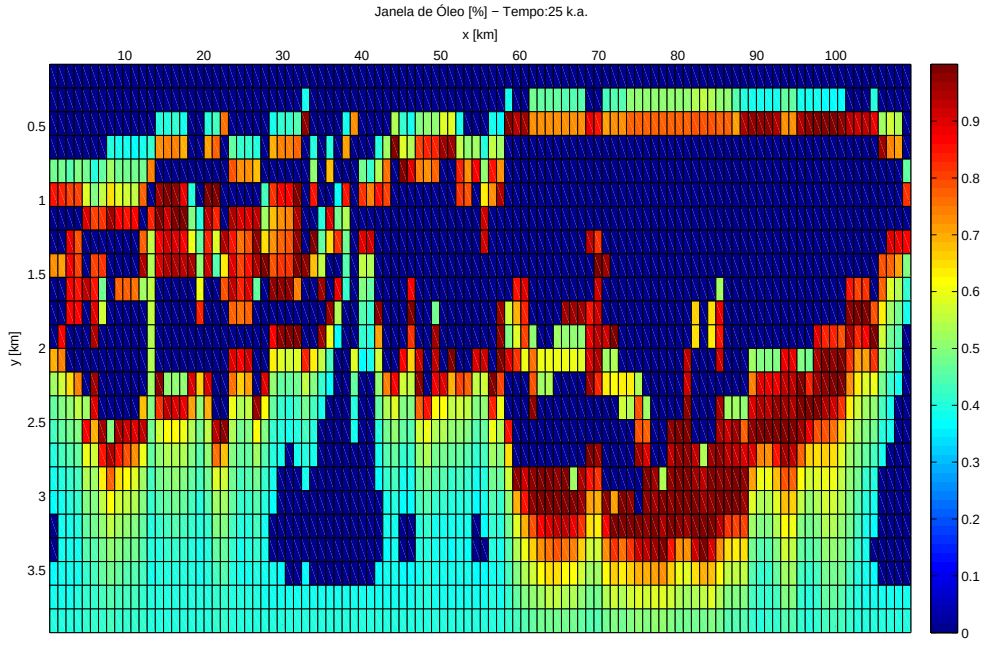


Figura 8.43: Janela de gás [%] para t = 25 k.a.

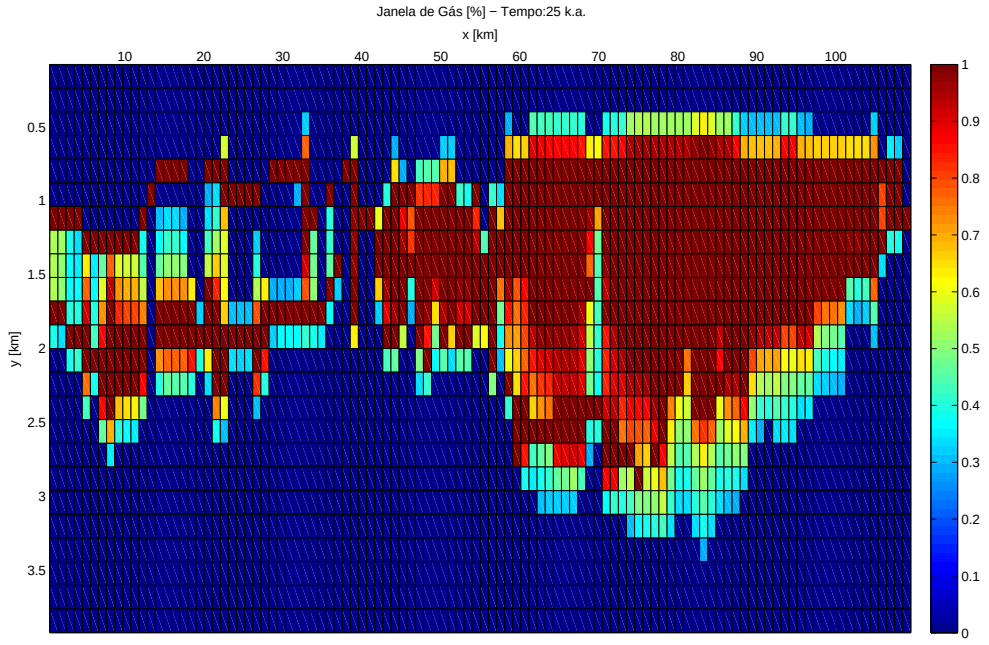


Figura 8.44: Perfil térmico [K] para t = 30 k.a.

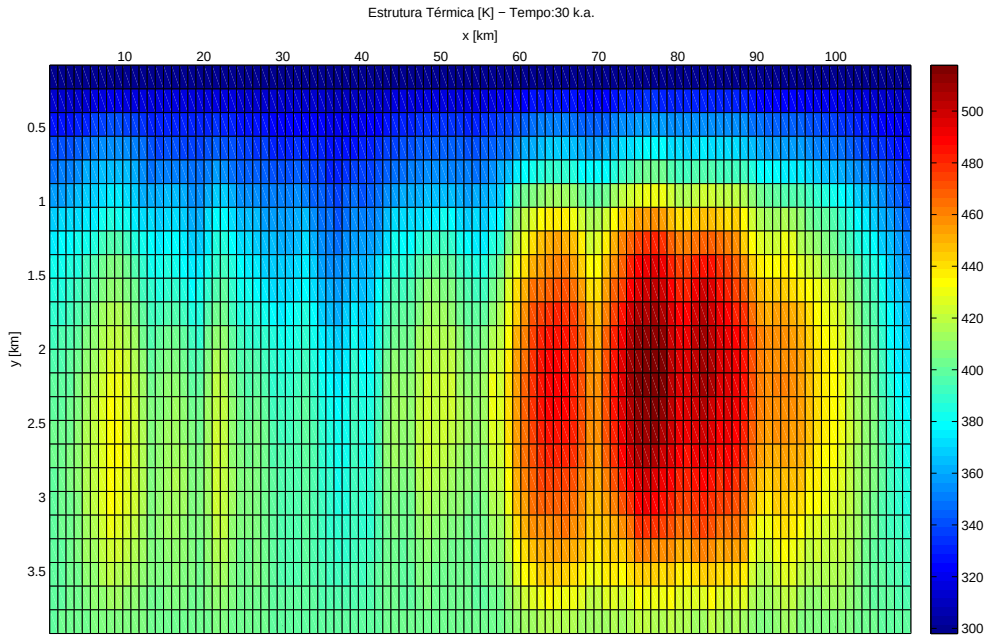


Figura 8.45: %Ro para t = 30 k.a.

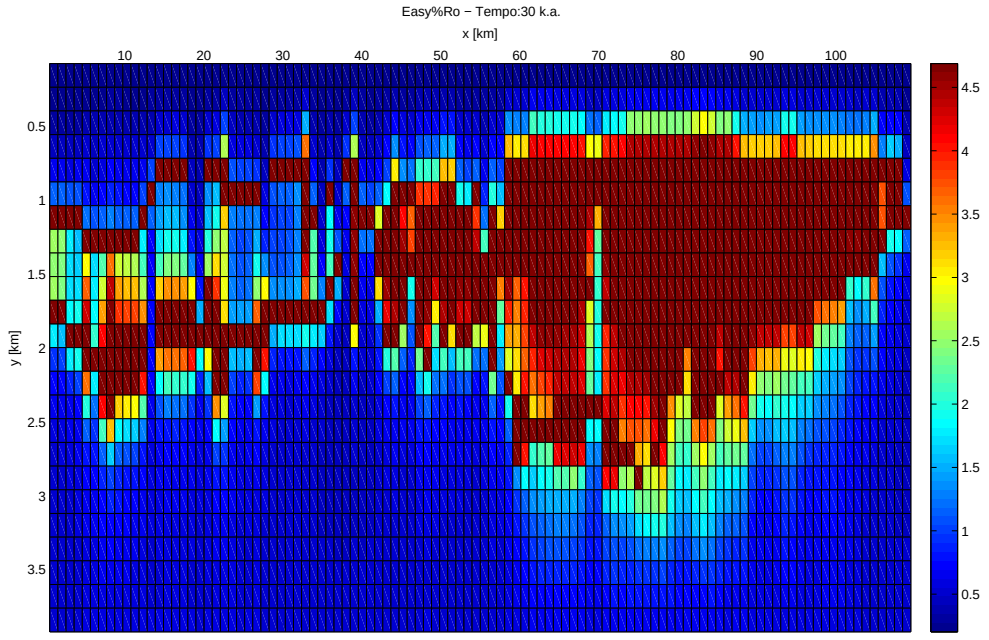


Figura 8.46: Janela de óleo [%] para t = 30 k.a.

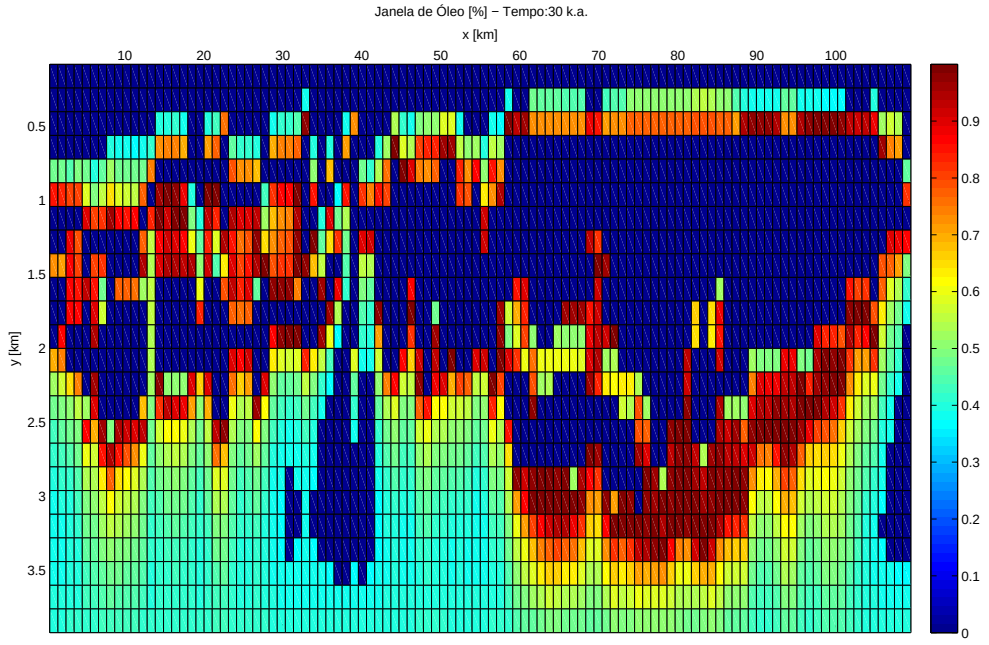
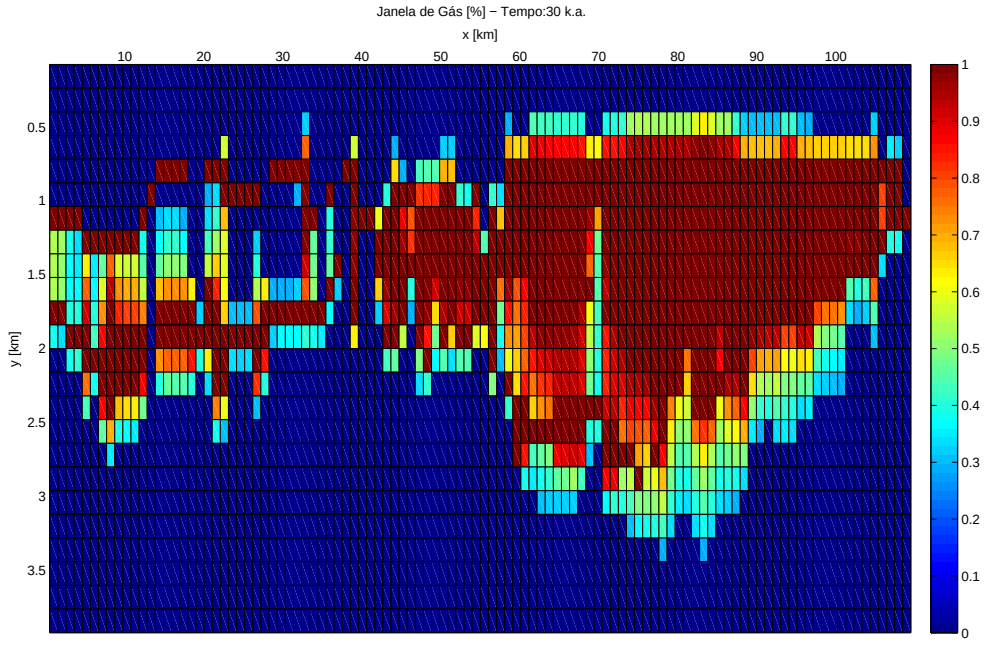


Figura 8.47: Janela de gás [%] para t = 30 k.a.



Observando as imagens acima nota-se que nas regiões próximas às intrusões, a matéria orgânica encontra-se supermaturada, o que era esperado, uma vez que a vitrinita é um paleomarcador térmico e, nos entornos das soleiras, as taxas de aquecimento são muito intensas. Além disso, observa-se que no Grupo Tefé, rocha geradora da Bacia do Solimões, há presença de "bolsões" nos quais não houve a supermaturação da matéria orgânica, indicando a possibilidade de ter havido de fato geração de hidrocarboneto e este teria migrado através do sistema de falhas da bacia para outras regiões.

Observa-se também, pelas janelas de óleo, que há potencial da existência de óleo em algumas localidades da bacia, inclusive em regiões ainda pouco exploradas.

Capítulo 9

Conclusão

A metodologia proposta neste trabalho foi aplicada com relativo sucesso no caso de estudo. O método numérico aplicado, mostrou-se satisfatório tanto pelo ponto de vista computacional quanto pela facilidade de implementação. Contudo, a restrição presente para os passos de tempo pode tornar-se um inconveniente. Deste modo, talvez seja interessante, em trabalhos futuros, substituir o método utilizado por algum outro mais robusto, tal como Volumes Finitos baseados em Elementos, Elementos Finitos, ou Elementos de Contorno.

Quanto ao modelo Easy%Ro, ele mostrou-se simples de implementar e apresenta bons resultados. Contudo, há ainda alguns problemas quando as taxas de aquecimento são muito pequenas e/ou muito grandes. Por outro lado, os resultados obtidos foram satisfatórios, com valores de %Ro dentro da zona de óleo em regiões correspondentes à rocha geradora do sistema petrolífire Jandiatuba-Juruá(!). Esse resultado é importante, uma vez que pode ajudar a justificar a existência de óleo na bacia.

Os pontos abaixo apresentam experiências adquiridas ao longo do trabalho e merecem ser ressaltadas:

- métodos incondicionalmente estáveis podem não garantir soluções fisicamente corretas;
- é importante compreender o significado numérico de cada termo nas equações diferenciais parciais, de modo a aplicar o melhor método de resolução para suas características;
- modelo Easy%Ro mostrou-se de fácil implementação, contudo para maior acurácia faz-se necessário utilizar esquemas mais incrementados do ponto de vista da cinética química;
- é fundamental que haja a possibilidade de calibração dos valores calculados de %Ro por meio de dados reais de poços, deste modo,

confirmando a qualidade da simulação;

- ferramentas de simulação de bacias sedimentares e avaliação de potencial petrolífero constituem-se em ferramentas extremamente importantes no atual momento da indústria do petróleo.

De acordo com os resultados gerados numericamente, pode-se dizer que:

- as janelas de óleo obtidas confirmam o potencial petrolífero da Bacia do Solimões, inclusive em regiões ainda pouco exploradas;
- as janelas de gás confirmam o potencial de gás na bacia, principalmente nas regiões localizadas nos entornos das soleiras de diabásio;
- os valores de %Ro obtidos mostram que, de fato, os eventos magmáticos (instrusões) ocorridos nessa bacia foram fundamentais para que houvesse a possibilidade de geração de hidrocarbonetos;
- as três conclusões anteriores possibilitam afirmar que sistemas petrolíferos ígneo-sedimentares podem ser uma alternativa interessante quanto a sua potencialidade de óleo e gás, deste modo, configurando-se como uma fronteira exploratória viável.

Referências Bibliográficas

[1] ANDERSON, J. D. *Computational fluid dynamics: the basics with applications*. 1st ed. New York, McGraw–Hill, Inc., 1995. ISBN: 0–07–001–001685–2.

[2] SWEENEY, J. J., BURNHAM, A. K. "Evaluation of a simple model of vitrinite reflectance based on chemical kinetics", *The American Association of Petroleum Geologists Bulletin*, v. 74, pp. 1559–1570, 1990.

[3] MALISKA, C. R. *Transferência de calor e mecânica dos fluidos computacional*. 2^a ed. , LTC, 2004. ISBN: 9788–52–1613–9–61.

[4] BRITZ, D., OLDHAM, K. B., ØESTERBY, O. "Strategies for damping the oscillations of the alternating direction implicit method of simulation of diffusion-limited chronoamperometry at disk electrodes", *Electrochimica Acta*, v. 54, pp. 4822–4828, 2009.

[5] HANTSCHEL, T., KAUERAUF, A. I. *Fundamentals of basin and petroleum systems modeling*. 1st ed. Berlin Heidelberg, Springer–Verlag, 2009. ISBN: 978–3–540–72317–2.

[6] MAGOON, L. B., DOW, W. G. "The petroleum system – from source to trap", *Am. Assoc. Petr. Geol. Memoir*, v. 60, pp. 3–24, 1994.

[7] KOENIG, A., MOO, B. E. *Accelerated C++ – practical programming by example*. 1st ed. Boston, Addison–Wesley, 2000. ISBN: 0–201–70353–x.

[8] SAVITCH, W. *C++ absoluto*. 1st ed. São Paulo, Addison–Wesley, 2004. ISBN: 85–88639–09–2.

[9] LUTZ, M., ASCHER, D. *Learning Python*. 1st ed. , O'Reilly Media, 1999. ISBN: 978–85–7780–013–1.

[10] BARATA, C. F., CAPUTO, M. V. "Geologia do petróleo da Bacia do Solimões: o estado da arte", *4º PDPETRO*, 2007.

[11] MELLO, M. R., MOHRIAK, W. U., KOUTSOUKOS, E. A. M., etal. "Selected petroleum systems in Brazil", *Am. Assoc. Petr. Geol. Memoir*, v. 60, pp. 499–512, 1994.

[12] EIRAS, J. F. "Cenário geológico nas bacias sedimentares no Brasil: tectônica, sedimentação e sistemas petrolíferos da Bacia do Solimões, Estado do Amazonas", *Searching for Oil and Gas in the Land of Giants Buenos Aires*, pp. 23–31, 1998.

[13] EIRAS, J. F. "Geologia e sistemas petrolíferos da Bacia do Solimões", *VI Simpósio de Geologia da Amazônia*, 1999.

[14] LUTHI, S. M., O'BRIEN, D. P. "A quantitative thermodynamic model for diabase intrusions in the amazonas basin – Brazil, and their effect on maturation of organic matter", *3rd International Congress SBGF*, v. 45, pp. 255–282, 1993.

[15] CORRÊA, L. M. S. A. *Avaliação do efeito térmico das soleiras de diabásio nas rochas geradoras da Formação Irati (Bacia do Paraná, Brasil) através de técnicas de modelagem numérica*. Tese de Mestrado, Universidade do Estado do Rio de Janeiro, Rio de Janeiro – RJ, 2007.

[16] BAUDINO, R., PONTET, M., COUMONT, H., etal. "Thermal intrusion: a non-conventional petroleum system and a challenge for basin modelling". . IV ALAGO Workshop – Basin Modeling, 2006.

[17] FJELDSKAAR, W., HELSET, H. M., JOHANSEN, H., etal. "Thermal modelling of magmatic intrusions in the Gjallar Ridge, Norwegian Sea: implications for vitrinite reflectance and hydrocarbon maturation", *Basin Research*, v. 20, pp. 143–159, 2008.

[18] CAPUTO, M. V. *Stratigraphy, tectonics, paleoclimatology and paleogeography of Northen Basins of Brazi*. Tese de Doutorado, Universidade da Califórnia, Santa Bárbara, 1984.

[19] "Portal Amazonia". , Fevereiro 2010. `http://portalamazonia.globo.com`.

[20] BENDER, A. A., EIRAS, J. F., FILHO, WANDERLEY, J. R., etal. "Quantificação 3D da evolução termal da Bacia do Solimões e suas implicações petrolíferas". . VII Simpósio de Geologia da Amazônia, 2001.

[21] FILHO, WANDERLEY, J. R., TRAVASSOS, W. A. S., ALVES, D. B. "O diabásio nas bacias paleozóicas amazônicas - herói ou vilão", *Boletim de Geociências da Petrobras*, v. 14, pp. 117–184, 2005–2006.

[22] EIRAS, J. F., BECKER, C. R., SOUZA, E. M., etal. "Bacia do Solimões", *Boletim de Geociências da Petrobras*, v. 8, pp. 17–45, 1994.

[23] LEVEQUE, R. J. *Numerical methods for conservation laws.* 2nd ed. , Birkh'auser Basel, 2005. ISBN: 978–3764327231.

[24] MINKOWYCZ, W. J., SPARROW, E. M., SCHNEIDER, G. E. *Handbook of numerical heat transfer.* 2nd ed. New York, John Wiley & Sons Inc., 1988. ISBN: 978–0–471–34878–8.

[25] PATANKAR, S. V. *Numerical heat transfer and fluid flow.* 1st ed. , Taylor & Francis, 1980. ISBN: 0–89116–522–3.

[26] PEACEMAN, D. W., RACHFORD, JR., H. H. "The numerical solution of parabolic and elliptic differential equations", *J. Soc. Indust. Math.*, v. 3, pp. 28–41, 1955.

[27] DOUGLAS, JR., J. "On the numerical integration of $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{\partial u}{\partial t}$ by implicit methods", *J. Soc. Indust. Appl. Math.*, v. 3, pp. 42–65, 1955.

[28] DOUGLAS, JR., J., GUNN, JAMES, E. "Alternating direction methods for parabolic systems in m space variables", *J. Assn. for Comp. Machinery*, v. 9, pp. 450, 1962.

[29] THIBAULT, J. "Comparison of nine three-dimensional numerical methods for the solution of the heat diffusion equation", *Numerical Heat Transfer*, v. 8, pp. 281, 1985.

[30] HUNDSDORFER, W. H., VERWER, J. G. "Stability and convergence of the peaceman-rachford ADI method for initial-boundary value problem", *Mathematics of Computation*, v. 53, pp. 81–101, 1989.

[31] DOUGLAS, JR., J., PEACEMAN, D. "Numerical solution of two-dimensional heat flow problems", *American Institute os Chemical Engineering Journal*, v. 1, pp. 505–512, 1955.

[32] PEARSON, C. E. "Impulsive end condition for diffusion equation", *Math. Comp.*, v. 19, pp. 570–576, 1965.

[33] DOUGLAS, JR., J. "A survey of numerical methods for parabolic differential equations", *Advances in Computers*, v. 2, pp. 1–54, 1961.

[34] GLASSTONE, S., LAIDLER, K. J., EYRING, H. "The theory of rate processes", *J. Chem. Educ.*, v. 5, pp. 249, 1942.

Apêndice A

Código Fonte

```
1  // Vector visualization.
2  // Ex: DisplayVector(V);
3
4  #ifndef DISPLAY_VECTOR_H
5  #define DISPLAY_VECTOR_H
6
7  using namespace std;
8
9  template <class T>
10 void DisplayVector(const T& v)
11 {
12     copy(v.begin(), v.end(), ostream_iterator<typename T::value_type> (
           cout, "\n"));
13 };
14
15 #endif
```

Listing A.1: display_vector.h

```
1  // Matrix definitions.
2  // Ex: matrix(3, 4);
3
4  #ifndef MATRIX_H
5  #define MATRIX_H
6
7  #include <vector>
8
9  using namespace std;
10
11 template <typename T>
12 class matrix
13 {
14 public:
```

```
15      matrix(){};
16
17      matrix(int rows, int cols)
18      {
19          for (int i = 0; i < rows; ++i)
20          {
21              data_.push_back(vector<T>(cols));
22          }
23      }
24
25      inline vector<T> & operator[](int i) { return data_[i]; }
26
27      inline const vector<T> & operator[](int i) const { return data_[i];
            }
28
29      void resize(int rows, int cols)
30      {
31          data_.resize(rows);
32          for (int i = 0; i < rows; ++i)
33          {
34              data_[i].resize(cols);
35          }
36      }
37
38  private:
39      vector<vector<T>> data_;
40  };
41
42  #endif
```

Listing A.2: matrix.h

```
1  #ifndef INPUT_STRUCT_H
2  #define INPUT_STRUCT_H
3
4  // Data struct for the input variables.
5  struct InputVariables
6  {
7      // Variables.
8      // Input.
9      double granite_density;
10     double granite_condutivity;
11     double granite_specific_heat_capacity;
12     double granite_anisotropy;
13     double shale_density;
14     double shale_condutivity;
```

```
15    double shale_specific_heat_capacity;
16   double shale_anisotropy;
17    double shaly_limestone_density;
18    double shaly_limestone_condutivity;
19    double shaly_limestone_specific_heat_capacity;
20   double shaly_limestone_anisotropy;
21    double calcareous_ooid_density;
22    double calcareous_ooid_condutivity;
23    double calcareous_ooid_specific_heat_capacity;
24    double calcareous_ooid_anisotropy;
25    double sandstone_density;
26    double sandstone_condutivity;
27    double sandstone_specific_heat_capacity;
28   double sandstone_anisotropy;
29    double magma_density;
30    double magma_condutivity;
31    double magma_specific_heat_capacity;
32    double magma_anisotropy;
33    double diabase_density;
34    double diabase_condutivity;
35    double diabase_specific_heat_capacity;
36    double diabase_anisotropy;
37
38    int number_of_blocks_in_x_direction;
39    double lenght_x_direction;
40    double dx;
41    int number_of_blocks_in_y_direction;
42    double length_y_direction;
43    double dy;
44    int number_of_time_steps;
45    double total_simulation_time;
46    double dt;
47
48    double geothermal_gradient;
49    double northern_boundary_prescribed_temperature;
50    double southern_boundary_prescribed_temperature;
51   double eastern_boundary_prescribed_flux;
52    double western_boundary_prescribed_flux;
53
54    double magma_solid_temp;
55    double magma_init_temp;
56
57
58    //EasyRo%
59    int number_of_activation_energies;
60
61    double frequency_factor;
```

```
62     double gas_constant;
63     double a1;
64     double a2;
65     double b1;
66     double b2;
67 
68     std::vector<double> activation_energy;
69     std::vector<double> stoichiometric_factor;
70 
71     bool calculate_easy_ro;
72 };
73 
74 #endif
```

Listing A.3: input_struct.h

```
1  // name OF CLASS: MaturationModel
2  // CREDIT:
3  // PURPOSE:
4  //     This class implements a organic matter maturation model.
5 
6 
7  #ifndef MATURATION_MODEL_H
8  #define MATURATION_MODEL_H
9 
10 #include "input_struct.h"
11 #include "mesh.h"
12 #include "simulation_io.h"
13 #include "matrix.h"
14 
15 using std::vector;
16 
17 class MaturationModel
18 {
19 public:
20     MaturationModel(){};
21 
22 
23     void EasyRoCalculation(
24         InputVariables &easy_ro_input,
25         SimulationIO io,
26         Mesh mesh
27         )
28     {
29         // Local variables.
30         int n_components = easy_ro_input.number_of_activation_energies;
```

```
31      int nt = easy_ro_input.number_of_time_steps;
32      int nx = easy_ro_input.number_of_blocks_in_x_direction;
33      int ny = easy_ro_input.number_of_blocks_in_y_direction;
34      int n;
35      int time_step;
36      int i;
37      int j;
38      int component;
39      int index;
40
41      double A = easy_ro_input.frequency_factor;
42      double R = easy_ro_input.gas_constant;
43      double a1 = easy_ro_input.a1;
44      double a2 = easy_ro_input.a2;
45      double b1 = easy_ro_input.b1;
46      double b2 = easy_ro_input.b2;
47      double dt = easy_ro_input.dt;
48      double E_R;
49      double E_RT;
50
51      vector<double> E = easy_ro_input.activation_energy;
52      vector<double> f = easy_ro_input.stoichiometric_factor;
53
54      matrix<int> block_number(ny, nx);
55      matrix<double> T1(ny, nx);
56      matrix<double> T2(ny, nx);
57      matrix<double> heating_rate(nt + 1, nx * ny);
58      matrix<double> I(nt + 1, (nx * ny) * n_components);
59      matrix<double> deltaI(nt + 1, (nx * ny) * n_components);
60      matrix<double> F(nt + 1, (nx * ny) * n_components);
61      matrix<double> cum_rxn(nt + 1, (nx * ny));
62      matrix<double> Ro(nt + 1, nx * ny);
63      matrix<double> TTI(nt + 1, nx * ny);
64
65
66    cout << endl;
67     cout << "==========================================" << endl;
68    cout << "= STARTING EASY%RO CALCULATION =" << endl;
69     cout << "==========================================" << endl;
70
71      mesh.BlockNumbering(easy_ro_input, block_number);
72
73      n = 1;
74      time_step = 0;
75      while (time_step <= nt)
76      {
77          if (time_step < nt)
```

```
78          {
79              io.ReadOutputFile(T1, nx, ny, time_step);
80              io.ReadOutputFile(T2, nx, ny, time_step + 1);
81          }
82
83          for (j = 0; j < ny; ++j)
84          {
85             for (i = 0; i < nx; ++i)
86             {
87                // Avoiding division by zero.
88                if (T2[j][i] == T1[j][i])
89                {
90                   T2[j][i] += 1.0E-5;
91                }
92
93                // Heating rate.
94                if (n < (nt + 1))
95                {
96                    heating_rate[n][block_number[j][i]] = (T2[j][i] - T1
                         [j][i]) / dt;
97                }
98
99                // I.
100               if (time_step == nt)
101               {
102                  T1 = T2;
103               }
104               index = block_number[j][i] * n_components;
105               for (component = 0; component < n_components; ++
                     component)
106               {
107                  E_R = E[component] / R;
108                  E_RT = E_R / T1[j][i];
109                  I[time_step][index] = A * T1[j][i] * exp(-E_RT) * (1
                        - ((pow(E_RT, 2.0) + a1 * E_RT + a2) / (pow(E_RT,
                         2.0) + b1 * E_RT + b2)));
110
111                  index += 1;
112               }
113
114               // Delta I.
115               if (time_step > 0)
116               {
117                  index = block_number[j][i] * n_components;
118                  for (component = 0; component < n_components; ++
                        component)
119                  {
```

```
120                 deltaI[time_step][index] = deltaI[time_step - 1][
                        index] + (I[time_step][index] - I[time_step -
                        1][index]) / heating_rate[time_step][
                        block_number[j][i]];
121                 index += 1;
122             }
123         }
124
125         // Extent of reaction.
126         index = block_number[j][i] * n_components;
127         for (component = 0; component < n_components; ++
                component)
128         {
129             if (deltaI[time_step][index] < 1.0e-20)
130             {
131                 F[time_step][index] = 0.0;
132             }
133             if (deltaI[time_step][index] > 220.0)
134             {
135                 F[time_step][index] = f[component];
136             }
137             else
138             {
139                 F[time_step][index] = f[component] * (1 - exp(-
                        deltaI[time_step][index]));
140             }
141             index += 1;
142         }
143
144         // Cum. rxn, wheighted.
145         index = block_number[j][i] * n_components;
146         for (component = 0; component < n_components; ++
                component)
147         {
148             cum_rxn[time_step][block_number[j][i]] += F[time_step
                    ][index];
149
150             index += 1;
151         }
152
153         // EasyRo%.
154         Ro[time_step][block_number[j][i]] = exp(-1.6 + 3.7 *
                cum_rxn[time_step][block_number[j][i]]);
155
156         if (Ro[time_step][block_number[j][i]] == 0.0)
157         {
158             cout << "Easy%Ro = 0" << endl;
```

```
159                     cout << "Block number: " << block_number[j][i] <<
                            endl;
160                 }
161
162                 // TTI
163                 TTI[time_step][block_number[j][i]] = pow(10.0, (log10(Ro
                        [time_step][block_number[j][i]]) - log10(0.06359)) /
                         0.2012) / 1444.0;
164             }
165         }
166         // Write solution.
167         io.WriteEasyRoOutputFile(time_step, ny, nx, Ro, TTI,
                block_number);
168
169         time_step += 1;
170         n += 1;
171       }
172     }
173 };
174
175 #endif
```

Listing A.4: model.h

```
1  // name OF CLASS: Numerical
2  // CREDIT:
3  // PURPOSE:
4  //     This class implements the numerical models used.
5
6
7  #ifndef NUMERICAL_H
8  #define NUMERICAL_H
9
10 #include "input_struct.h"
11 #include "matrix.h"
12
13 using std::vector;
14
15 class Numerical
16 {
17 public:
18     Numerical(){};
19
20
21     // name OF FUNCTION: ArithmeticMean
22     // CREDIT:
```

```
23  // PURPOSE:
24  //    Receives a variable number of parameters and returns its
        arithmetic mean.
25
26  double ArithmeticMean(int numargs, ...)
27  {
28     // Local variables.
29     vector<double> arg;
30
31     double mean;
32     double sum;
33     double value;
34     double proportion;
35
36     int i;
37
38
39     va_list vl;
40     va_start(vl, numargs);
41
42     sum = 0.0;
43     for (i = 0; i < numargs; ++i)
44     {
45        arg = va_arg(vl, vector<double>);
46        proportion = arg[0];
47        value = arg[1] ;
48        sum += value * proportion;
49     }
50
51      va_end(vl);
52
53     mean = sum;
54
55     return mean;
56  }
57
58
59  // name OF FUNCTION: GeometricMean
60  // CREDIT:
61  // PURPOSE:
62  //    Receives a variable number of parameters and returns its
        geometric mean.
63
64  double GeometricMean(int numargs, ...)
65  {
66     // Local variables.
67     vector<double> arg;
```

```
68
69      double mean;
70      double prod;
71      double value;
72      double proportion;
73
74      int i;
75
76
77      va_list vl;
78      va_start(vl, numargs);
79
80      prod = 1.0;
81      for (i = 0; i < numargs; ++i)
82      {
83          arg = va_arg(vl, vector<double>);
84          proportion = arg[0];
85          value = arg[1];
86          prod *= pow(value, proportion);
87      }
88
89      va_end(vl);
90
91      mean = prod;
92
93      return mean;
94  }
95
96
97  // name OF FUNCTION: HarmonicMean
98  // CREDIT:
99  // PURPOSE:
100 //    Receives a variable number of parameters and returns its
        geometric mean.
101
102 double HarmonicMean(int numargs, ...)
103 {
104     // Local variables.
105       vector<double> arg;
106
107     double mean;
108     double proportion;
109     double inv_sum;
110     double value;
111
112     int i;
113
```

```
114         va_list vl;
115         va_start(vl, numargs);
116 
117         inv_sum = 0.0;
118         for (i = 0; i < numargs ; ++i)
119         {
120             arg = va_arg(vl, vector<double>);
121             proportion = arg[0];
122             value = arg[1];
123             inv_sum += proportion / value;
124         }
125 
126         va_end(vl);
127 
128         mean = 1 / inv_sum;
129 
130         return mean;
131     }
132 
133 
134     // name OF FUNCTION: SquareRootMean
135     // CREDIT:
136     // PURPOSE:
137     //     Receives a variable number of parameters and returns its
            square root mean.
138     //
139 
140     double SquareRootMean(int numargs, ...)
141     {
142         // Local variables.
143           vector<double> arg;
144 
145         double mean;
146         double sqrt_sum;
147         double proportion;
148         double value;
149 
150         int i;
151 
152         va_list vl;
153         va_start(vl, numargs);
154 
155         sqrt_sum = 0.0;
156         for (i = 0; i < numargs ; ++i)
157         {
158             arg = va_arg(vl, vector<double>);
159             proportion = arg[0];
```

```
160             value = arg[1];
161             sqrt_sum += proportion * sqrt(value);
162         }
163 
164         va_end(vl);
165 
166         mean = pow(sqrt_sum, 2.0);
167 
168         return mean;
169     }
170 
171 
172     // name OF FUNCTION: GetNewTemperature
173     // CREDIT:
174     // PURPOSE:
175     //     This function updates the temperature profile.
176 
177     void GetNewTemperature(
178         matrix<double> &rho,
179         matrix<double> &cp,
180         matrix<double> &k_s,
181         matrix<double> &k_n,
182         matrix<double> &k_e,
183         matrix<double> &k_w,
184         matrix<double> &mass,
185         matrix<double> &T,
186         InputVariables &input
187         )
188     {
189 
190         // Local variables
191         int nx = input.number_of_blocks_in_x_direction;
192         int ny = input.number_of_blocks_in_y_direction;
193         double dx = input.dx;
194         double dy = input.dy;
195         double dt = input.dt;
196         double Tfs = input.southern_boundary_prescribed_temperature;
197         double Tfn = input.northern_boundary_prescribed_temperature;
198         double qfe = input.eastern_boundary_prescribed_flux;
199         double qfw = input.western_boundary_prescribed_flux;
200         matrix<double> T_h(ny, nx);
201         vector<double> main_diagonal(nx);
202         vector<double> upper_diagonal(nx - 1);
203         vector<double> lower_diagonal(nx - 1);
204         vector<double> new_temperature(nx);
205         vector<double> main_diagonal_h(ny);
206         vector<double> upper_diagonal_h(ny - 1);
```

```
207         vector<double> lower_diagonal_h(ny - 1);
208         vector<double> new_temperature_h(ny);
209 
210         vector<double> b(nx);
211         vector<double> b_h(ny);
212 
213         int i;
214         int j;
215 
216 
217         // t -> t + 1/2.
218         for (j = 0; j < ny; ++j)
219         {
220             for (i = 0; i < nx; ++i)
221             {
222                 if (i == 0)
223                 {
224                     if (j == 0)
225                     {
226                         b[i] = mass[j][i] * cp[j][i] * T[j][i] + 0.5 * dt *
                                dx * (k_s[j][i] * (T[j + 1][i] - T[j][i]) / (0.5
                                * dy) - k_n[j][i] * (T[j][i] - Tfn) / (0.5 * dy))
                                 + dt * dy * qfe;
227                     }
228                     if (j == ny - 1)
229                     {
230                         b[i] = mass[j][i] * cp[j][i] * T[j][i] + 0.5 * dt *
                                dx * (k_s[j][i] * (Tfs - T[j][i]) / (0.5 * dy) -
                                k_n[j][i] * (T[j][i] - T[j - 1][i]) / (0.5 * dy))
                                 + dt * dy * qfe;
231                     }
232                     if ((j > 0) && (j < ny - 1))
233                     {
234                         b[i] = mass[j][i] * cp[j][i] * T[j][i] + 0.5 * dt *
                                dx * (k_s[j][i] * (T[j + 1][i] - T[j][i]) / dy -
                                k_n[j][i] * (T[j][i] - T[j - 1][i]) / dy) + dt *
                                dy * qfe;
235                     }
236                 }
237 
238                 if (i == nx - 1)
239                 {
240                     if (j == 0)
241                     {
242                         b[i] = mass[j][i] * cp[j][i] * T[j][i] + 0.5 * dt *
                                dx * (k_s[j][i] * (T[j + 1][i] - T[j][i]) / (0.5
                                * dy) - k_n[j][i] * (T[j][i] - Tfn) / (0.5 * dy))
```

```
                - dt * dy * qfw;
243             }
244             if (j == ny - 1)
245             {
246                 b[i] = mass[j][i] * cp[j][i] * T[j][i] + 0.5 * dt *
                        dx * (k_s[j][i] * (Tfs - T[j][i]) / (0.5 * dy) -
                        k_n[j][i] * (T[j][i] - T[j - 1][i]) / (0.5 * dy))
                         - dt * dy * qfw;
247             }
248             if ((j > 0) && (j < ny - 1))
249             {
250                 b[i] = mass[j][i] * cp[j][i] * T[j][i] + 0.5 * dt *
                        dx * (k_s[j][i] * (T[j + 1][i] - T[j][i]) / dy -
                        k_n[j][i] * (T[j][i] - T[j - 1][i]) / dy) - dt *
                        dy * qfw;
251             }
252         }
253
254         if ((i > 0) && (i < nx - 1))
255         {
256             if (j == 0)
257             {
258                 b[i] = mass[j][i] * cp[j][i] * T[j][i] + 0.5 * dt *
                        dx * (k_s[j][i] * (T[j + 1][i] - T[j][i]) / (0.5
                        * dy) - k_n[j][i] * (T[j][i] - Tfn) / (0.5 * dy))
                        ;
259             }
260             if (j == ny - 1)
261             {
262                 b[i] = mass[j][i] * cp[j][i] * T[j][i] + 0.5 * dt *
                        dx * (k_s[j][i] * (Tfs - T[j][i]) / (0.5 * dy) -
                        k_n[j][i] * (T[j][i] - T[j - 1][i]) / (0.5 * dy))
                        ;
263             }
264             if ((j > 0) && (j < ny - 1))
265             {
266                 b[i] = mass[j][i] * cp[j][i] * T[j][i] + 0.5 * dt *
                        dx * (k_s[j][i] * (T[j + 1][i] - T[j][i]) / dy -
                        k_n[j][i] * (T[j][i] - T[j - 1][i]) / dy);
267             }
268         }
269
270         // Main diagonal.
271         if ((i > 0) && (i < nx - 1))
272         {
273             if ((j == 0) || (j == ny - 1))
274             {
```

```
275                 main_diagonal[i] = mass[j][i] * cp[j][i] + 0.5 * dt *
                        (0.5 * dy / dx) * (k_w[j][i] + k_e[j][i]);
276             }
277             else
278             {
279                 main_diagonal[i] = mass[j][i] * cp[j][i] + 0.5 * dt *
                        (dy / dx) * (k_w[j][i] + k_e[j][i]);
280             }
281         }
282         else
283         {
284             if (i == 0)
285             {
286                 if ((j == 0) || (j == ny - 1))
287                 {
288                     main_diagonal[i] = mass[j][i] * cp[j][i] + 0.5 *
                            dt * (0.5 * dy / dx) * k_w[j][i];
289                 }
290                 else
291                 {
292                     main_diagonal[i] = mass[j][i] * cp[j][i] + 0.5 *
                            dt * (dy / dx) * k_w[j][i];
293                 }
294             }
295             if (i == nx - 1)
296             {
297                 if ((j == 0) || (j == ny - 1))
298                 {
299                     main_diagonal[i] = mass[j][i] * cp[j][i] + 0.5 *
                            dt * (0.5 * dy / dx) * k_e[j][i];
300                 }
301                 else
302                 {
303                     main_diagonal[i] = mass[j][i] * cp[j][i] + 0.5 *
                            dt * (dy / dx) * k_e[j][i];
304                 }
305             }
306         }
307
308         // Lower diagonal.
309         if (i > 0)
310         {
311             if ((j == ny - 1) || (j == 0))
312             {
313                 lower_diagonal[i - 1] = - 0.5 * dt * (0.5 * dy / dx)
                        * k_e[j][i];
314             }
```

```
315                 else
316                 {
317                     lower_diagonal[i - 1] = - 0.5 * dt * (dy / dx) * k_e[
                            j][i];
318                 }
319             }
320 
321             // Upper diagonal.
322             if (i < nx - 1)
323             {
324                 if ((j == ny - 1) || (j == 0))
325                 {
326                     upper_diagonal[i] = - 0.5 * dt * (0.5 * dy / dx) *
                            k_w[j][i];
327                 }
328                 else
329                 {
330                     upper_diagonal[i] = - 0.5 * dt * (dy / dx) * k_w[j][i
                            ];
331                 }
332             }
333         }
334 
335         // Solve for the half time step.
336         TridiagonalSolver(new_temperature, lower_diagonal,
                main_diagonal, upper_diagonal, b, nx);
337 
338         for (i = 0; i < nx; ++i)
339         {
340             T_h[j][i] = new_temperature[i];
341         }
342     }
343 
344 
345     // t + 1/2 -> t + 1.
346     for (i = 0; i < nx; ++i)
347     {
348         for (j = 0; j < ny; ++j)
349         {
350             if (j == 0)
351             {
352                 if (i == 0)
353                 {
354                     b_h[j] = mass[j][i] * cp[j][i] * T_h[j][i] + 0.5 * dt
                            * (0.5 * dy) * (k_w[j][i] * (T_h[j][i + 1] - T_h
                            [j][i]) / dx + 2.0 * qfe) + 0.5 * dt * (dx / (0.5
                            * dy)) * k_n[j][i] * Tfn;
```

```
355                 }
356                 if (i == nx - 1)
357                 {
358                     b_h[j] = mass[j][i] * cp[j][i] * T_h[j][i] + 0.5 * dt
                            * (0.5 * dy) * (- 2.0 * qfw - k_e[j][i] * (T_h[j
                        ][i] - T_h[j][i - 1]) / dx) + 0.5 * dt * (dx /
                        (0.5 * dy)) * k_n[j][i] * Tfn;
359                 }
360                 if ((i > 0) && (i < nx - 1))
361                 {
362                     b_h[j] = mass[j][i] * cp[j][i] * T_h[j][i] + 0.5 * dt
                            * (0.5 * dy) * (k_w[j][i] * (T_h[j][i + 1] - T_h
                        [j][i]) / dx - k_e[j][i] * (T_h[j][i] - T_h[j][i
                        - 1]) / dx) + 0.5 * dt * (dx / (0.5 * dy)) * k_n[
                        j][i] * Tfn;
363                 }
364             }
365
366             if (j == ny - 1)
367             {
368                 if (i == 0)
369                 {
370                     b_h[j] = mass[j][i] * cp[j][i] * T_h[j][i] + 0.5 * dt
                            * (0.5 * dy) * (k_w[j][i] * (T_h[j][i + 1] - T_h
                        [j][i]) / dx + 2.0 * qfe) + 0.5 * dt * (dx / (0.5
                            * dy)) * k_s[j][i] * Tfs;
371                 }
372                 if (i == nx - 1)
373                 {
374                     b_h[j] = mass[j][i] * cp[j][i] * T_h[j][i] + 0.5 * dt
                            * (0.5 * dy) * (- 2.0 * qfw - k_e[j][i] * (T_h[j
                        ][i] - T_h[j][i - 1]) / dx) + 0.5 * dt * (dx /
                        (0.5 * dy)) * k_s[j][i] * Tfs;
375                 }
376                 if ((i > 0) && (i < nx - 1))
377                 {
378                     b_h[j] = mass[j][i] * cp[j][i] * T_h[j][i] + 0.5 * dt
                            * (0.5 * dy) * (k_w[j][i] * (T_h[j][i + 1] - T_h
                        [j][i]) / dx - k_e[j][i] * (T_h[j][i] - T_h[j][i
                        - 1]) / dx) + 0.5 * dt * (dx / (0.5 * dy)) * k_s[
                        j][i] * Tfs;
379                 }
380             }
381
382             if ((j > 0) && (j < ny - 1))
383             {
384                 if (i == 0)
```

```
385                 {
386                     b_h[j] = mass[j][i] * cp[j][i] * T_h[j][i] + 0.5 * dt
                            * dy * (k_w[j][i] * (T_h[j][i + 1] - T_h[j][i])
                        / dx + 2.0 * qfe);
387                 }
388                 if (i == nx - 1)
389                 {
390                     b_h[j] = mass[j][i] * cp[j][i] * T_h[j][i] + 0.5 * dt
                            * dy * (- 2.0 * qfw - k_e[j][i] * (T_h[j][i] -
                        T_h[j][i - 1]) / dx);
391                 }
392                 if ((i > 0) && (i < nx - 1))
393                 {
394                     b_h[j] = mass[j][i] * cp[j][i] * T_h[j][i] + 0.5 * dt
                            * dy * (k_w[j][i] * (T_h[j][i + 1] - T_h[j][i])
                        / dx - k_e[j][i] * (T_h[j][i] - T_h[j][i - 1]) /
                        dx);
395                 }
396             }
397
398             // Storing the diagonals.
399             // Main diagonal.
400             if ((j > 0) && (j < ny - 1))
401             {
402                 main_diagonal_h[j] = mass[j][i] * cp[j][i] + 0.5 * dt *
                        (dx / dy) * (k_s[j][i] + k_n[j][i]);
403             }
404             else
405             {
406                 if (j == 0)
407                 {
408                     main_diagonal_h[j]  = 1.0;
409                     b_h[j] = Tfn;
410                 }
411                 if (j == ny - 1)
412                 {
413                     main_diagonal_h[j]  = 1.0;
414                     b_h[j] = Tfs;
415                 }
416             }
417
418             // Lower diagonal.
419             if (j > 0)
420             {
421                 if (j == ny - 1)
422                 {
423                     lower_diagonal_h[j - 1] = 0.0;
```

```
424                 }
425                 else
426                 {
427                     lower_diagonal_h[j - 1] = - 0.5 * dt * (dx / dy) *
                        k_n[j][i];
428                 }
429             }
430 
431             // Upper diagonal.
432             if (j < ny - 1)
433             {
434                 if (j == 0)
435                 {
436                     upper_diagonal_h[j] = 0.0;
437                 }
438                 else
439                 {
440                     upper_diagonal_h[j] = - 0.5 * dt * (dx / dy) * k_s[j
                        ][i];
441                 }
442             }
443         }
444         // Solve for the half time step.
445         TridiagonalSolver(new_temperature_h, lower_diagonal_h,
            main_diagonal_h, upper_diagonal_h, b_h, ny);
446 
447         for (j = 0; j < ny; ++j)
448         {
449            T[j][i] = new_temperature_h[j];
450         }
451     }
452 }
453 
454 
455 // name OF FUNCTION: TridiagonalSolver
456 // CREDIT:
457 // PURPOSE:
458 //    This function solves the linear tridiagonal system of
       equations by applaying TDMA or,
459 //    most known as, Thomas Algorithm.
460 
461 void TridiagonalSolver(
462     vector<double> &new_temperature,
463     vector<double> &lower_diagonal,
464     vector<double> &main_diagonal,
465     vector<double> &upper_diagonal,
466     vector<double> &independent_term,
```

```
467         int number_of_blocks
468         )
469     {
470         int n = number_of_blocks;
471         int indice = 0;
472         int i;
473 
474         for (i = 1; i < n; ++i)
475         {
476             main_diagonal[i] = main_diagonal[i] - upper_diagonal[i - 1] *
                    (lower_diagonal[indice] / main_diagonal[i - 1]);
477             independent_term[i] = independent_term[i] - independent_term[i
                     - 1] * (lower_diagonal[indice] / main_diagonal[i - 1]);
478             indice += 1;
479         }
480 
481         // Back substitution.
482         new_temperature[n - 1] = independent_term[n - 1] / main_diagonal[
                n - 1];
483         for (i = n - 2; i >= 0; --i)
484         {
485             new_temperature[i] = (independent_term[i] - upper_diagonal[i]
                    * new_temperature[i + 1]) / main_diagonal[i];
486         }
487     }
488 };
489 
490 #endif
```

Listing A.5: numerical.h

```
1  // name OF CLASS: PropertyDistribution
2  // CREDIT:
3  // PURPOSE:
4  //     This class defines the properties distribution.
5  
6  
7  #ifndef PROPERTY_DISTRIBUTION_H
8  #define PROPERTY_DISTRIBUTION_H
9  
10 #include "input_struct.h"
11 
12 
13 class PropertyDistribution
14 {
15 public:
```

```
16    PropertyDistribution(){};
17
18
19    void CalculateInterfacialThermalConductivity(
20       matrix<double> &k,
21       matrix<double> &k_s,
22       matrix<double> &k_n,
23       matrix<double> &k_e,
24       matrix<double> &k_w,
25       matrix<double> &anisotropy,
26       InputVariables &input
27       )
28    {
29       // Local variables.
30       int nx = input.number_of_blocks_in_x_direction;
31       int ny = input.number_of_blocks_in_y_direction;
32
33       // Counters.
34       int i;
35       int j;
36
37
38       // Condutivity calculated at the interfaces.
39       for (j = 0; j < ny; ++j)
40       {
41          k_e[j][0] = k[j][0];
42          k_w[j][nx - 1] = k[j][nx - 1];
43       }
44       for (i = 0; i < nx; ++i)
45       {
46          k_n[0][i] = anisotropy[0][i] * k[0][i];
47          k_s[ny - 1][i] = anisotropy[ny - 1][i] * k[ny - 1][i];
48       }
49
50       for (j = 0; j < ny; ++j)
51       {
52          for (i = 0; i < (nx - 1); ++i)
53          {
54             k_w[j][i] = 2.0 * ((k[j][i + 1] * k[j][i]) / (k[j][i + 1] +
                   k[j][i]));
55          }
56       }
57       for (i = 0; i < nx; ++i)
58       {
59          for (j = 0; j < (ny - 1); ++j)
60          {
61             k_s[j][i] = 2.0 * ((anisotropy[j + 1][i] * k[j + 1][i] *
```

```
                anisotropy[j][i] * k[j][i]) / (anisotropy[j +  1][i] *
                k[j + 1][i] + anisotropy[j][i] * k[j][i]));
62          }
63       }
64
65       for (j = 0; j < ny; ++j)
66       {
67          for (i  = 1; i < nx; ++i)
68          {
69             k_e[j][i] = k_w[j][i - 1];
70          }
71       }
72       for (i = 0; i < nx; ++i)
73       {
74          for (j  = 1; j < ny; ++j)
75          {
76             k_n[j][i] = k_s[j - 1][i];
77          }
78       }
79    }
80
81
82    void InitialTemperatureDistribution(
83       matrix<double> &initial_temperature,
84       matrix<bool> &magmatic_intrusion,
85       InputVariables &input
86       )
87    {
88       // Local variables.
89       double dx = input.dx;
90       double dy = input.dy;
91       double Tfs = input.southern_boundary_prescribed_temperature;
92       double Tfn = input.northern_boundary_prescribed_temperature;
93       double geothermal_grad = input.geothermal_gradient;
94       double magma_init_temp = input.magma_init_temp;
95
96       int nx = input.number_of_blocks_in_x_direction;
97       int ny = input.number_of_blocks_in_y_direction;
98       int i;
99       int j;
100
101
102       // Initial temperature distribution.
103       for (j = 0; j < ny; ++j)
104       {
105          for (i = 0; i < nx; ++i)
106          {
```

```
107             if (j == 0)
108             {
109                 initial_temperature[j][i] = Tfn;
110             }
111             if (j == ny - 1)
112             {
113                 initial_temperature[j][i] = Tfs;
114             }
115             if ((j > 0) && (j < ny - 1))
116             {
117                 //initial_temperature[j][i] = 298.0;
118                 initial_temperature[j][i] = initial_temperature[j - 1][i
                        ] + geothermal_grad * dy;
119             }
120         }
121     }
122
123     //magma_init_temp = 298.0;
124     for (j = 0; j < ny; ++j)
125     {
126         for (i = 0; i < nx; ++i)
127         {
128             if (magmatic_intrusion[j][i])
129             {
130                 initial_temperature[j][i] = magma_init_temp;
131                 //initial_temperature[j][i] = 298;
132             }
133         }
134     }
135 }
136
137
138 bool MagmaSolidificationCheck(
139  matrix<double> &rho,
140     matrix<double> &cp,
141     matrix<double> &k,
142     matrix<double> &anisotropy,
143     matrix<double> &T,
144     matrix<bool> &magmatic_intrusion,
145     InputVariables &input
146     )
147 {
148     // Local variables.
149     bool solidification = false;
150
151     double rho_diabase = input.diabase_density;
152     double cp_diabase = input.diabase_specific_heat_capacity;
```

```
153       double k_diabase = input.diabase_condutivity;
154      double diabase_anisotropy = input.diabase_anisotropy;
155       double magma_solidification_temperature = input.magma_solid_temp;
156
157       int nx = input.number_of_blocks_in_x_direction;
158       int ny = input.number_of_blocks_in_y_direction;
159       int i;
160       int j;
161
162
163       for (j = 0; j < ny; ++j)
164       {
165          for (i = 0; i < nx; ++i)
166          {
167             if (magmatic_intrusion[j][i])
168             {
169                if(T[j][i] < magma_solidification_temperature)
170                {
171                   solidification = true;
172                   magmatic_intrusion[j][i] = false;
173                   rho[j][i] = rho_diabase;
174                   cp[j][i] = cp_diabase;
175                   k[j][i] = k_diabase;
176                   anisotropy[j][i] = diabase_anisotropy;
177                }
178
179             }
180          }
181       }
182
183       return solidification;
184    }
185
186
187    void CalculateMass(
188       matrix<double> &rho,
189       matrix<double> &mass,
190       InputVariables &input
191       )
192    {
193       // Local variables.
194       double dx = input.dx;
195       double dy = input.dy;
196
197       int nx = input.number_of_blocks_in_x_direction;
198       int ny = input.number_of_blocks_in_y_direction;
199       int i;
```

```
200         int j;
201
202
203         for (j = 0; j < ny; ++j)
204         {
205             for (i = 0; i < nx; ++i)
206             {
207                 mass[j][i] = rho[j][i] * dx * dy * 1.0;
208             }
209         }
210     }
211
212
213 };
214
215 #endif
```

Listing A.6: property_distribution.h

```
1  // Headers.
2  #pragma once
3  // Standard header.
4  #include "targetver.h"
5  #include <stdio.h>
6  #include <stdarg.h>
7  #include <tchar.h>
8  #include <iostream>
9  #include <cmath>
10 #include <cstdio>
11 #include <string>
12 #include <fstream>
13 #include <sstream>
14 #include <vector>
15 #include <assert.h>
16 #include <iomanip>
17 #include <stdexcept>
18 // User defined header.
19 #include "mesh.h"
20 #include "model.h"
21 #include "matrix.h"
22 #include "numerical.h"
23 #include "input_struct.h"
24 #include "simulation_io.h"
25 #include "display_vector.h"
26 //#include "functions_prototypes.h"
27 #include "property_distribution.h"
```

```
28
29 // Consts.
30 const double PI = 3.14159265;
31
32 // Using standard namespace.
33 using namespace std;
34
35
36 /*-------------------------------------------------------------------*/
37
38
39 /*MAIN*/
40
41 // Main function.
42 int _tmain(int argc, _TCHAR* argv[])
43 {
44     /*Instances*/
45     // Mesh object.
46     Mesh mesh;
47     // Property object.
48     PropertyDistribution prop;
49     // IO object.
50     SimulationIO io;
51     // Maturarion model object.
52     MaturationModel model;
53     // Numerical object.
54     Numerical num;
55
56     // Example:
57     //double mean1 = num.ArithmeticMean(5, 1.0, 2.0, 3.0, 4.0, 5.0);
58     //double mean2 = num.GeometricMean(5, 1.0, 2.0, 3.0, 4.0, 5.0);
59     //double mean3 = num.HarmonicMean(5, 1.0, 2.0, 3.0, 4.0, 5.0);
60     //double mean4 = num.SquareRootMean(5, 1.0, 2.0, 3.0, 4.0, 5.0);
61
62
63     /*Variables Declaration*/
64
65     cout << "==================" << endl;
66    cout << "= READING DATA =" << endl;
67     cout << "==================" << endl;
68
69     // Read input variables.
70     InputVariables input;
71      io.GetData(input);
72
73     // Test properties.
74     bool THERMALTEST;
```

```
75    THERMALTEST = false;
76     if (THERMALTEST)
77    {
78       input.number_of_blocks_in_y_direction = 50;
79       input.number_of_blocks_in_x_direction = 50;
80       //input.number_of_blocks_in_y_direction = 51;
81       //input.number_of_blocks_in_x_direction = 51;
82       input.length_y_direction = 10.0;
83       input.lenght_x_direction = 10.0;
84       input.dx = input.lenght_x_direction / static_cast<double>(input.
             number_of_blocks_in_x_direction) * 1000.0;
85       input.dy = input.length_y_direction / static_cast<double>(input.
             number_of_blocks_in_y_direction) * 1000.0;
86     input.geothermal_gradient = (input.
          southern_boundary_prescribed_temperature - input.
          northern_boundary_prescribed_temperature) / (input.
          length_y_direction * 1000.0 - input.dy);
87       //input.southern_boundary_prescribed_temperature = input.
             northern_boundary_prescribed_temperature;
88       //input.geothermal_gradient = 0.0;
89       input.number_of_time_steps = 100;
90       input.total_simulation_time = 25.0 * 1.0e+3 * 365.25 * 24.0 *
             60.0 * 60.0;
91       input.dt = input.total_simulation_time / static_cast<double>(
             input.number_of_time_steps);
92    }
93
94    bool EASYROTEST;
95    EASYROTEST = false;
96     if (EASYROTEST)
97    {
98       input.number_of_blocks_in_y_direction = 1;
99       input.number_of_blocks_in_x_direction = 1;
100      input.length_y_direction = 1.0;
101      input.lenght_x_direction = 1.0;
102      input.dx = input.lenght_x_direction / static_cast<double>(input.
             number_of_blocks_in_x_direction * 1000.0);
103      input.dy = input.length_y_direction / static_cast<double>(input.
             number_of_blocks_in_y_direction * 1000.0);
104      input.number_of_time_steps = 12;
105      input.total_simulation_time = 240.0 * 1.0e+6 * 365.25 * 24.0 *
             60.0 * 60.0;
106      input.dt = input.total_simulation_time / static_cast<double>(
             input.number_of_time_steps);
107   }
108
109
```

```
110     // Notation simplification.
111     int nx = input.number_of_blocks_in_x_direction;
112     int ny = input.number_of_blocks_in_y_direction;
113     int nt = input.number_of_time_steps;
114     int nt_m = 250;
115 
116     double Tfs = input.southern_boundary_prescribed_temperature;
117     double Tfn = input.northern_boundary_prescribed_temperature;
118     double dx = input.dx;
119     double dy = input.dy;
120     double dt = input.dt;
121     double dt_m = input.dt / static_cast<double>(nt_m);
122 
123     bool calculate_easy_ro = input.calculate_easy_ro;
124     bool magma_solidification;
125     bool time_step_break = true;
126 
127     // Variables.
128     matrix<double> T(ny, nx);
129     matrix<double> initial_temperature(ny, nx);
130     matrix<double> mass(ny, nx);
131     matrix<double> rho(ny, nx);
132     matrix<double> cp(ny, nx);
133     matrix<double> k(ny, nx);
134     matrix<double> k_e(ny, nx);
135     matrix<double> k_w(ny, nx);
136     matrix<double> k_n(ny, nx);
137     matrix<double> k_s(ny, nx);
138     matrix<double> anisotropy(ny, nx);
139     matrix<bool> magmatic_intrusion(ny, nx);
140 
141     double time;
142 
143     // Counters.
144     int n;
145     int m;
146 
147 
148     /*Create grid*/
149 
150     // Get mesh.
151     if (THERMALTEST)
152     {
153         mesh.TestMesh(magmatic_intrusion, rho, cp, k, anisotropy, input);
154     }
155     if (EASYROTEST)
156     {
```

```
157      rho[0][0] = input.sandstone_density ;
158        cp[0][0] = input.sandstone_specific_heat_capacity;
159        k[0][0] = input.sandstone_condutivity;
160     }
161     if (!THERMALTEST && !EASYROTEST)
162     {
163         mesh.SolimoesBasinMesh(magmatic_intrusion, rho, cp, k,
                anisotropy, input);
164         io.WriteOutputFile(0, ny, nx, 0, rho);
165     }
166     // Mesh testing.
167     mesh.NonNegativeCoefficienteTest(rho, cp, k, input);
168
169
170      /*Property initialization*/
171
172     // Interfacial condutivities.
173     prop.CalculateInterfacialThermalConductivity(k, k_s, k_n, k_e, k_w,
            anisotropy, input);
174     // Mass calculation.
175      prop.CalculateMass(rho, mass, input);
176     // Initial temperature distribution.
177     if (EASYROTEST)
178     {
179          initial_temperature[0][0] = 293.0;
180     }
181     else
182     {
183         prop.InitialTemperatureDistribution(initial_temperature,
                magmatic_intrusion, input);
184     }
185
186
187     /*Assembly*/
188
189     cout << endl;
190     cout << "========================" << endl;
191    cout << "= STARTING SIMULATION =" << endl;
192     cout << "========================" << endl;
193
194     // Initial condition.
195     n = 0;
196     time = 0;
197     T = initial_temperature;
198
199     printf ("time: %.4f My\n", time / (1.0e+3*365.25*24.0*60.0*60.0));
200
```

```
201     //HeatEquation1DAnalyticalSolution(time, n, rho, cp, k, input);
202
203     // Writing initial outputs to a data file.
204     io.WriteOutputFile(n, ny, nx, time, T);
205
206     // Big loop in time.
207     n = 1;
208     while (n <= nt)
209     {
210         if (EASYROTEST)
211          {
212             if (n == 1)
213             {
214                 T[0][0] = 333.0;
215             }
216             if (n == 2)
217             {
218                 T[0][0] = 353.0;
219             }
220             if (n == 3)
221             {
222                 T[0][0] = 373.0;
223             }
224              if (n == 4)
225             {
226                 T[0][0] = 393.0;
227             }
228             if (n == 5)
229             {
230                 T[0][0] = 413.0;
231             }
232             if (n == 6)
233             {
234                 T[0][0] = 373.0;
235             }
236             if (n == 7)
237             {
238                 T[0][0] = 333.0;
239             }
240             if (n == 8)
241             {
242                 T[0][0] = 333.0;
243             }
244             if (n == 9)
245             {
246                 T[0][0] = 363.0;
247             }
```

```
248             if (n == 10)
249             {
250                 T[0][0] = 393.0;
251             }
252             if (n == 11)
253             {
254                 T[0][0] = 423.0;
255             }
256             if (n == 12)
257             {
258                 T[0][0] = 453.0;
259             }
260         }
261         else
262 
263         {
264             if (n == 1 && time_step_break)
265             {
266                 m = 0;
267                 while (m < nt_m)
268                 {
269                     input.dt = dt_m;
270                     num.GetNewTemperature(rho, cp, k_s, k_n, k_e, k_w, mass,
                            T, input);
271                     ++m;
272                 }
273             }
274             else
275             {
276                 input.dt = dt;
277                  num.GetNewTemperature(rho, cp, k_s, k_n, k_e, k_w, mass, T
                        , input);
278             }
279         }
280 
281         // Check if the magma changed its phase.
282         magma_solidification =  prop.MagmaSolidificationCheck(rho, cp, k,
              anisotropy, T, magmatic_intrusion, input);
283         if (magma_solidification)
284         {
285             // Recalculate the interfacial thermal conductivities.
286             prop.CalculateInterfacialThermalConductivity(k, k_s, k_n, k_e,
                  k_w, anisotropy, input);
287         }
288 
289         // Writing outputs to a data file.
290         io.WriteOutputFile(n, ny, nx, time, T);
```

```
291
292         ++n;
293
294         time += dt;
295
296         printf ("time: %.4f My\n", time / (1.0e+3*365.25*24.0*60.0*60.0))
                ;
297     }
298
299     if (calculate_easy_ro)
300     {
301         // Easy%Ro calculation.
302         model.EasyRoCalculation(input, io, mesh);
303     }
304
305      return 0; // End of the program.
306 }
```

Listing A.7: thermal_conductivity_2D.cpp