UNIVERSIDADE FEDERAL FLUMINENSE

AUGUSTO GARCIA ALMEIDA

Aspectos computacionais do cálculo de dispersão de calor em meios porosos

NITERÓI

2007

UNIVERSIDADE FEDERAL FLUMINENSE

AUGUSTO GARCIA ALMEIDA

Aspectos computacionais do cálculo de dispersão de calor em meios porosos

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Modelagem Computacional.

Orientador:
Otton da Silveira Teixeira

NITERÓI

2007

Aspectos computacionais do cálculo de dispersão de calor em meios porosos

Augusto Garcia Almeida

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre.

Aprovada por:

Prof. D.Sc. Otton Teixeira da Silveira Filho / IC-UFF
(Presidente)

Prof. D.Sc. Hélio Pedro Amaral Souto / IPRJ-UERJ

Prof. D.Sc. Mauricio Kischinhevsky / IC-UFF

Niterói, 19 de Setembro de 2007.

Dedico esta obra aos meus pais e amigos encarnados e desencarnados que sempre me apoiaram.

Agradecimentos

Em primeiro lugar gostaria de agradecer a Deus pela existencia do tudo, agradeço aos meus pais e a minha família por tudo que tem feito e continuam fazendo por mim, agradeço ao meu orientador Otton Teixeira pela enorme paciência e dedicação que teve comigo ao longo dessa jornada, ao Diretor do IC Maurício Kischinhevsky que sempre foi e continua sendo uma grande fonte de sabedoria, a chefe da modelagem computacional Regina Leal que foi mais que professora foi uma grande conselheira principalmente nos momentos de dificuldade, aos meus grandes antigos e novos amigos Edgar Barbosa, Kennedy Moraes, Márcio Belleza, Ildenir Costa, Diego Brandão, Sanderson Gonzaga, Ebert Viard, Fernando Silva, Rogério Gomes, Luiz Valente, Warley Gramacho, Marcelo Marzan, Alexandre Antunes, Alex Machado, Esteban Clua, Luiz Felipe Santana e a todos que tive o grande prazer de conhecer na UFF e UFRRJ e que estiveram ao meu lado nessa jornada e espero que nas próximas também, ao meu amigo Arnaldo e seus pais pelas várias vezes que cederam sua casa em Piratininga que foi de grande ajuda, agradeço também ao grande Jacques Silva que é sem sombra de dúvidas uma das pessoas mais humildes e prestativas que conheço, as secretárias Angela e Maria que sempre me trataram bem e sempre foram muito atenciosas, ao corpo que compõem o colegiado que sempre atendeu aos meus pedidos, não posso de maneira alguma de deixar de agradecer aos meus mais que companheiros de trabalho e sim amigos Waldomiro Neves e Angel Ramon da UFRRJ pelas cartas de recomendação e apoio, aproveitando agradeço também aos meus grandiosos amigos da UFRRJ o diretor do ICE Miguel Angelo, a chefe do DEMAT Rosane Ferreira e Maria Teresa, ao coordenador do curso de matemática Paulo Parga, e a todos docentes, discentes e técnicos administrativos da matemática da UFRRJ, agradeço também a Capes pelo financiamento deste trabalho e deixando claro que o incentivo a produção científica é imprescindível, a todos eu digo que nada disso seria possível sem a participação direta ou indireta, obrigado de coração e que Deus os presenteie da mesma forma que fizeram comigo!

Resumo

Neste trabalho são analisados alguns aspectos computacionais de uma implementação do cálculo de dispersão de calor em um meio poroso periódico, indeformável e tridimensional.

Palavras-chave: Modelagem Computacional, Dispersão Calor, Método dos Térmions

Abstract

This work examines some aspects of a computational implementation of the calculation of dispersion of heat in a periodic, undeformable and three-dimensional porous media.

Keywords: Computational Modelling, Heat Dispersion, Termions Method

Sumário

Lista de Figuras

Capítulo 1

Introdução

O estudo dos fenômenos de transporte em meios porosos é estimulado por áreas que vão da indústria do petróleo até a medicina, passando por temas como dispersão de poluentes, secagem de madeira, escoamento de águas subterrâneas, trocadores de calor e conversores catalíticos, absorção de medicamentos pela pele ou pelas vias aéreas [1]. No nosso meio ambiente existe um quantidade muito grande de meios porosos e é por esse motivo que seu estudo é importante.

Iniciaremos este capítulo apresentando alguns aspectos básicos e históricos sobre os meios porosos. Para maiores detalhes ver a bibliografia [2].

1.1 O Meio Poroso

Um meio poroso é uma porção de espaço ocupada por matéria hetereogênea ou multifásica. Pelo menos umas destas fases não é sólida, podendo ser líquida ou gasosa. A parte sólida é denominada *matriz sólida* e o espaço que não faz parte da matriz sólida é denominado *espaço vazio*. Se porções do espaço vazio são interconectadas umas às outras porções, este espaço é denominado *espaço vazio efetivo*. Dizemos ainda que este espaço constitui-se dos *poros* do meio podendo haver, em geral, vários caminhos de conexão entre os poros.

A área de contato entre o sólido e o fluido do meio é geralmente alta e os poros são geralmente estreitos em relação às dimensões do sólido. Uma representação esquemática ilustrativa de um meio poroso é dada na figura 1.1.

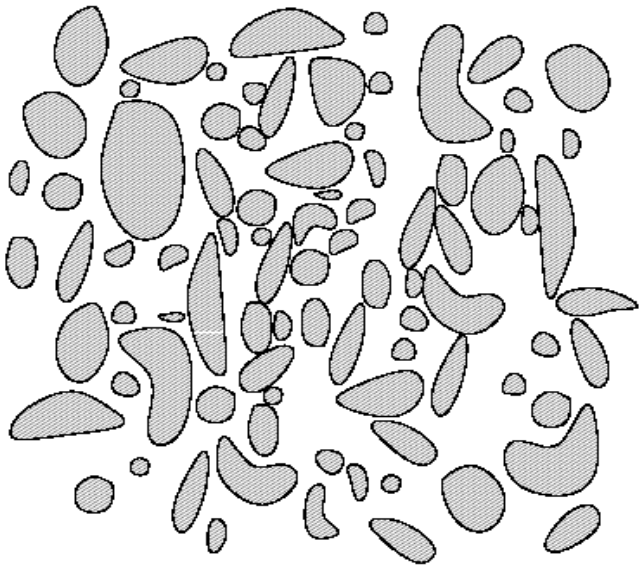

Figura 1.1: Representação esquemática de um meio poroso.

1.1.1 Classificação de um Meio Poroso

Sob o ponto de vista estrutural podemos classificar o meio poroso em relação à homogeneidade, isotropia e ordenação. Por homogeneidade entendemos que o meio é invariante por translação. A isotropia implica na sua invariância por rotação em torno de um determinado ponto, além disso um meio poroso pode ser ordenado ou desordenado. Os primeiros se caracterizam pela disposição regular dos elementos sólidos e os segundos pela disposição irregular ou aleatória dos sólidos. Quanto à estabilidade do meio, tomando-se como referência a matriz sólida, podemos classificar o meio poroso em consolidados e não consolidados, ou seja, ser consolidado significa que existe uma rigidez que mantém a configuração dos poros, por exemplo concreto, osso, pedra-pomes, e não consolidado essa rigidez é menor, por exemplo esponja, areia.

1.1.2 O Meio Poroso como um Contínuo

No intuito de construirmos modelos para o meio poroso, faremos uma analogia entre a hipótese do contínuo para fluidos na mecânica clássica e o seu equivalente em se tratando de um meio poroso. No caso dos fluidos, trabalhar no *nível molecular* é impraticável dado o número elevado de moléculas para suas porções macroscópicas (1 mol contém 10^{23} moléculas). Usamos, então, uma abstração: consideraremos um *volume elementar* do fluido que seja grande o suficiente para conter uma quantidade considerável de moléculas (tal que suas dimensões ainda sejam muito maiores que o livre caminho médio das moléculas

constituintes) mas muito pequena em relação ao volume total do fluido estudado. Neste ponto, introduzimos a definição de uma determinada propriedade ϕ do fluido num ponto P como

$$\phi(P) = \lim_{\Delta V_i \to \Delta V_0} \phi_i \tag{1.1}$$

onde ΔV_i é o volume cujo centróide é o ponto P e ΔV_0, denominado *ponto material*, é o volume do *volume elementar* descrito acima. Examinando-se esta hipótese, vemos que um grande número de moléculas podem estar colidindo entre si e entrando e saindo do ΔV_0 num determinado instante arbitrário. Digamos que tais fenômenos ocorrem num intervalo de tempo Δt_0. Suporemos que este intervalo de tempo seja muito pequeno em consideração às escalas de tempo nas quais trabalharemos, mas maior que o tempo médio entre colisões as quais, em último caso, são responsáveis pela dinâmica do processo. A estas escalas de espaço e tempo denominaremos de *nível microscópico*.

Por exemplo, se a propriedade ϕ é a massa específica, partindo de escalas moleculares teríamos tipicamente o comportamento mostrado no gráfico da figura 1.2 onde passamos

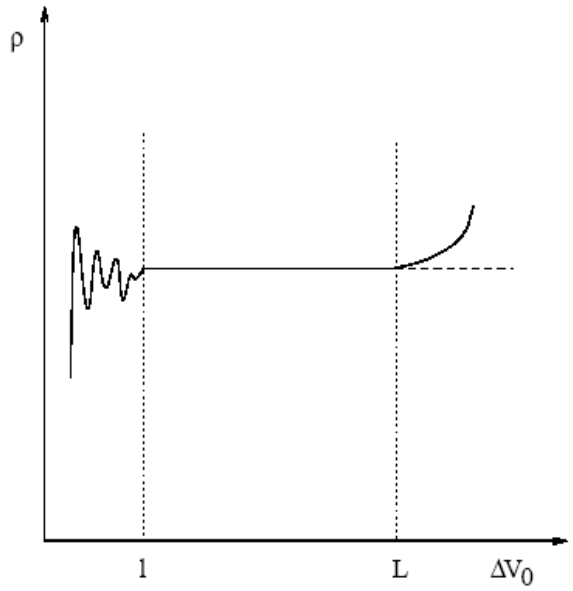

Figura 1.2: Massa Específica de um fluido.

da região de individualização molecular para uma na qual vale a hipótese do contínuo até atingirmos dimensões onde se manifestam as heterogeniedades macroscópicas.

Nesta escala poderíamos usar as equações da mecânica clássica para resolver o problema do escoamento do fluido dentro do meio poroso. No entanto, neste caso, sabemos que teremos novas dificuldades pois a estrutura da matriz sólida é geralmente complexa gerando problemas em relação às condições de contorno.

Como uma saída para este impasse, partiremos para a definição de uma outra escala espacial que denominaremos *escala macroscópica*, na qual usaremos um desenvolvimento análogo ao usado na hipótese do contínuo para fluidos.

1.1.3 Volume Elementar Representativo

Um ponto crucial no desenvolvimento anterior foi a definição de ponto material. Analogamente, definiremos um *Volume Elementar Representativo* (VER) como um volume do meio poroso que seja grande o suficiente para conter elementos do meio poroso que sejam representativos do meio como um todo, e pequeno o suficiente em comparação às dimensões do meio poroso. Representamos esquematicamente o VER na figura 1.3. Devemos entendê-lo como um porção representativa do meio, de forma que ao efetuarmos translações não haja mudanças significativas de suas propriedades, contudo é possível haver meios para os quais o VER não exista. Por exemplo, a porosidade do meio (mais precisamente porosidade volumétrica) seria definida como

$$\varepsilon(P) = \lim_{\Delta\mathcal{V}_i \to \Delta\mathcal{V}_0} \frac{\Delta\mathcal{V}_i}{\Delta V_i} \tag{1.2}$$

onde V_i, $\Delta\mathcal{V}_i$ e $\Delta\mathcal{V}_0$ são respectivamente um volume de uma porção do meio poroso, o volume de espaço vazio dentro de V_i e finalmente o volume do VER ou ponto material do meio poroso.

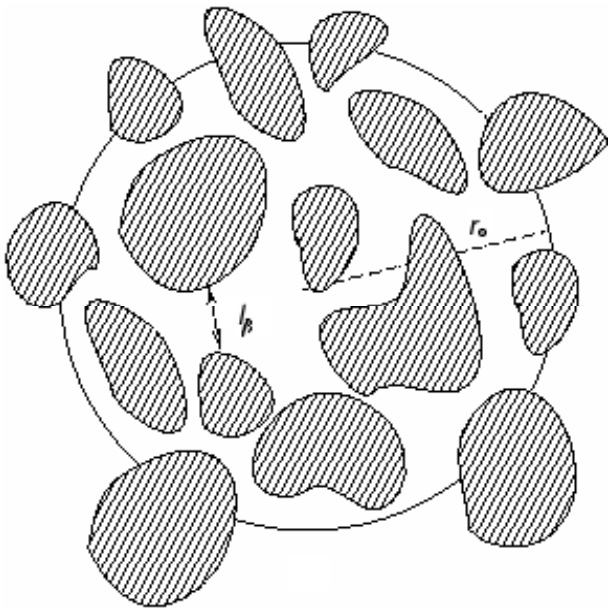

Figura 1.3: Volume elementar representativo.

Neste trabalho, as seguintes dimensões características serão utilizadas: l, l_β, r_o e

L. Elas representam respectivamente as dimensões microscópicas (no nível dos poros), as dimensões dos poros, o "raio" do volume elementar representativo, e as dimensões macroscópicas (relativa às dimensões do meio).

Apresentadas estas dimensões, podemos definir as restrições impostas às escalas de comprimento no sentido de que o VER, caso ele exista, satisfaça as condições descritas anteriormente

$$l_\beta << r_o << L \tag{1.3}$$

Com esta definição de separação de escalas, ocultamos a natureza descontínua do meio poroso e as flutuações estatísticas devidas a esta natureza. Considerando-se, por exemplo, a porosidade ε, a figura 1.4 nos fornece a sua variação à medida que mudamos a escala na qual trabalhamos.

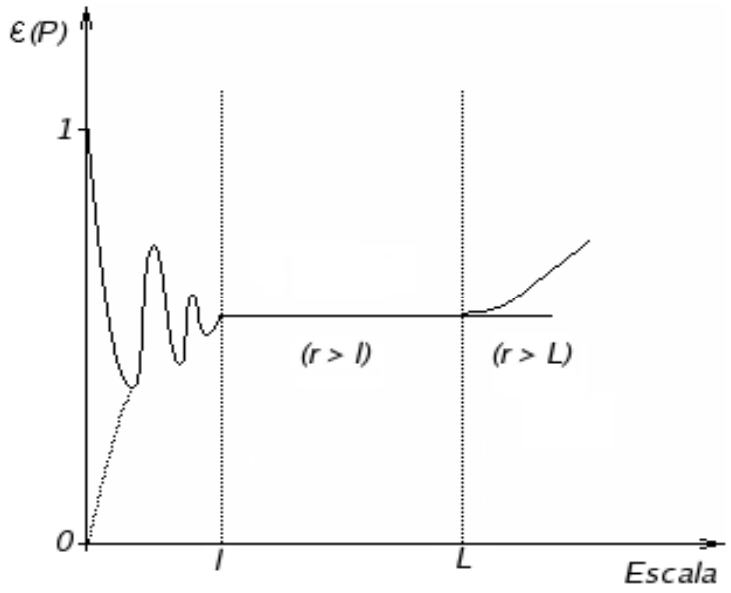

Figura 1.4: Porosidade.

Nesta representação, vemos o valor da porosidade partindo de 1 ou 0 (respectivamente, partindo de um ponto dentro da fase líquida ou do sólida) e sofrendo as flutuações devidas à não homogeneidade local, até atingir a região $(r > \ell)$ para a qual o volume elementar representativo existe e o seu valor passa a ser constante. A partir de certo ponto $(r > L)$, caso o meio seja macroscopicamente heterogêneo, voltaremos a ter variações da porosidade com a posição.

Quanto às escalas de tempo, são tipicamente definidas por ${l_\beta}^2/\alpha$, ${r_0}^2/\alpha$ e L^2/α para o caso de difusão de calor e ainda l_β/u_0, r_0/u_0 e L/u_0 associadas ao tempo de residência do fluido nas diversas escalas de comprimento. Aqui α e u_0 são respectivamente a difu-

sibilidade do meio e a velocidade do fluido. Ambas as escalas de espaço e tempo podem variar numa ampla gama de valores [3].

1.1.4 Porosidade e Superfície Específica

A porosidade volumétrica ε é uma propriedade macroscópica do meio poroso, sendo definida por

$$\varepsilon(P) = \frac{V_\beta}{V} = \frac{V - V_\sigma}{V} \tag{1.4}$$

onde V_β é o volume do fluido, V_σ é o volume do sólido e V o volume total do meio poroso.

Observemos que esta definição leva em conta os poros que não estão conectados. Se considerarmos apenas os poros onde os fluidos podem circular, então teremos a definição da *porosidade efetiva* ε_e

$$\varepsilon_e(P) = \frac{V_\beta}{V_\sigma} \tag{1.5}$$

Mais um parâmetro a considerar é a *superfície específica* a_v, definida como sendo a razão entre a área total da superfície do meio poroso $A_{\beta\sigma}$ e o volume total de sólido V_σ, ou seja,

$$a_v = \frac{A_{\sigma\beta}}{V_\sigma} \tag{1.6}$$

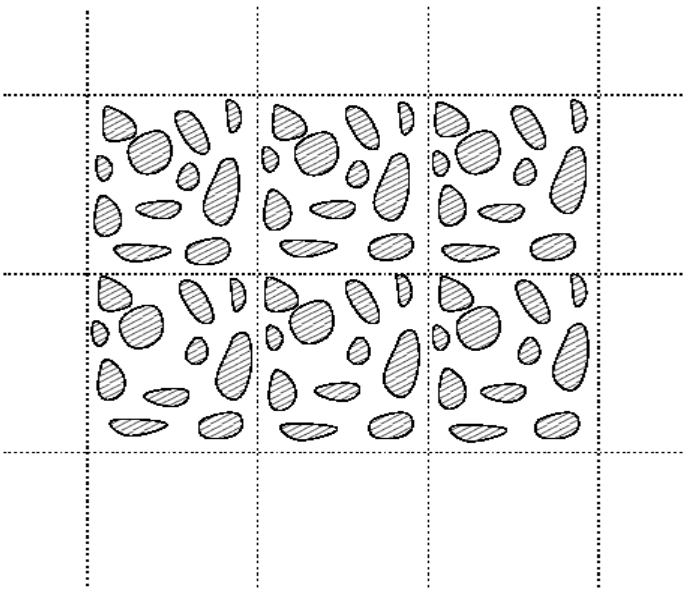

Figura 1.5: Meio poroso periódico.

1.2 Meio Poroso Periódico

Em conseqüência das dificuldades apresentadas na modelagem dos fenômenos de transporte no interior de meios porosos, várias abordagens alternativas são encontradas na literatura. Faremos uso de um meio que se replica pelo espaço tridimensional. Na figura 1.5 podemos ver um corte bidimensional (apenas para facilitar a visualização) de um meio composto por células. Nas referências [4, 5] existe uma abordagem mais completa sob a replicação em meios porosos. Chamaremos esta célula de *célula unidade*, *célula primitiva* ou *célula fundamental*. Podemos descrever a geometria do meio em questão de forma algébrica se definimos $\vec{r}$ como sendo o vetor posição de um ponto dentro da célula. Então, qualquer ponto $\vec{r'}$ dentro do meio poderá ser representado por

$$\vec{r'} = \vec{r} + n_1\vec{l_1} + n_2\vec{l_2} + n_3\vec{l_3}, \tag{1.7}$$

com

$$n_i = 0, \pm 1, \pm 2, \ldots, \quad i = 1, 2, 3 \tag{1.8}$$

onde $\vec{l_1}$, $\vec{l_2}$, e $\vec{l_3}$ são vetores linearmente independentes.

Embora seja possível adotarmos qualquer geometria para o interior da célula elementar (como por exemplo a mostrada na figura 1.5) no decurso deste trabalho adotaremos geometrias relativamente simples e que facilitem, o aspecto computacional. Usaremos células tridimensionais retangulares que contém uma série de blocos de seção retangular, dispostos em diferentes posições como a apresentada na figura 1.6. Apesar da simplicidade, veremos que as mesmas permitem análises relevantes dos problemas estudados.

1.3 Escoamento em Meios Porosos

Coloca-se como referência inicial do estudo do escoamento em meios porosos, o trabalho de Darcy [2], onde está enunciado um fenômeno típico destes meios, a proporcionalidade entre a velocidade média do escoamento e o gradiente de pressão. Esta proporcionalidade é denominada de *Lei de Darcy* e é válida para escoamentos a baixo número de *Reynolds*, ou seja, onde as forças com origem na viscosidade do fluido sejam maiores que as forças originárias das forças de inércia.

Existem tipos de escoamento em meios porosos onde ocorre a dispersão ou arraste de um soluto, e a dinâmica deste processo está em associada à dinâmica do fluido.

As tentativas de descrever este comportamento aplicando-se as equações da hidrodi-

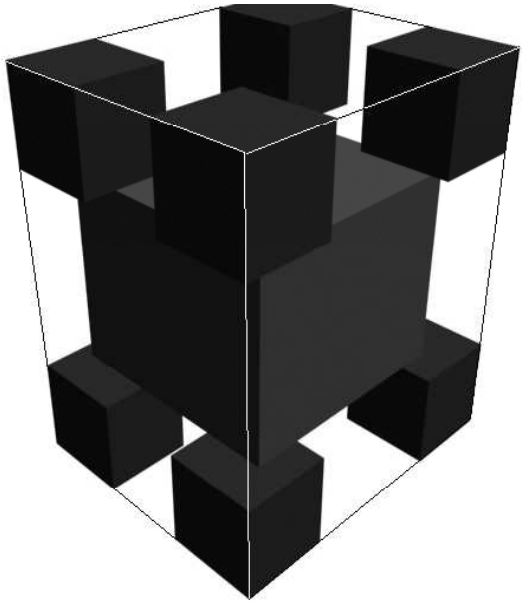

Figura 1.6: Exemplo de célula fundamental tridimensional.

nâmica são complexas devido à necessidade da descrição detalhada da geometria interna do meio, que é geralmente muito complexa, como foi dito na secção anterior. Assim, a determinação das condições de contorno tornam-se inviáveis a não ser para estruturas simples como um conjunto de tubos capilares, esferas ou cilíndros que são idealizações que têm sucesso parcial na descrição de fenômenos nos meios porosos.

No escoamento em meios porosos surgem vários fenômenos dos quais destacaremos:

- **Difusão Molecular.** Aqui o fenômeno ocorre por existir um gradiente de concentração do soluto no meio onde o mesmo se encontra. O que provoca esta dispersão é a dinâmica molecular devido ao movimento *browniano* [6, 7, 8] e é dependente da energia (temperatura) do meio. Assim, este tipo de dispersão sempre ocorre em todas as situações.

- **Mistura Devido a uma Obstrução.** Este tipo de situação ocorre devido aos múltiplos caminhos que as partículas do soluto podem tomar no escoamento do fluido no interior do meio poroso. Dependendo do trajeto, podemos ter parte do soluto sendo retardada em relação à outra porção do mesmo modificando a maneira com que o soluto se dispersa como um todo. Uma representação desta situação está esquematizada na figura 1.7.

- **Recirculações do Escoamento.** Recirculações podem ser provocadas pela estrutura da matriz sólida (figura 1.8), fazendo com que haja um retardo no arrasto do

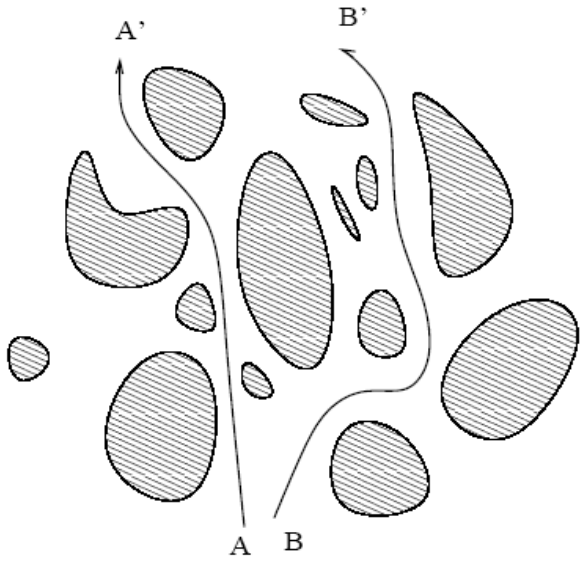

Figura 1.7: Mistura por obstrução.

soluto. No caso de dispersão térmica, a maior área de contato com a matriz sólida num determinado ponto, poderá trazer mudanças adicionais à dispersão.

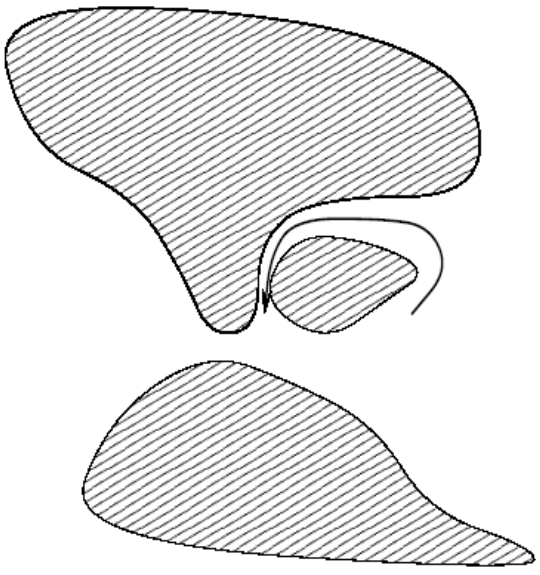

Figura 1.8: Recirculação.

- **Conectividade do Meio.** De uma maneira geral, o meio poroso tem seus poros conectados de forma aleatória o que faz com que os caminhos percorridos no meio poroso não sejam equivalentes. Com isto, podemos ter situações nas quais o soluto siga por caminhos que podem se reencontrar em pontos muito distantes entre si ou mesmo termos caminhos desconexos (figura 1.9).

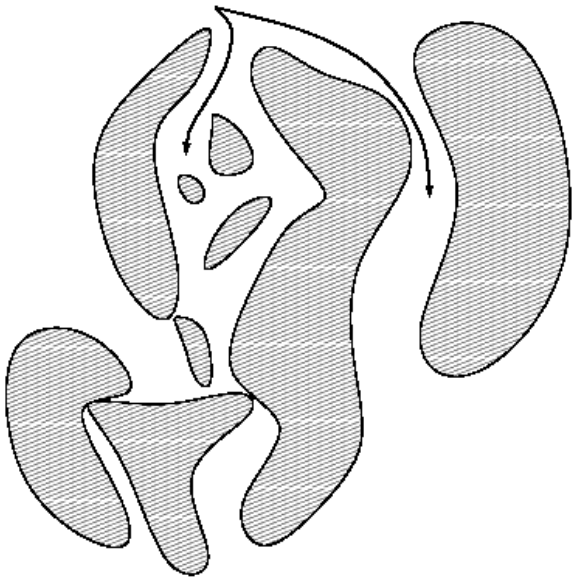

Figura 1.9: Conectividade do meio.

- **Zonas de Estagnação.** Caso o meio poroso tenha zonas de estagnação, teremos o soluto sendo transportando de/para esta região, basicamente de forma *browniana*. Dependendo do escoamento que tenhamos, podemos ter tais regiões praticamente sem o soluto, mudando a distribuição do mesmo (figura 1.10).

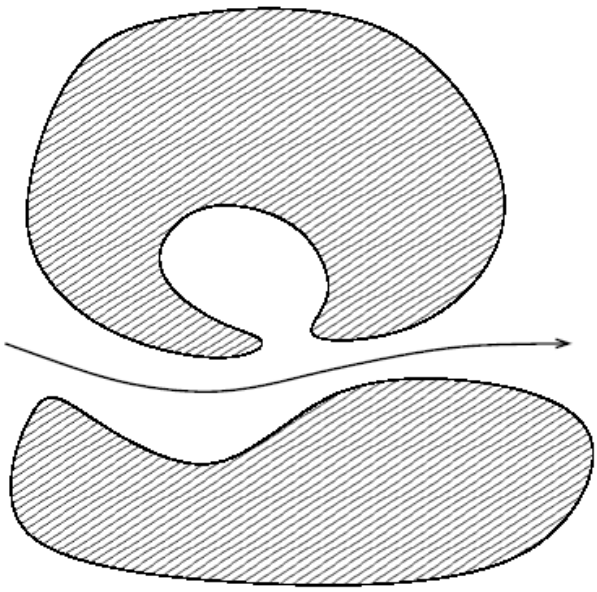

Figura 1.10: Zona de estagnação.

- **Dispersão Hidrodinâmica.** Tal dispersão ocorre pela existência de gradientes de velocidade no escoamento. *Taylor* [9] foi o primeiro a estudar a questão, sendo esta situação conhecida como *Problema de difusão-dispersão de Taylor*. Na figura

1.11 fazemos uma representação na qual mostramos o comprimento de cada seta proporcional a velocidade do fluido nos pontos indicados.

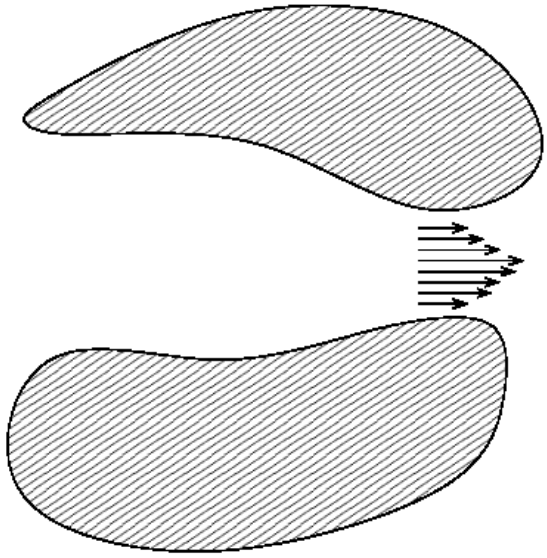

Figura 1.11: Dispersão hidrodinâmica

- **Adsorção.** Se a matriz sólida é capaz de reter (adsorver) o soluto de alguma forma (por exemplo, carvão ativado ou esponja de platina), isto afetará a dispersão que até a fase sólida se encontre saturada. Portanto, haverá um fenômeno transiente que deve ser considerado. Da mesma maneira, podemos ter a liberação do soluto sob determinadas condições e não poderemos negligenciar tal contribuição.

1.4 Dispersão Térmica

Sendo este tema o objetivo central deste trabalho, devemos avaliar como os parâmetros térmicos podem afetar a disperão de calor num meio poroso. Apresentaremos uma análise resumida da encontrada no livro de *Massoud Kaviany* [3].

- **Números de Péclet e Reynolds, Estrutura e Porosidade** : É de se esperar que a dispersão seja dependente da hidrodinâmica na escala dos poros. No entanto, a estrutura da célula fundamental é de importância primordial pois poderiamos ter recirculações e regiões de estagnação. Também é claro que variando a porosidade (mas mantendo fixo Pe e/ou Re) teremos fatores de dispersões diferentes. Algo importante a se destacar é que experimentalmente se verifica que a dependência da dispersão com Re não é muito relevante, o que é algo surpreendente pois dependendo

deste valor, teremos regimes de escoamento diferentes na escala dos poros. Da experimentação temos ainda que a dependência das componentes do tensor dispersão se dá proporcionalmente a uma potência do número de *Péclet*.

Existem razões importantes entre as fases sólido e fluida, então adotando as seguintes simbologias, σ representa a fase sólida, β representa a fase fluida, k a condutividade térmica, α a difusividade térmica e ρc_p a capacidade térmica, escrevemos :

- $\boldsymbol{k_\sigma / k_\beta}$: Esta razão deve influenciar pois o campo de temperatura deve ser influenciado no nível microscópico pela condutividade do sólido. Dependendo das propriedades relativas entre os meios, podemos ter uma maior influência de um determinado meio na dispersão.

- $\boldsymbol{(\rho c_p)_\sigma / (\rho c_p)_\beta}$: Estes valores influenciarão a dispersão principalmente no regime transiente da dispersão de calor. Se, por exemplo, a fase sólida retiver mais calor que a fase fluida, teremos um retardo na dispersão. Portanto, neste caso temos a fase sólida como uma zona de estagnação para o calor.

- $\boldsymbol{\alpha_\sigma / \alpha_\beta}$: Como é de se esperar, devido às análises feitas das grandezas anteriores, esperamos uma dependência da difusividade térmica.

Como visto nessa introdução, o problema de dispersão em meios porosos tem largo interesse. No próximo capítulo serão apresentadas as equações microscópicas e macroscópicas que governam o processo de dispersão em meios porosos.

Capítulo 2

Equação Macroscópica e o Método dos Momentos

Neste capítulo, introduziremos as equações microscópicas que governam o processo do transporte de energia no interior de um meio poroso. Em seguida faremos uma breve introdução sobre o método da média volumétrica que foi utilizado na obtenção da equação macroscópica que governa a dispersão térmica em um meio poroso.

2.1 Equações Microscópicas

O transporte de energia na escala dos poros é descrito pelas seguintes equações de balanço para o fluido (fase β) e o sólido (fase σ)

$$(\rho c_p)_\beta \frac{\partial T_\beta}{\partial t} + (\rho c_p)_\beta \vec{v}_\beta \cdot \nabla T_\beta = \nabla \cdot (k_\beta \nabla T_\beta) \tag{2.1}$$

$$(\rho c_p)_\sigma \frac{\partial T_\sigma}{\partial t} = \nabla \cdot (k_\sigma \nabla T_\sigma) \tag{2.2}$$

com as seguintes condições de contorno na interface sólido-fluido $A_{\beta\sigma}$

$$T_\beta = T_\sigma \tag{2.3}$$

$$\vec{n}_{\beta\sigma} \cdot k_\beta \nabla T_\beta = \vec{n}_{\beta\sigma} \cdot k_\sigma \nabla T_\sigma \tag{2.4}$$

Devemos ainda fornecer as condições iniciais para T_β e T_σ em $t = 0$ e as condições de contorno na entrada e saída, $A_{\beta\sigma}$ e $A_{\sigma\beta}$, do meio.

Como hipótese de trabalho faremos com que os problemas térmico e hidrodinâmico sejam separáveis. Esta separação pode ser considerada se os regimes de trabalho forem tais que as propriedades do meio não sejam modificadas, por exemplo, via a dissipação

viscosa. A descrição completa do processo de transferência de calor necessita também da introdução das equações de continuidade e de momentum para a fase fluida, no intuito de obtermos resultados numéricos para as mesmas.

2.2 Sobre a Média Volumétrica

Descreveremos brevemente o método da média volumétrica que neste trabalho foi utilizado para o cálculo das grandezas de interesse do sistema estudado no *VER* definido no Capítulo anterior.

Neste método pretende-se que as equações macroscópicas fiquem livres das flutuações associadas às pequenas escalas de comprimento.

Apresentaremos a técnica para o caso de termos um meio bifásico constituido de uma matriz sólida (indicada pelo índice σ) e um meio fluido (indicado pelo índice β).

A média volumétrica é definida da seguinte forma [10].

$$\langle \psi_\beta \rangle = \frac{1}{V} \int_{V_\beta} \psi_\beta dV \tag{2.5}$$

onde V é o volume do *VER* e ψ_β é uma grandeza da fase β da qual queremos determinar a média volumétrica.

De forma semelhante introduzimos a média intrínseca à fase β como

$$\langle \psi \rangle^\beta = \frac{1}{V_\beta} \int_{V_\beta} \psi dV = \frac{1}{\varepsilon_\beta} \langle \psi_\beta \rangle \tag{2.6}$$

onde ε_β é a porosidade ou fração volumétrica da fase β, ou seja,

$$\varepsilon_\beta = \frac{V_\beta}{V} \tag{2.7}$$

Com a suposição feita para as relações de comprimento entre as escalas ($l_\beta << r_0 << L$), esperamos que a média volumétrica seja relativamente independente da posição. Assim, para comprimento da ordem de grandeza de R, valeria [11]

$$\left\langle \langle \psi \rangle^\beta \right\rangle^\beta \approx \langle \psi \rangle^\beta \tag{2.8}$$

Observemos que, pela definição, o *VER* pode ser diferente para cada grandeza nas quais estamos interessados, pois cada uma destas grandezas pode exigir dimensões diferentes e, portanto, devemos fazer a média sobre regiões de tamanhos diferentes.

A definição desta média volumétrica e a sua aplicação em meios porosos ordenados (onde a distribuição dos poros é regular) e desordenados (onde a distribuição dos poros é aleatória), foi motivo de estudo em vários trabalhos, dentre eles, os seguintes artigos [12, 13, 14, 15, 16]. O primeiro trabalho a estudar as questões por trás da média volumétrica foi o de *Marle* [17] no qual ele propôs que fosse incluída uma função de ponderação $m(\vec{r})$ de suporte compacto (menor subconjunto fechado e limitado de pontos do domínio onde a função é não nula), na definição da média volumétrica. Denominamos esta de *Média Volumétrica Generalizada*, que é escrita como

$$\langle \psi \rangle = \int_{\Omega} m(\vec{x} - \vec{r}) \psi dV_{\vec{r}} = (m * \psi) \tag{2.9}$$

onde Ω representa o domínio do meio poroso. Assim, interpretamos $\langle \psi \rangle$ como uma distribuição e podemos pensar a média volumétrica como sendo o resultado do produto de duas outras a partir de um operador matemático, ou seja um produto de convolução.

Com esta última interpretação, ficamos com todo o aparato matemático desenvolvido no estudo de convoluções. Observe-se que podemos retornar à notação anterior de média volumétrica se escolhermos como $m(\vec{r})$

$$m(\vec{r}) = \begin{cases} \frac{1}{V} & : \quad \vec{r} \in V \\ 0 & : \quad \vec{r} \notin V \end{cases}$$

Trabalharemos, como anteriormente, com um sistema bifásico constituido de uma matriz sólida e um fluido. Definiremos uma fração volumétrica generalizada como

$$\epsilon_{\beta_m} = (M * \gamma_{\beta}) \tag{2.10}$$

onde γ_{β} é denominada função indicadora da fase β e é definida por

$$\gamma_{\beta} = \begin{cases} 1 & : \quad fase\, \beta \\ 0 & : \quad fase\, \sigma \end{cases}$$

Podemos então definir a média intrínseca generalizada como

$$\langle \psi \rangle_m^{\beta} = \frac{(m * \psi)}{(m * \gamma_{\beta})} \tag{2.11}$$

Para a função peso $m(\vec{r})$ suporemos que ela é de classe C^{∞}, de suporte compacto em R^3 e normalizada, ou seja,

$$\langle 1 \rangle = \int_{\Omega} m(\vec{x} - \vec{r}) dV_{\vec{r}} = 1 \tag{2.12}$$

2.3 Obtenção da Equação Macroscópica

A fim de obter a equação macroscópica, que governa o processo macroscópico de transporte de energia, foi empregado o método da média volumétrica na sua definição clássica, uma vez que só consideraremos meios porosos com periodicidade espacial.

Para obtermos as relações entre as temperaturas médias entre as fases, trabalhamos com a entalpia $\mathcal{H}$. Sob o ponto de vista físico é mais interessante trabalharmos com a entalpia e não com a temperatura devido à natureza intensiva da temperatura, isto é, independe da quantidade de matéria no sistema enquanto que a entalpia é de natureza extensiva, pois depende da quantidade de matéria. A definição de entalpia média é dada por,

$$\langle \rho \rangle \langle \mathcal{H} \rangle = \varepsilon_\beta \rho_\beta \mathcal{H}_\beta + \varepsilon_\sigma \rho_\sigma \mathcal{H}_\sigma \tag{2.13}$$

com $\langle \rho \rangle = \epsilon_\beta \rho_\beta + \epsilon_\sigma \rho_\sigma$, ϵ_σ, ϵ_β respectivamente as frações volumétricas das fases sólida e líquida e ρ_σ e ρ_β as massas específicas das duas fases. Considerando-se a equação calorimétrica para cada fase (por exemplo, $\mathcal{H} = c_{p\beta} T_\beta$) e postulando o mesmo tipo de relação para todo o meio, a temperatura média do meio é definida por,

$$\langle \rho \rangle \langle c_p \rangle \langle T \rangle = \langle \rho c_p \rangle \langle T \rangle = \varepsilon_\beta \left(\rho c_p \right)_\beta \langle T_\beta \rangle^\beta + \varepsilon_\sigma \left(\rho c_p \right)_\sigma \langle T_\sigma \rangle^\sigma \tag{2.14}$$

Aplicado o método da média volumétrica às equações (2.1) e (2.2) e o desenvolvimento algébrico feito em [1], obtem-se para um meio homogêneo e consolidado (ϵ_β = constante) :

$$\langle \rho c_p \rangle \frac{\partial \langle T \rangle}{\partial t} + \varepsilon_\beta (\rho c_p)_\beta \langle \vec{v}_\beta \rangle^\beta \cdot \nabla \langle T \rangle = \nabla \cdot \left(\overline{\overline{K}} \nabla \langle T \rangle \right) \tag{2.15}$$

Nesta equação o tensor efetivo de dispersão térmica é dado por

$$\overline{\overline{K}} = \left(\varepsilon_\beta k_\beta + \varepsilon_\sigma k_\sigma \right) \overline{\overline{I}} + \varepsilon_\beta k_\beta \overline{\overline{\tau}}_\beta + \varepsilon_\sigma k_\sigma \overline{\overline{\tau}}_\sigma - \varepsilon_\beta \left(\rho c_p \right)_\beta \overline{\overline{D}} \tag{2.16}$$

e os tensores de segunda ordem $\overline{\overline{\tau}}_{\beta\sigma}$ e $\overline{\overline{D}}$ por

$$\overline{\overline{\tau}}_{\beta\sigma} = \frac{1}{V_{\beta,\sigma}} \int_{A_{\beta\sigma}} \vec{n}_{\beta\sigma} \vec{f} \, dA \tag{2.17}$$

$$\overline{\overline{D}} = \frac{1}{V_\beta} \int_{V_\beta} \vec{v}_\beta \vec{f} \, dV \tag{2.18}$$

conhecidos na literatura como tensores de *tortuosidade* e de *dispersão hidrodinâmica*, respectivamente e $\vec{f}$ é o vetor que indica a direção do gradiente de temperatura da fase fluida.

2.4 Problema Físico

O problema básico que será discutido corresponde ao processo de dispersão da entalpia ($\mathcal{H} = \langle \rho c_p \rangle \langle T \rangle$), apresentando inicialmente uma determinada quantidade de energia distribuída numa região finita do espaço. A energia será transportada não só por convecção como também por um processo difusivo. Assim, a evolução temporal desta grandeza será descrita por uma equação do tipo difusão-convecção. Conforme visto em [1], o problema do escoamento será tratado independentemente do problema do calor no meio poroso. Sabemos que a equação que governa o tranporte macroscópico de energia é a (2.15), que em termos da entalpia e com um termo de fonte, se escreve como

$$\frac{\partial \mathcal{H}}{\partial t} + \vec{\mathcal{V}} \cdot \nabla \mathcal{H} = \nabla \cdot \left(\overline{\overline{\mathcal{K}}} \nabla \mathcal{H} \right) + \phi \delta(\vec{x}) \delta(t) \tag{2.19}$$

onde $\vec{\mathcal{V}} = r_\beta \langle \vec{v_\beta} \rangle^\beta = \varepsilon_\beta (\rho c_p)_\beta \langle \vec{v_\beta} \rangle^\beta / \langle \rho c_p \rangle$, $\phi \delta(\vec{x}) \delta(t)$ é o termo de fonte e $\overline{\overline{\mathcal{K}}} = \overline{\overline{K}} / \langle \rho c_p \rangle$ com $\overline{\overline{K}}$ sendo o tensor efetivo de dispersão, e $\langle \rho c_p \rangle = \varepsilon_\beta (\rho c_p)_\beta + \varepsilon_\sigma (\rho c_p)_\sigma$.

Antes de prosseguirmos, reescreveremos a equação (2.19) numa forma adimensional. Para tanto, nós vamos introduzir as seguintes variáveis adimensionais

$$t^* = \frac{\alpha_\beta t}{L^2}$$

$$x^* = \frac{x}{L}$$

$$y^* = \frac{y}{L}$$

$$z^* = \frac{z}{L}$$

onde α_β é a difusividade térmica da fase fluida e L o comprimento macroscópico característico. O tempo admensionalizado t^* chamado de número de *Fourier*.

Substituindo-se estas variáveis na equação (2.19), nós obtemos

$$\frac{\partial \mathcal{H}}{\partial t^*} + \vec{\mathcal{V}}^* \cdot \nabla^* \mathcal{H} = \nabla^* \cdot \left(\overline{\overline{\mathcal{K}}}^* \nabla^* \mathcal{H} \right) + \phi \delta(\vec{x}) \delta(t) \tag{2.20}$$

onde nesta equação

$$\mathcal{V}^* = r_\beta Pe = \frac{\varepsilon_\beta (\rho c_p)_\beta}{\langle \rho c_p \rangle} \frac{\langle \vec{v_\beta} \rangle^\beta L}{\alpha_\beta}$$

e

$$\overline{\overline{\mathcal{K}}}^* = \frac{\overline{\overline{\mathcal{K}}}}{\alpha_\beta \langle \rho c_p \rangle}$$

sendo Pe o número de *Péclet* que nos fornece a razão do transporte de energia devido a convecção e ao processo de difusão.

Não nos interessa resolver a equação (2.20), mas acharmos $\overline{\overline{\mathcal{K}}}$ que está ligado ao tensor de dispersão $\overline{\overline{\mathcal{K}}} = \overline{\overline{K}} / \langle \rho c_p \rangle$, para isto usaremos um método que se baseia no cálculo dos momentos de probabilidade.

2.5 Momentos de uma Variável Aleatória

Descreveremos apenas alguns aspectos de estatística que são necessários para o desenvolvimento do método dos momentos. Recomendamos a referência [18] para maiores detalhes.

Dada uma determinada variável aleatória X discreta e que pode tomar os valores $x_1 \cdots x_n$, de acordo com uma densidade de probabilidade $P(x)$, vamos definir a *esperança* matemática desta variável aleatória como

$$E(X) = \sum_{\forall x} xP(X = x) \tag{2.21}$$

Podemos definir a esperança para uma variável aleatória contínua com densidade de probabilidade $P(x)$ da seguinte forma

$$E(X) = \int_{-\infty}^{+\infty} xP(x)dx \tag{2.22}$$

Da mesma forma, podemos definir a esperança para o caso de funções de variáveis aleatórias

$$E[g(X)] = \sum_{\forall x} g(x)P(X = x) \tag{2.23}$$

e

$$E[g(X)] = \int_{-\infty}^{+\infty} g(x)P(x)dx \tag{2.24}$$

chamando a atenção para que geralmente $E[g(x)] \neq g[E(x)]$.

Um caso especial destes resultados é obtido quando a função em questão $g(x)$ é uma potência de X, ou seja,

$$E[X^k] = \sum_{\forall x} x^k P(X = x) = M_k^* \tag{2.25}$$

e no caso contínuo

$$E(X^k) = \int_{-\infty}^{+\infty} x^k P(x)dx = M_k^* \tag{2.26}$$

Este é denominado *momento de ordem k* da distribuição de números aleatórios e é comum chamá-lo de *k-ésimo* momento da distribuição.

Definimos ainda os momentos centrados (ou centrais) como sendo dados por

$$M_k = E[(X - E[X])^k] \tag{2.27}$$

Por exemplo, o momento centrado de ordem 1 é igual a zero, pois

$$E[X - E[X]] = E[X - M_1] = E[X] - M_1 = M_1 - M_1 = 0 \tag{2.28}$$

enquanto o momento centrado de ordem dois é dado por

$$E[(X - E[X])^2] = E[X^2 - 2XE[X] + E[X]E[X]] = \tag{2.29}$$

$$E[X^2] - 2{M_1}^2 + M_1 M_1 = M_2 - {M_1}^2 \tag{2.30}$$

ou ainda,

$$E[X^2] - E[X]E[X] = \sigma^2 \tag{2.31}$$

que é chamado de variância da distribuição.

Os resultados acima podem ser facilmente generalizados para mais de uma dimensão. No caso de 3 dimensões, podemos escrever a esperança de uma função $g(x, y, z)$ como

$$E(g(X, Y, Z)) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g(x, y, z) P(x, y, z) dx dy dz \tag{2.32}$$

Alguns destes momentos são de fácil interpretação. O momento de ordem 0 nos dá o *valor de normalização da densidade de probabilidade associada a* X, o momento de ordem 1 é a *média da variável aleatória* X, ou ainda, o *baricentro da distribuição*, já o momento centrado de ordem 2 é a *variância de* X e contém a informação de como a *distribuição se dispersa em torno da média*, enquanto o momento centrado de ordem 3 reflete *assimetrias que existam da distribuição em torno da média*. No caso unidimensional, o coeficiente de assimetria é definido como

$$\gamma_3 = \frac{M_3}{\sigma^3} \tag{2.33}$$

Temos ainda o momento centrado de ordem 4, a partir do qual pode ser definida uma *taxa de achatamento* (ou *curtose*) da distribuição tomando como referência a distribuição normal. Para o caso unidimensional teremos para a taxa de achatamento o valor

$$\gamma_4 = \frac{M_4}{\sigma^4} \tag{2.34}$$

sendo que os demais momentos são de uso mais incomum.

Na próxima seção, aplicaremos as noções apreendidas aqui ao caso de momentos

definidos num domínio tridimensional, sobre variáveis contínuas e funções com suporte compacto.

2.6 Cálculo dos Momentos

Calcularemos agora os momentos de distribuição de probabilidade da distribuição espacial da entalpia. Assim, a integração será feita em todo espaço físico que contém a quantidade de energia fornecida pelo meio. Temos então os momentos da distribuição de entalpia dados por

$$M_p\left(\mathcal{H}\right) = \int_{\Omega} \mathcal{H}\{\vec{r}^*\}^p d\Omega \tag{2.35}$$

onde $\{\vec{r}^*\}^p$ representa p vezes o produto direto (produto tensorial) do vetor posição $\vec{r}^*$ e a integração é feita sobre todo o domínio tridimensional.

2.6.1 Sobre a Notação

Para facilitarmos a compreensão usaremos a notação indicial mais natural para trabalharmos com tensores de ordem p e operadores diferenciais. Usaremos basicamente a notação apresentada em [19] e não escreveremos as bases com o intuito de aliviarmos o peso da notação. Com isto em mente, notaremos $\{\vec{r}\}^p$ como

$$\{\vec{r}\}^p = r_i r_j r_k \ldots r_m r_l \tag{2.36}$$

ou seja p vezes o produto direto (ou tensorial) de $\vec{r}$. Usaremos, ainda, como representação de diferenciação

$$\frac{\partial G}{\partial r^\alpha} = G_{,\alpha} \tag{2.37}$$

onde G é uma grandeza que pode ser escalar, vetorial ou tensorial e α toma os valores 1, 2 e 3, correspondendo a x, y e z.

2.6.2 Termo Transiente

Apliquemos a definição de momento ao termo transiente da equação macroscópica,

$$M_p\left(\frac{\partial \mathcal{H}}{\partial t^*}\right) = \int_{\Omega} r_i r_j r_k \ldots r_m r_l \frac{\partial \mathcal{H}}{\partial t^*} d\Omega = \frac{dM_p}{dt^*} \tag{2.38}$$

uma vez que a integral comuta com a derivada no tempo pelo fato do domínio não ter dependência temporal.

2.6.3 Termo Convectivo

Multiplicando o termo convectivo da equação (2.20) por $\{\vec{r^*}\}^p$ e integrando-o sobre o domínio Ω teremos

$$M_p\left(\mathcal{V^*}_\alpha \mathcal{H}_{,\alpha}\right) = \int_\Omega r_i r_j r_k \ldots r_m r_l \mathcal{V^*}_\alpha \mathcal{H}_{,\alpha} d\Omega \tag{2.39}$$

Para calcularmos esta integral usaremos a identidade

$$\begin{aligned}\left(r_i r_j r_k \ldots r_m r_l \mathcal{V^*}_\alpha \mathcal{H}\right)_{,\alpha} = \left(r_i r_j r_k \ldots r_m r_l\right)_{,\alpha} \mathcal{V^*}_\alpha \mathcal{H} + \\ r_i r_j r_k \ldots r_m r_l \mathcal{V^*}_{\alpha,\alpha} \mathcal{H} + \\ r_i r_j r_k \ldots r_m r_l \mathcal{V^*}_\alpha \mathcal{H}_{,\alpha}\end{aligned} \tag{2.40}$$

Reescreveremos a integral acima temos :

$$\begin{aligned}M_p\left(\mathcal{V^*}_\alpha \mathcal{H}_{,\alpha}\right) = \int_\Omega \left(r_i r_j r_k \ldots r_m r_l \mathcal{V^*}_\alpha \mathcal{H}\right)_{,\alpha} d\Omega \\ - \int_\Omega \left(r_i r_j r_k \ldots r_m r_l\right)_{,\alpha} \mathcal{V^*}_\alpha \mathcal{H} d\Omega\end{aligned} \tag{2.41}$$

Usando o teorema da divergência no primeiro termo do lado direito desta equação obtemos

$$\begin{aligned}M_p\left(\mathcal{V^*}_\alpha \mathcal{H}_{,\alpha}\right) = \int_\Gamma r_i r_j r_k \ldots r_m r_l \mathcal{V^*}_\alpha \mathcal{H} n_\alpha d\Gamma \\ - \int_\Omega \left(r_i r_j r_k \ldots r_m r_l\right)_{,\alpha} \mathcal{V^*}_\alpha \mathcal{H} d\Omega\end{aligned} \tag{2.42}$$

onde n_α é o vetor unitário normal à superfície. Com este resultado obtemos finalmente

$$M_p\left(\mathcal{V^*}_\alpha \mathcal{H}_{,\alpha}\right) = \int_\Omega \mathcal{V^*}_\alpha \left(r_i r_j r_k \ldots r_m r_l\right)_{,\alpha} \mathcal{H} d\Omega \tag{2.43}$$

2.6.4 Termo Difusivo

Agora integraremos o termo de difusão da equação macroscópica após a sua multiplicação por $\{\vec{r^*}\}^p$

$$M_p\left[\left(\mathcal{K^*}_{\alpha\beta} \mathcal{H}_{,\beta}\right)_{,\alpha}\right] = \int_\Omega r_i r_j r_k \ldots r_m r_l \left(\mathcal{K^*}_{\alpha\beta} \mathcal{H}_{,\beta}\right)_{,\alpha} d\Omega \tag{2.44}$$

Aqui faremos uso de mais uma identidade algébrica

$$\begin{aligned}(r_i r_j r_k \dots r_m r_l \mathcal{K^*}_{\alpha\beta}\mathcal{H}_\beta)_{,\alpha} &= r_i r_j r_k \dots r_m r_l (\mathcal{K^*}_{\alpha\beta}\mathcal{H}_{,\beta})_{,\alpha} \\ &+(r_i r_j r_k \dots r_m r_l)_{,\alpha}\mathcal{K^*}_{\alpha\beta}\mathcal{H}_{,\beta}\end{aligned} \tag{2.45}$$

que após sua substituição na expressão do momento, resultará em

$$\begin{aligned}M_p\left[(\mathcal{K^*}_{\alpha\beta}\mathcal{H}_\beta)_{,\alpha}\right] &= \int_\Omega (r_i r_j r_k \dots r_m r_l \mathcal{K^*}_{\alpha\beta}\mathcal{H}_{,\beta})_{,\alpha} d\Omega \\ &- \int_\Omega (r_i r_j r_k \dots r_m r_l)_{,\alpha}\mathcal{K^*}_{\alpha\beta}\mathcal{H}_{,\beta} d\Omega\end{aligned} \tag{2.46}$$

Aplicando-se o teorema da divergência na primeira integral após o sinal de igualdade obtemos

$$\begin{aligned}M_p\left((\mathcal{K^*}_{\alpha\beta}\mathcal{H}_{,\beta})_{,\alpha}\right) &= \int_{\partial\Gamma} r_i r_j r_k \dots r_m r_l \mathcal{K^*}_{\alpha\beta}\mathcal{H}_{,\beta} n_\alpha d\Gamma \\ &- \int_\Omega (r_i r_j r_k \dots r_m r_l)_{,\alpha}\mathcal{K^*}_{\alpha\beta}\mathcal{H}_{,\beta} d\Omega\end{aligned} \tag{2.47}$$

e como tanto a entalpia como seu gradiente são de suporte compacto teremos que a integral de superfície é nula. Logo

$$M_p\left[(\mathcal{K}_{\alpha\beta}\mathcal{H}_\beta)_{,\alpha,\alpha}\right] = -\int_\Omega (r_i r_j r_k \dots r_m r_l)_{,\alpha}\,\mathcal{K^*}_{\alpha\beta}\mathcal{H}_{,\beta} d\Omega \tag{2.48}$$

Empregando a seguinte identidade algébrica

$$\begin{aligned}[(r_i r_j r_k \dots r_m r_l)_{,\alpha}\mathcal{K^*}_{\alpha\beta}\mathcal{H}]_{,\beta} &= (r_i r_j r_k \dots r_m r_l)_{,\alpha,\beta}\mathcal{K^*}_{\alpha\beta}\mathcal{H} \\ &+(r_i r_j r_k \dots r_m r_l)_{,\alpha}\mathcal{K^*}_{\alpha\beta,\beta}\mathcal{H} \\ &+(r_i r_j r_k \dots r_m r_l)_{,\alpha}\mathcal{K^*}_{\alpha\beta}\mathcal{H}_{,\beta}\end{aligned} \tag{2.49}$$

e considerando o caso no qual o tensor de dispersão seja uniforme, o segundo termo após o sinal de igualdade se anula. Substituindo os termos restantes na integral (2.48) temos

$$\begin{aligned}M_p\left((\mathcal{K^*}_{\alpha\beta}\mathcal{H}_\beta)_{,\alpha}\right) &= -\int_\Omega \left[(r_i r_j \dots r_m r_l)_{,\alpha}\mathcal{K^*}_{\alpha\beta}\mathcal{H}\right]_{,\beta} d\Omega \\ &+ \int_\Omega (r_i r_j \dots r_m r_l)_{,\alpha,\beta}\mathcal{K^*}_{\alpha\beta}\mathcal{H} d\Omega\end{aligned} \tag{2.50}$$

Novamente, usando o teorema da divergência na primeira integral à direita do sinal

de igualdade,

$$M_p\left[\left(\mathcal{K^*}_{\alpha\beta}\mathcal{H}_\beta\right)_{,\alpha}\right] = -\int_\Gamma r_i r_j \ldots r_m r_l \mathcal{K^*}_{\alpha\beta}\mathcal{H}n_\alpha d\Gamma + \int_\Omega (r_i r_j \ldots r_m r_l)_{,\alpha,\beta}\mathcal{K^*}_{\alpha\beta}\mathcal{H}d\Omega \tag{2.51}$$

e a integral de superfície anula-se devido a $\mathcal{H}$ ser de suporte compacto. Assim, determinamos finalmente os momentos da parte difusiva como

$$M_p\left[\left(\mathcal{K^*}_{\alpha\beta}\mathcal{H}_\beta\right)_{,\alpha}\right] = \int_\Omega (r_i r_j \ldots r_m r_l)_{,\alpha,\beta}\mathcal{K^*}_{\alpha\beta}\mathcal{H}d\Omega \tag{2.52}$$

2.7 Os Três Primeiros Momentos

Na determinação dos coeficientes do tensor efetivo de dispersão, conforme será mostrado mais à frente neste capítulo, teremos necessidade de calcular somente os três primeiros momentos da equação macroscópica. Isto será feito a seguir para os momentos de ordem $p = 0, 1, 2$. Neste desenvolvimento, voltaremos a empregar a notação vetorial clássica.

2.7.1 Momento de Ordem Zero

Particularizando o desenvolvimento para o caso do momento de ordem zero, obtemos para o termo transiente

$$M_p\left(\frac{\partial\mathcal{H}}{\partial t^*}\right) = \frac{dM_0}{dt^*} \tag{2.53}$$

sendo que os termos correspondentes aos momentos de ordem zero dos termos convectivo e difusivos são nulos. Portanto, a equação do momento de ordem zero é dada por

$$\frac{dM_0}{dt^*} = \phi\delta(t) \rightarrow M_0 = \int_0^\infty \phi\delta(t)dt^* \rightarrow \phi + c \tag{2.54}$$

se a condição inicial for $M_0(0) = 0$, que M_0 é a *quantidade de energia*, o que é coerente uma vez que este este momento é obtido integrando-se a entalpia sobre todo o domínio.

2.7.2 Momento de Ordem Um

Em se tratando do momento de ordem um do termo dependente do tempo obtemos

$$M_1\left(\frac{\partial\mathcal{H}}{\partial t^*}\right) = \frac{d\vec{M}_1}{dt^*} \tag{2.55}$$

enquanto que do termo convectivo resulta

$$M_1\left(\vec{\mathcal{V}}^* \cdot \nabla^* \mathcal{H}\right) = -\int_\Omega \vec{\mathcal{V}}^* \cdot \nabla^* \vec{r}^* \mathcal{H} d\Omega \tag{2.56}$$

Sendo o gradiente do vetor posição igual ao tensor unitário $\overline{\overline{I}}$, ou seja,

$$\nabla^* \vec{r}^* = \overline{\overline{I}} = \delta_{ij} \tag{2.57}$$

que aplicado ao vetor $\vec{\mathcal{V}}^*$ nos leva a

$$M_1\left(\vec{\mathcal{V}}^* \cdot \nabla^* \mathcal{H}\right) = -\vec{\mathcal{V}}^* \int_\Omega \mathcal{H} d\Omega = -\vec{\mathcal{V}}^* M_0 \tag{2.58}$$

Resta-nos calcular o momento de primeira ordem do termo difusivo, portanto,

$$M_1\left[\nabla^* \cdot \left(\overline{\overline{\mathcal{K}}}^* \nabla^* \mathcal{H}\right)\right] = \int_\Omega \nabla^* \cdot (\nabla^* \vec{r}^*)\, \overline{\overline{\mathcal{K}}}^* \mathcal{H} d\Omega = 0 \tag{2.59}$$

pois sendo o gradiente de $\vec{r}^*$ uma constante, a sua divergência é nula.

Assim, obtemos a equação para o momento de primeira ordem

$$\frac{d\vec{M}_1}{dt^*} - \vec{\mathcal{V}}^* M_0 = 0 \tag{2.60}$$

que, devido ao fato de M_0 ser constante, pode ser reescrita como

$$\frac{1}{M_0}\frac{d\vec{M}_1}{dt^*} = \vec{\mathcal{V}}^* \tag{2.61}$$

Usando a condição inicial $M_1 = 0$ para $t^* = 0$, teremos então a solução

$$\vec{M^*}_1 = \mathcal{V}^* t^* \tag{2.62}$$

onde

$$\vec{M^*}_1 = \frac{\vec{M}_1}{M_0} \tag{2.63}$$

$\vec{M^*}_1$ (primeiro momento da entalpia), nos fornece a posição em cada instante do baricentro da distribuição de energia. A partir da sua determinação numérica podemos obter *a velocidade média do escoamento em regime estacionário.*

2.7.3 Momento de Ordem Dois

Similarmente aos outros casos, para $p = 2$ obtemos para o termo transiente

$$M_2\left(\frac{\partial \mathcal{H}}{\partial t^*}\right) = \frac{d\vec{M}_2}{dt^*} \tag{2.64}$$

e para o termo convectivo

$$M_2\left(\vec{\mathcal{V}}^* \cdot \nabla^* \mathcal{H}\right) = -\int_\Omega \vec{\mathcal{V}}^* \cdot \nabla^* \left(\vec{r}^* \vec{r}^*\right) \mathcal{H} d\Omega \tag{2.65}$$

que após um desenvolvimento algébrico resulta em

$$M_2\left(\vec{\mathcal{V}}^* \cdot \nabla \mathcal{H}\right) = -\int_\Omega \left[(\nabla^* \vec{r}^*) \cdot \vec{\mathcal{V}}^* \vec{r}^* \mathcal{H} + \vec{r}^* (\nabla^* \vec{r}^*) \cdot \vec{\mathcal{V}}^* \mathcal{H}\right] d\Omega \tag{2.66}$$

Como o gradiente de $\vec{r}$ é o tensor identidade temos,

$$M_2\left(\vec{\mathcal{V}}^* \cdot \nabla^* \mathcal{H}\right) = -\vec{\mathcal{V}}^* \int_\Omega \vec{r}^* \mathcal{H} d\Omega - \left(\int_\Omega \vec{r}^* \mathcal{H}\right) \vec{\mathcal{V}}^* d\Omega \tag{2.67}$$

observando que as expressões do lado direito da igualdade constituem-se de dois produtos tensoriais. Pela definição do momento de ordem um, podemos escrever

$$M_2\left(\vec{\mathcal{V}}^* \cdot \nabla^* \mathcal{H}\right) = -\left(\vec{\mathcal{V}}^* \vec{M}_1 + \vec{M}_1 \vec{\mathcal{V}}^*\right) \tag{2.68}$$

Por último, para o termo difusivo

$$M_2\left[\nabla^* \cdot \left(\overline{\overline{\mathcal{K}}} \nabla^* \mathcal{H}\right)\right] = \int_\Omega \left[\nabla^* \left(\nabla^* (\vec{r}^* \vec{r}^*)\right)\right] \overline{\overline{\mathcal{K}}}^* \mathcal{H} d\Omega \tag{2.69}$$

que ao desenvolvermos nos levará a

$$M_2\left[\nabla \cdot \left(\overline{\overline{\mathcal{K}}} \nabla^* \mathcal{H}\right)\right] = \int_\Omega \nabla^* \left(\overline{\overline{I}} \vec{r}^* + \vec{r}^* \overline{\overline{I}}\right) : \overline{\overline{\mathcal{K}}}^* \mathcal{H} d\Omega \tag{2.70}$$

e usando a definição do momento de ordem um, obtemos

$$M_2\left[\nabla^* \cdot \left(\overline{\overline{\mathcal{K}}}^* \nabla^* \mathcal{H}\right)\right] = 2\overline{\overline{\mathcal{K}}}^* \int_\Omega \mathcal{H} d\Omega = 2\overline{\overline{\mathcal{K}}}^* M_0 \tag{2.71}$$

Na sua forma final, a equação do momento de segunda ordem é dada por

$$\frac{d\overline{\overline{M}}_2}{dt^*} - \left(\vec{\mathcal{V}}^* \vec{M}_1 + \vec{M}_1 \vec{\mathcal{V}}^*\right) = 2\overline{\overline{\mathcal{K}}}^* M_0 \tag{2.72}$$

Para compreedermos melhor a situação física, substituiremos na equação (2.72) a

equação (2.63) e, analogamente, escrevemos

$$\overline{\overline{M}}^*_2 = \frac{\overline{\overline{M}}_2}{M_0} \tag{2.73}$$

Usando o resultado de que o momento de ordem zero é constante, reescreveremos a equação para o segundo momento como

$$\frac{d\overline{\overline{M}}^*_2}{dt^*} - \left(\vec{\mathcal{V}}\vec{M^*}_1 + \vec{M^*}_1\vec{\mathcal{V}}^*\right) = 2\overline{\overline{\mathcal{K}}}^* \tag{2.74}$$

Empregaremos agora a equação diferencial do momento de ordem um no intuito de substituirmos $\vec{\mathcal{V}}^*$ e obtermos a equação do momento de ordem dois apenas em função dos momentos já calculados. Desta maneira, escreveremos

$$\frac{d\overline{\overline{M}}^*_2}{dt^*} - \left(\frac{d\vec{M^*}_1}{dt}\vec{M^*}_1 + \vec{M^*}_1\frac{d\vec{M^*}_1}{dt^*}\right) = 2\overline{\overline{\mathcal{K}}}^* \tag{2.75}$$

ou mais concisamente,

$$\frac{d\left(\overline{\overline{M}}^*_2 - \vec{M^*}_1\vec{M^*}_1\right)}{dt^*} = 2\overline{\overline{\mathcal{K}}}^* \tag{2.76}$$

Observemos que o termo sob diferenciação é o momento centrado de ordem 2 que notaremos como $\overline{\overline{\mathcal{M}}}_2$. Assim, chegamos a equação que governa o momento centrado de ordem dois

$$\frac{d\overline{\overline{\mathcal{M}}}_2}{dt^*} = 2\overline{\overline{\mathcal{K}}}^* \tag{2.77}$$

que com a condição inicial $\overline{\overline{\mathcal{M}}}_2 = 0$ para $t^* = 0$ tem como solução

$$\overline{\overline{\mathcal{M}}}_2 = 2\overline{\overline{\mathcal{K}}}^* t^* \tag{2.78}$$

Portanto, podemos relacionar o momento de segunda ordem centrado com o tensor de dispersão da seguinte maneira

$$\overline{\overline{\mathcal{K}}}^* = \frac{\overline{\overline{\mathcal{M}}}_2}{2t^*} \tag{2.79}$$

O momento centrado de segunda ordem representa a *dispersão, em torno do baricentro, da distribuição do calor* e nos mostra como a "nuvem" de energia cresce linearmente com o tempo.

2.8 Conclusões

Apresentamos a equação microscópica e em seguida citamos o método da média volumétrica que foi usado para se obter a equação macroscópica. A partir disso estabelecemos os princípios para determinar o tensor de dispersão partindo da definição dos momentos da distribuição de energia. Neste capítulo discutimos a transmissão de energia num meio poroso percorrido por um fluido em regime estacionário. A equação que governa o transporte macroscópico de energia é uma equação parabólica, similar à equação de energia da mecânica do contínuo. Uma característica deste tipo de equação é de ter uma "velocidade de propagação" infinita, assim as "informações" das condições de contorno são supostas se propagarem de forma instantânea por todo o meio [1]. Então podemos dizer que temos duas escalas de tempo, uma de comunicação e outra de difusão.

Mostramos também que podemos utilizar a equação (2.77) ou (2.79), para obtermos o tensor de dispersão. Os resultados aqui obtidos serão válidos uma vez que as condições de contorno tenham sido transmitidas para todo o domínio de resolução para que sejam válidas médias espaciais e temporais.

Capítulo 3

Metodologia de Trabalho

A fim de que possamos calcular os coeficientes do tensor efetivo de dispersão térmica, trabalharemos com o momento centrado de ordem dois obtido da equação que governa o processo macroscópico de dispersão térmica. Para tanto teremos que efetuar o cálculo do campo de velocidades para o escoamento em questão, e o cálculo dos três primeiros momentos da distribuição de energia. Isto será feito por um método determinístico para o campo de velocidades e um método probabilístico para o cálculo da dispersão térmica. Neste capítulo descreveremos também o desenvolvimento computacional necessário.

3.1 Determinação do Campo das Velocidades

No caso específico deste trabalho, para o problema hidrodinâmico iremos supor que temos um fluido *newtoniano* incompressível cujas propriedades não variam em função das variações de temperaturas supostas no modelo.

As equações que modelam tal situação são,

$$\rho\left[\frac{\partial\vec{v}}{\partial t}+\vec{v}\cdot\nabla\vec{v}\right]=\nabla P+\mu\nabla\cdot(\nabla\vec{v})+\rho\vec{f}\,, \tag{3.1}$$

e

$$\nabla\cdot\vec{v}=0 \tag{3.2}$$

onde ρ é a massa específica, $\vec{v}$ a velocidade do escoamento, P a pressão, μ a viscosidade e $\vec{f}$ uma força externa.

No caso do problema térmico iremos usar como motivação o movimento browniano. O movimento browniano é um movimento aleatório de partículas macroscópicas num fluido, como conseqüência dos choques das moléculas do fluido nas partículas. Vemos uma

ilustração do movimento browniano na figura 3.1. Robert Brown foi um dos primeiros a observar este movimento, a priori achou se tratar de uma nova forma de vida pois as moléculas pareciam descrever movimentos por vontade própria, mas foi em 1905 que Einstein descreveu propriamente esse movimento.

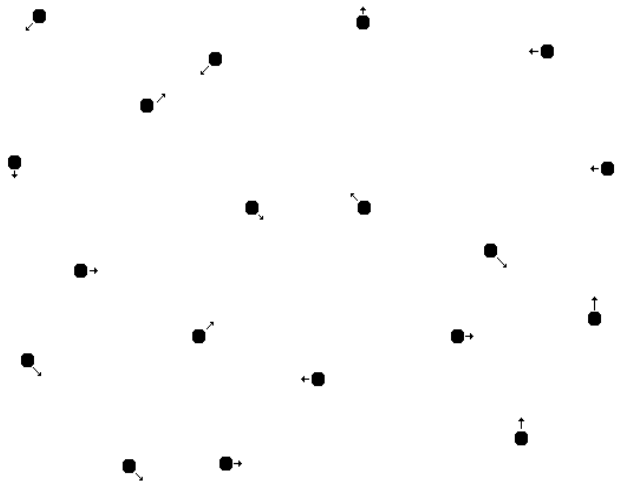

Figura 3.1: Movimento browninano.

Usaremos a teoria de Einstein sobre *Movimento Browniano* numa técnica de solução de problemas envolvendo energia em meios materiais, então examinemos essa teoria que descreve estatisticamente a dispersão de partículas num determinado meio submetido a um banho térmico.

3.2 O Movimento Browniano e a Equação do Calor

Seguiremos com a descrição do *Movimento Browniano* desenvolvida por *Eistein*.

Seja a equação abaixo que informa a quantidade de partículas entre as posições s e $s + ds$

$$n(x, t + \tau)dx = dx \int n(x - s, t)\phi(s)ds \tag{3.3}$$

onde $\phi(s)$ é uma densidade de probabilidade e τ é o tempo no qual ocorre o processo.

Fazendo-se uma expansão em série de Taylor [9] em relação a t pelo lado esquerdo da equação e expandindo-se o lado direito em relação à posição s, iremos obter,

$$n(x,t)+\frac{\partial n}{\partial t}\tau+\frac{\partial^2 n\tau^2}{\partial t^2 2}+... = n(x,t)\int \phi(s)ds-\frac{\partial n}{\partial x}\int s\phi(s)ds+\frac{1\partial^2 n}{2\partial x^2}\int s^2\phi(s)ds+... \tag{3.4}$$

Supondo $\phi(s)$ uma função par, isto é, a probabilidade da partícula se mover para esquerda é a mesma que para direita, e que $\phi(s)$ é normalizada, a expressão acima pode ser reescrita como,

$$n(x,t) + \frac{\partial n}{\partial t}\tau + \frac{\partial^2 n\tau^2}{\partial t^2 2} + ... = n(x,t) + \frac{\overline{s^2}\partial^2 n}{2\partial x^2} + \frac{\overline{s^4}\partial^4 n}{24\partial x^4} \tag{3.5}$$

ou

$$\frac{\partial n}{\partial t}\tau + \frac{\partial^2 n\tau^2}{\partial t^2 2} + ... = \frac{\overline{s^2}\partial^2 n}{2\partial x^2} + \frac{\overline{s^4}\partial^4 n}{24\partial x^4} \tag{3.6}$$

onde $\overline{s^2}$ e $\overline{s^4}$ são respectivamente o segundo e o quarto momento de probabilidade. Supondo os deslocamentos e os intervalos de tempo pequenos o suficiente para desprezarmos os termos de derivada de ordem maior do que 1, no caso do tempo, e maiores que 2 no caso do espaço, a equação acima se reduz a,

$$\frac{\partial n}{\partial t} = \left(\frac{\overline{s^2}}{2\tau}\right)\frac{\partial^2 n}{\partial x^2} \tag{3.7}$$

Com condições de contorno que caem a zero no infinito, a equação (3.7) possui a solução analítica

$$n(x) = \frac{1}{\sigma_x\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma_x^2}} \tag{3.8}$$

com

$$\sigma_x = \frac{\overline{s^2}}{2\tau} \tag{3.9}$$

Observemos que a equação (3.7) é análoga a equação de difusão de calor (3.10) dada por [20],

$$\frac{\partial T}{\partial t} = \alpha\frac{\partial^2 T}{\partial x^2} \tag{3.10}$$

Admitindo $\frac{\overline{s^2}}{2\tau}$ e α iguais nas equações (3.7) e (3.10), obtemos uma conexão entre a difusão de calor num meio material e a equação que informa a quantidade de partículas entre as posições s e $s + ds$.

Estudaremos agora este problema associado ao movimento do meio no qual as partículas se encontram.

No instante de tempo $t = 0$, uma certa quantidade de partículas será introduzida no meio. Para cada passo de tempo δt, as posições das partículas serão atualizadas pela adição de um deslocamento convectivo (determinístico), mais um deslocamento aleatório

devido ao *Movimento Browniano*.

$$\vec{r_i}(t+\delta t) = \vec{r_i}(t) + \delta r_c^i + \delta r_d^i \tag{3.11}$$

onde $\vec{r_i}$ é o vetor posição do térmion i, $\delta r_c^i = \vec{v}_\beta(\vec{r_i})\delta t$ fornecerá o deslocamento convectivo, enquanto que o deslocamento difusivo será dado por $\delta r_d^i = \delta r$

Agora seja o modelo de convecção-difusão tridimensional abaixo

$$\frac{\partial \phi}{\partial t} + v_x\frac{\partial \phi}{\partial x} + v_y\frac{\partial \phi}{\partial y} + v_z\frac{\partial \phi}{\partial z} = k\left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2}\right) \tag{3.12}$$

onde v_x, v_y, v_z são as componentes do vetor velocidade. Discretizando por diferenças finitas em primeira ordem em relação ao tempo e segunda ordem em relação ao espaço obtem-se a equação (3.13):

$$\begin{aligned} \frac{\phi_{(i+1),j,m,l} - \phi_{i,j,m,l}}{\delta t} + v_x\frac{\phi_{i,(j+1),m,l} - \phi_{i,(j-1),m,l}}{2\delta r} + \\ v_y\frac{\phi_{i,j,(m+1),l} - \phi_{i,j,(m-1),l}}{2\delta r} + v_z\frac{\phi_{i,j,m,(l+1)} - \phi_{i,j,m,(l-1)}}{2\delta r} = \\ k\left(\frac{\phi_{i,(j+1),m,l} - 2\phi_{i,j,m,l} + \phi_{i,(j-1),m,l}}{\delta r^2}\right) + \\ k\left(\frac{\phi_{i,j,(m+1),l} - 2\phi_{i,j,m,l} + \phi_{i,j,(m-1),l}}{\delta r^2}\right) + \\ k\left(\frac{\phi_{i,j,m,(l+1)} - 2\phi_{i,j,m,l} + \phi_{i,j,m,(l-1)}}{\delta r^2}\right) \end{aligned} \tag{3.13}$$

onde δt é a variação de tempo e δr a variação de espaço. Reescrevendo a equação acima obtemos :

$$\begin{aligned} \phi_{(i+1),j,m,l} = \frac{\delta t}{\delta r}\left[\left(\frac{k}{\delta r} - \frac{v_x}{2}\right)\phi_{i,(j+1),m,l} + \left(\frac{k}{\delta r} + \frac{v_x}{2}\right)\phi_{i,(j-1),m,l}\right] \\ + \frac{\delta t}{\delta r}\left[\left(\frac{k}{\delta r} - \frac{v_y}{2}\right)\phi_{i,j,(m+1),l} + \left(\frac{k}{\delta r} + \frac{v_y}{2}\right)\phi_{i,j,(m-1),l}\right] \\ + \frac{\delta t}{\delta r}\left[\left(\frac{k}{\delta r} - \frac{v_z}{2}\right)\phi_{i,j,m,(l+1)} + \left(\frac{k}{\delta r} + \frac{v_z}{2}\right)\phi_{i,j,m,(l-1)}\right] \\ + \left(1 - 6k\frac{\delta t}{\delta r^2}\right)\phi_{i,j,m,l} \end{aligned} \tag{3.14}$$

Vejamos quais as condições para que a equação (3.14) represente o movimento de uma partícula que tem seu movimento difusivo determinado pelo *Movimento Browniano* e que é arrastada por um escoamento com velocidade (v_x, v_y, v_z). Como no *Movimento Browniano* o observável é o deslocamento, desconsideraremos que a partícula não se mova. Adotaremos também a interpretação probabilística para a equação acima onde os coe-

ficientes $\phi_{i,j,m,l}$, $\phi_{i,(j-1),m,l}$, $\phi_{i,(j+1),m,l}$, $\phi_{i,j,(m-1),l}$, $\phi_{i,j,(m+1),l}$, $\phi_{i,j,m,(l-1)}$, $\phi_{i,j,m,(l+1)}$ são as probabilidades da partícula ter estado nestes pontos estando no tempo $(i+1)$ na posição (j,m,l). Assim, a probabilidade da partícula ter ficado parada no momento i é nula, ou seja, devemos satisfazer

$$6k\frac{\delta t}{\delta r^2} = 1 \tag{3.15}$$

ou

$$\delta t = \frac{\delta r^2}{6k} \tag{3.16}$$

ou

$$\delta r = \sqrt{6\alpha\delta t} \tag{3.17}$$

e estabelecemos uma conexão entre o deslocamento no espaço tridimensional com o tempo.

Supondo que as componentes v_x, v_y e v_z são positivas, temos que a probabilidade da partícula vir dos pontos $((j-1),m,l)$, $(j,(m-1),l)$, $(j,m,(l-1))$, $((j-1),(m-1),l)$, $((j-1),m,(l-1))$ e $((j-1),(m-1),(l-1))$, tomarão valores positivos. No entanto, para evitarmos valores negativos de probabilidade da partícula vir de $((j+1),m,l)$, $(j,(m+1),l)$, $(j,m,(l+1))$, $((j+1),(m+1),l)$, $((j+1),m,(l+1))$ e $((j+1),(m+1),(l+1))$ (que físicamente corresponde a termos eventos não causais) são necessárias as condições

$$\delta r < \frac{2k}{v_x} \tag{3.18}$$

$$\delta r < \frac{2k}{v_y} \tag{3.19}$$

$$\delta r < \frac{2k}{v_z} \tag{3.20}$$

Caso a velocidade seja negativa, teremos que a probabilidade da partícula de vir de alguma dessas posições $((j-1),m,l)$, $(j,(m-1),l)$, $(j,m,(l-1))$, $((j-1),(m-1),l)$, $((j-1),m,(l-1))$ e $((j-1),(m-1),(l-1))$, também estarão sujeitas às restrições dadas pelas equações (3.18), (3.19, (3.20).

3.2.1 Marcha Aleatória e o Método dos Térmions

Passaremos agora a apresentar a marcha aleatória (*ramdom walk*) [1] onde estudamos as propriedades do movimento de um determinado objeto (idealmente pontual) que percorre uma trajetória aleatória.

O exemplo clássico é dado pelo caminhar de um bêbado (não pontual) junto a um ponto fixo, por exemplo, um poste. Além de descrever este caso de intoxicação etílica,

tal modelo também descreve situações como determinar o momento magnético de meios magnéticos desordenados, intensidade de luz devida a fontes de luz incoerente ou a difusão de moléculas de uma substância em um meio constituído de moléculas de mesmo tipo (autodifusão).

Devido à natureza aleatória, esta marcha só nos dará resultados úteis (como "para que lado realmente o bêbado vai") se :

a) esperarmos um tempo suficientemente longo para acharmos uma "tendência";

b) repetirmos muitas vezes a nossa experiência ou termos muitas experiências simultâneas ocorrendo.

Consideramos que os experimentos são independentes uns dos outros o que é equivalente a supormos que as entidades envolvidas (bêbados, partículas se difundindo, etc) não interagem.

Vamos considerar um caso de marcha aleatória unidimensional. Supondo que a um instante de tempo n, o nosso bêbado dê um passo de comprimento l_n escolhido a cada instante de acordo com uma certa densidade de probabilidade $p(l)$. A sua posição depois de N passos, tendo decorrido um tempo $t = N\tau$, é a soma de seus N deslocamentos, ou seja,

$$X_t = \sum_{n=1}^{N} l_n \tag{3.21}$$

Desde que o primeiro e o segundo momentos $< l >$ e $< l^2 >$ da densidade $p(l)$ sejam finitos, a média e a variância da posição dependem linearmente do tempo, ou seja,

$$\overline{X_t} = Vt \tag{3.22}$$

e

$$\overline{X_t^2} - \overline{X_t}^2 = 2Dt \tag{3.23}$$

onde definimos, respectivamente, a "velocidade" e a "constante de difusão" como

$$V = \frac{< l >}{\tau} \tag{3.24}$$

$$D = (2\tau)^{-1} \left[< l^2 > - < l >^2 \right]. \tag{3.25}$$

A dispersão do tipo acima é denominada *dispersão normal*. No entanto, caso $p(l)$ não tenha algum dos momentos finitos, as definições acima não serão válidas. Devemos ainda

observar que as definições dadas só têm sentido na situação limite na qual o deslocamento espacial e o deslocamento temporal vão a zero conjuntamente.

Uma descrição mais precisa da marcha aleatória pode ser dada pelo uso do Teorema Central do Limite, o qual estabelece que para o primeiro e segundos momentos a distribuição da posição X_t toma, para tempos suficientemente grandes, a forma Gaussiana, ou seja,

$$\lim_{t\to\infty} \text{Probabilidade}\left(u_1 \leq (X_t - Vt)/2\sqrt{Dt} \leq u_2\right) = \frac{1}{\sqrt{\pi}} \int_{u_1}^{u_2} e^{-\xi^2} d\xi \tag{3.26}$$

Da mesma forma que falamos numa marcha unidimensional, poderíamos estender estes resultados para situações tridimensionais.

Neste ponto definiremos Térmion como uma partícula hipotética que transporta uma quantidade arbitrária de energia.

Neste trabalho faremos com que os térmions, partindo de uma determinada posição, sigam trajetórias aleatórias durante um determinado tempo, ao mesmo tempo que são arrastadas pelo fluido, em um meio poroso periódico. Mesmo examinando superficialmente a proposta, ficam evidentes os seguintes aspectos:

i) O número de térmions deverá ser grande o suficiente para que seja estatisticamente relevante;

ii) O número de deslocamentos deverá ser em número suficiente para que os térmions tenham "visitado" uma região fisicamente representativa do domínio considerado;

iii) O tamanho de cada passo deverá ser compatível com as dimensões do meio no qual os térmions se espalham.

Examinemos a questão de nossa definição de marcha aleatória onde definimos deslocamentos diferentes para cada passo. Para simplificarmos o procedimento computacional usaremos passos iguais para o caminhante aleatório. Como trabalharemos com médias, este procedimento não provocará mudanças significativas nos resultados. Tal ponto de vista simplificará o desenvolvimento da parte computacional. Sob este último aspecto, podemos gerar cada térmion e acompanharmos sua evolução no tempo ou podemos gerar todos os térmions e acompanhá-los em cada passo indistintamente. O que determinará qual a abordagem será utilizada serão os resultados que desejamos obter, não nos esquecendo que do ponto de vista computacional tais procedimentos poderão não ser equivalen-

tes. Queremos deixar claro que a proposta aqui sugerida é próxima da técnica conhecida como *Método de Monte-Carlo* [1].

3.2.2 Trajetórias em Meios Homogêneos

Usando novamente a analogia do bêbado, quando ele está, por exemplo, numa rua perfeitamente horizontal, não haverá diferença na probabilidade dele ir para um lado ou para o outro. Assim, qualquer trajetória é equiprovável. Em seguida, nos concentraremos na questão de como os térmions evoluirão neste meio.

Podemos construir duas abordagens para trabalharmos os deslocamentos dos térmions:

a) Definirmos um passo no espaço δr;

b) Definirmos um passo no tempo δt.

Definido um dos dois, poderemos calcular o outro através da relação obtida na pela equação $\delta r = \sqrt{6\alpha\delta t}$.

Quanto às trajetórias, a única restrição que temos é que estas sejam tais que permitam a cada térmion atingir todo o domínio, ou seja, trajetórias isotrópicas.

Em [1] existem duas propostas de geração de direções aleatórias e pelos motivos descritos nela, adotaremos o regime de *Deslocamento Alternado* como pode ser visto na figura 3.4 e que consiste em fazer com que os térmions avancem ora numa direção ora na direção perpendicular à anterior. No caso tridimensional, poderíamos fazer deslocamentos na direções dos eixos x, y e z, sendo as direções e os sentidos escolhidos aleatoriamente. Em [1] foi observado que se o número de deslocamentos dos térmions for suficientemente grande, não teremos mudanças significativas em relação ao deslocamento multidirecional.

3.2.3 Probabilidade de Transição entre Dois Meios

Neste trabalho temos que as propriedades térmicas da fase sólida e fluida podem ser, a priori, diferentes. Aqui analisaremos esta situação levando em conta o desenvolvimento anterior. Partiremos do problema unidimensional, com meios de propriedades térmicas diferentes à esquerda e à direita da origem como apresentado na figura 3.3.

Figura 3.2: Visão bidimensional do deslocamento alternado.

Figura 3.3: Passagem entre meios.

No intuito de obter as probabilidades de transição entre os meios, partiremos de um sistema com as equações de difusão de calor para as fases fluida T_β e sólida T_σ, com uma fonte de calor do tipo delta de Dirac na interface β-σ de dois meios unidimensionais semi-infinitos:

$$\begin{aligned} \frac{\partial T_\beta}{\partial t} &= \alpha_\beta \frac{\partial^2 T_\beta}{\partial x^2} + \delta(x)\delta(t) \\ \frac{\partial T_\sigma}{\partial t} &= \alpha_\sigma \frac{\partial^2 T_\sigma}{\partial x^2} + \delta(x)\delta(t) \end{aligned} \tag{3.27}$$

As condições de contorno são

$$T_\beta = T_\sigma \qquad \text{para } x = 0 \tag{3.28}$$

$$T_\beta = 0 \qquad \text{para } x \to -\infty \tag{3.29}$$

$$T_\sigma = 0 \qquad \text{para } x \to +\infty \tag{3.30}$$

$$k_\beta \frac{\partial T_\beta}{\partial x} = k_\sigma \frac{\partial T_\sigma}{\partial x} + Q\delta_{t=0}\delta_{x=0} \qquad \text{para } x = 0 \tag{3.31}$$

Como resultado da solução deste problema nós obtemos as distribuições de energia

nos dois meios :

$$Q_\beta = \int_{-\infty}^{0} \frac{(\rho c_p)_\beta Q}{\sqrt{(\rho c_p)_\beta k_\beta} + \sqrt{(\rho c_p)_\sigma k_\sigma}} \frac{e^F}{\sqrt{\pi t}} dx$$
$$= \frac{\sqrt{(\rho c_p)_\beta k_\beta}}{\sqrt{(\rho c_p)_\beta k_\beta} + \sqrt{(\rho c_p)_\sigma k_\sigma}} Q \tag{3.32}$$

onde $F = -\frac{x^2}{4\alpha_\beta t}$, e

$$Q_\sigma = \int_{-\infty}^{0} \frac{(\rho c_p)_\sigma Q}{\sqrt{(\rho c_p)_\beta k_\beta} + \sqrt{(\rho c_p)_\sigma k_\sigma}} \frac{e^S}{\sqrt{\pi t}} dx$$
$$= \frac{\sqrt{(\rho c_p)_\sigma k_\sigma}}{\sqrt{(\rho c_p)_\beta k_\beta} + \sqrt{(\rho c_p)_\sigma k_\sigma}} Q \tag{3.33}$$

com $S = -\frac{x^2}{4\alpha_\sigma t}$.

Interpretaremos as constantes de proporcionalidade, que surgem multiplicando Q em Q_β e Q_σ, como probabilidades de transição do térmion de um meio para o outro.

Definindo a efusividade como

$$b_{\beta,\sigma} = \sqrt{(\rho c_p)_{\beta,\sigma} k_{\beta,\sigma}} \tag{3.34}$$

as probabilidades de transição entre as fases serão dadas respectivamente por

$$p_{\sigma\to\beta} = \frac{b_\sigma}{b_\beta + b_\sigma} \tag{3.35}$$

e

$$p_{\beta\to\sigma} = \frac{b_\beta}{b_\beta + b_\sigma} \tag{3.36}$$

Embora este desenvolvimento tenha sido feito para o caso unidimensional, o resultado aqui obtido nos será usado no caso tridimensonal tratado neste trabalho, já que o movimento se dará a cada passo em somente uma direção.

3.2.4 Trajetória de um Térmion ao Cruzar a Interface

Podemos continuar a nossa análise, fazendo uma analogia entre o fenômeno de reflexão ótica e as trajetórias dos térmions ao passar de um meio para outro (mas não devemos considerar esta analogia como sendo rigorosa devido à natureza diversa entre os sistemas físicos, propagação e difusão, nos dê uma certa segurança). Como vimos anteriormente, a relação entre o desvio médio quadrático e a difusividade é o tempo. Se mudamos de meio,

para que ao final do processo tenhamos um resultado compatível com a nossa definição, teremos que de alguma forma compensar, não só a variação no espaço, como também o tempo percorrido na transição entre os meios. Este aspecto pode ser claramente visto ao observarmos a equação,

$$\delta r = \sqrt{6\alpha_i \delta t} \tag{3.37}$$

que relaciona o passo espacial com o passo temporal, onde o índice i indica o meio que estamos considerando. Temos que para diferentes valores de α_i, teremos diferentes intervalos de tempo o que também significa diferentes passos espaciais percorridos nos dois meios. Então, teremos que considerar que ao cruzarmos uma interface deveremos mudar o espaço percorrido pelo térmion. Assim, se o térmion for "refletido", o segmento do passo restante, que penetraria no sólido, será percorrido no meio do qual ele veio. Caso ele atravesse a interface, o segmento percorrido dentro do novo meio será recalculado de forma que o passo de tempo seja conservado.

Como adotamos o caso de trajetórias alternadas (figura 3.4), aliado a estrutura que o meio periódico em estudo tem, teremos uma considerável economia no tempo de cálculo já que as "reflexões" e "refrações" se darão num ângulo igual a zero ou π, simplificando os cálculos. Adicionalmente teremos de levar em consideração apenas a diferença do comprimento do passo em cada meio.

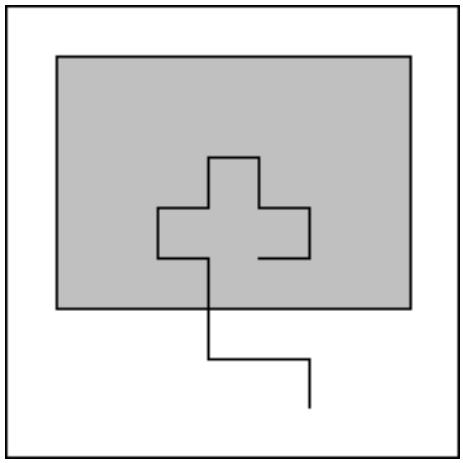

Figura 3.4: Visão bidimensional do deslocamento alternado em meios distintos.

Quando um térmion com probabilidade de cruzar ou não a fronteira de uma interface, o mesmo é "parado" na superfície desta interface e determinamos a sua probabilidade de transição entre os dois meios. Caso um número escolhido aleatóriamente seja menor que a probabilidade de passagem $p_{\sigma\to\beta}$ (ou $p_{\beta\to\sigma}$), então o térmion irá penetrar no sólido (ou fluido). Caso contrário, a partícula sofrerá um choque elástico na fronteira. Além do

mais, como devemos conservar o tempo de deslocamento para uma partícula que tenha gasto $\delta t'_\beta$ (ou $\delta t'_\sigma$) para atingir a interface sólida (fluida), devemos adicionar uma fração de tempo $\delta_{\beta\sigma}$

$$\delta t_{\beta\sigma} = \delta t - \delta t'_\beta = \delta t - \frac{(\delta y_\beta{}')^2}{4\alpha_\beta} \tag{3.38}$$

onde $\delta y'_\beta$ representa o deslocamento sofrido pelo térmion no intervalo de tempo $\delta t'_\beta$. Este incremento de tempo permitirá que a partícula realize um deslocamento espacial caso ela cruze a interface igual a

$$\delta y_\sigma = \sqrt{6\alpha\delta t} \tag{3.39}$$

e caso ela sofra um choque elástico igual a

$$\delta y_\beta = -\sqrt{6\alpha\delta t} \tag{3.40}$$

conforme ilustrado esquematicamente na figura 3.5

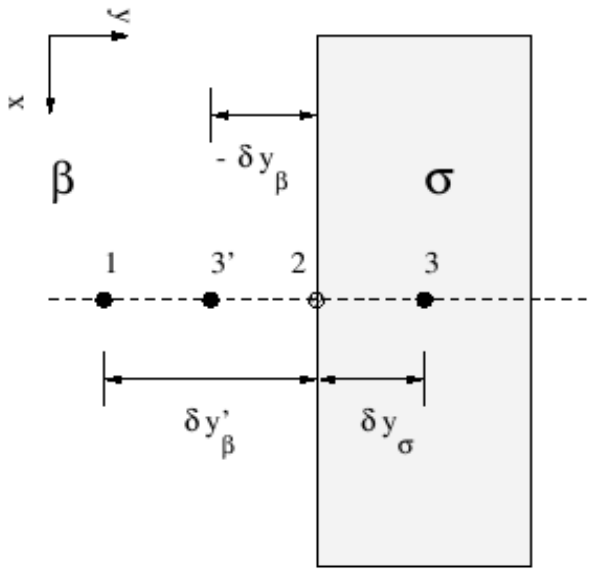

Figura 3.5: Visão bidimensional da trajetória ao cruzar uma interface : (1) Posição inicial, (2) Na interface, (3) Mudança de meio e (3') Choque elástico.

3.2.5 Caminhar Aleatório e Autômatos Celulares

O conceito dos térmions e do movimento aleatório possui uma analogia com autômatos celulares. Um autômato celular é um modelo discreto estudado na teoria da computabilidade e em matemática. Consiste de uma grelha infinita e regular de células, cada uma podendo estar em um número finito de estados, que variam de acordo com regras determinísticas, por exemplo na figura 3.6 pode ser visto o autômato celular cuja a tabela 3.1

tem sua regra de formação, essa regra diz que se três células adjacentes tem atualmente o padrão 100 (célula da esquerda 1, com as outras 0) ou 001 (célula da direita 1, com as outras 0) então a célula do meio se tornará 1 na próxima iteração, essa regra é conhecida como regra 30. O tempo também é discreto, e o estado de uma célula no tempo (t) é uma função do estado no tempo $(t-1)$ de um número finito de células na sua vizinhança. Essa vizinhança corresponde a uma determinada seleção de células próximas (podendo eventualmente incluir a própria célula). Todas as células evoluem segundo a mesma regra para atualização, baseada nos valores das suas células vizinhas. Cada vez que as regras são aplicadas à grelha completa, uma nova geração é produzida. Os autômatos celulares foram introduzidos por *von Neumann* e *Ulam* como modelos para estudar processos de crescimento e auto-reprodução. Qualquer sistema com muitos elementos idênticos que interagem local e deterministicamente podem ser modelados usando autômatos celulares [21].

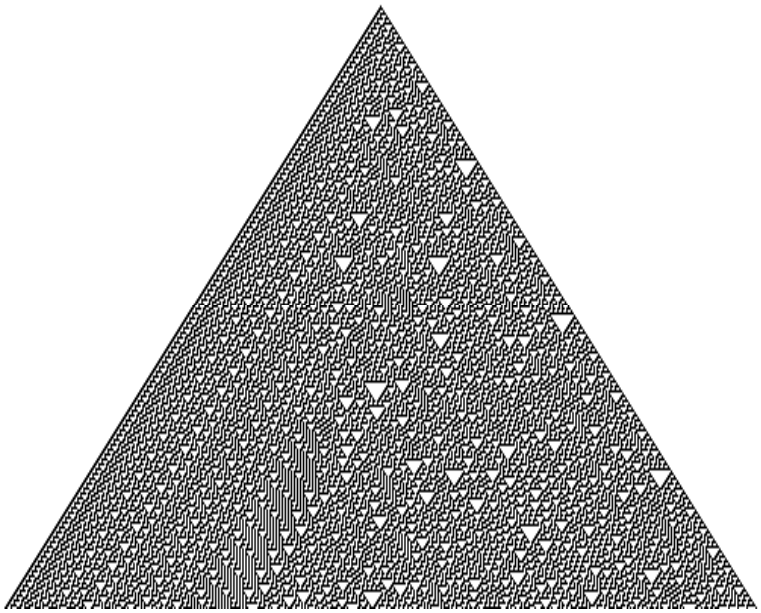

Figura 3.6: Autômato celular.

Tabela 3.1: Regra 30

Padrão atual	111	110	101	100	011	010	001	000
Novo estado para célula central	0	0	0	1	1	1	1	0

3.3 O Modelo e sua Implementação

Na implementação do método dos térmions, devemos ter de forma clara os conceitos físicos que estão por trás do cálculo dos momentos. Ao fazermos uma simulação deveremos

usar um determinado número de térmions e fazê-los evoluir por um determinado tempo de tal forma que tenhamos uma descrição estatisticamente significativa dos fenômenos que investigaremos. Podemos dizer, de outra maneira, que os térmions deslocam-se pelo meio no qual se dispersam durante tempo suficientemente longo de modo que possamos estar livres das flutuações estatísticas. Usando o jargão usado para métodos de Monte Carlo e afins, diremos que os térmions devem ter tempo suficiente para "visitar" uma região considerável do meio no qual caminham. Em termos exatos, apenas quando o tempo de simulação tender a infinito é que teremos os momentos e conseqüentemente o tensor de dispersão. Além disto, o número de partículas deveria ser infinito. Como isto não é possível, trabalharemos com uma quantidade grande o suficiente para termos médias de comportamento representativas.

Quanto ao tempo, a simulação realizada para um tempo finito apresentará um comportamento inicial não linear, que não será analisado na presente formulação. Tal comportamento ocorrerá até que tenha decorrido um tempo suficiente para que os térmions espalhem-se pelo domínio computacional, colhendo informações necessárias para atingir o regime estacionário. Ocorrerão oscilações nos valores obtidos para os momentos devido ao fato de empregarmos um número finito de térmions na simulação. Assim, os resultados determinados para os coeficientes do tensor de dispersão só serão significativos quando o regime assintótico for atingido, ou seja, uma vez transcorrido um intervalo de tempo suficientemente longo para que tenhamos o regime permanente. Contudo, estes valores dos coeficientes ainda conterão um erro devido ao emprego de um número finito de térmions.

No que diz respeito ao tempo necessário para atingirmos o estado estacionário, podemos conseguir alguma informação a partir do número de Fourier, ou tempo adimensional.

$$t^* = \frac{\alpha t}{L^2} \tag{3.41}$$

Obtemos uma interpretação para o seu significado se reescrevermos a equação acima como,

$$t^* = \frac{k/(1/L)L^2}{(\rho c_p L^3)/t} \tag{3.42}$$

assim, podemos dizer que o número de Fourier fornece a relação entre a taxa de condução de calor através de um volume e a taxa de armazenamento de calor neste mesmo volume. Observe que quando estas taxas forem iguais teremos $t^* = 1$. Como veremos a seguir, este será o tempo usado na determinação das propriedades efetivas.

Iniciaremos a implementação com a "geração" de um número de térmions, que no

instante inicial ($t = 0$) terão a sua posição conhecida. A maneira como os térmions são introduzidos no meio representará diferentes tipos de distribuição como Dirac, degrau, uniformemente distribuídos etc. Para cada partícula introduzida no meio seguiremos a sua trajetória e o seu deslocamento espacial durante um intervalo de tempo previamente fixado. A partir destes deslocamentos, podemos calcular os momentos de primeira e segunda ordem da distribuição de térmions. Como já visto, a velocidade média de translação do baricentro desta "nuvem" de térmions e os coeficientes do tensor efetivo de dispersão serão calculados por intermédio destes momentos. Como estamos interessados no comportamento assintótico, podemos considerar que os instantes iniciais não deverão contribuir de modo relevante na determinação das propriedades estatísticas para valores do número de Fourier compatíveis com a hipótese de "estacionaridade".

3.3.1 Estrutura Básica do Programa

O programa parte de uma distribuição inicial de térmions que evoluem mediante deslocamentos devidos ao movimento browniano e a existência do escoamento da fase fluida. A distribuição inicial poderá ser de três tipos: pontual, faixa e uniforme. No caso pontual a posição inicial dos térmions é escolhida aleatoriamente dentro de um quadrado de lado δr, centrado no ponto especificado como sendo a fonte de térmions. No caso da opção faixa, os térmions são distribuídos aleatoriamente dentro de uma faixa de altura e largura definidas em torno do ponto de fonte. No caso uniforme, os térmions são distribuídos aleatoriamente e uniformemente dentro da célula unitária. A direção do movimento do térmion é determinada alternadamente como sendo as direções dos eixos Ox, Oy e Oz. Caso haja possibilidade de mudança de meio, um número aleatório é escolhido e comparado com a probabilidade de transição, a fim de determinarmos se a mudança ocorrerá ou não. Concluído o deslocamento difusivo, efetuamos o movimento devido à existência do escoamento da fase fluida, ou seja, adicionamos o deslocamento convectivo. Neste ponto, a posição de cada térmion é armazenada de modo a calcularmos os momentos parciais de ordem 1 e 2.

Ao final da simulação são calculados os momentos centrados totais e, por regressão linear, as grandezas físicas associadas aos mesmos: a velocidade média do centróide da distribuição de térmions e os coeficientes do tensor efetivo de dispersão.

Na figura 3.7 é apresentado o diagrama de fluxo simplificado do código numérico desenvolvido para a implementação do método dos térmions.

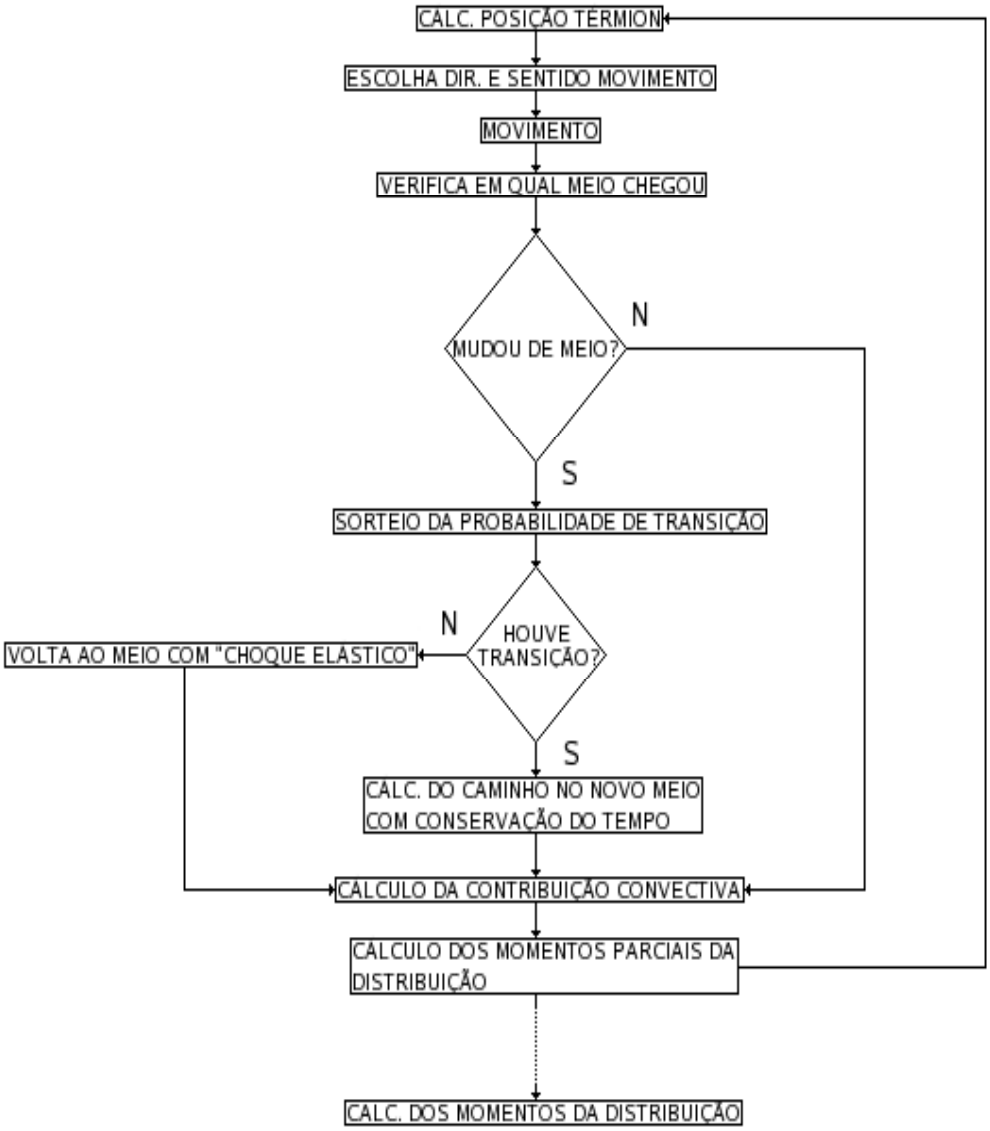

Figura 3.7: Diagrama de fluxo

3.3.2 Implementação do Método

Temos um processo do tipo de Markov, isto é, para qualquer seqüência de eventos no domínio do tempo, a probabilidade condicional de um evento atual dados todos os eventos passados e presentes só depende do evento imediatamente anterior. Cada deslocamento leva em consideração apenas a posição imediatamente anterior do térmion, o que acarretará num "esquecimento" das condições iniciais à medida que o sistema evolui.

A interação dos térmions com o campo de velocidades, conforme já mencionado, dar-se-á pelo "arrasto" destas partículas pelo fluido e será adicionado ao deslocamento devido ao movimento browniano. Devemos observar que o deslocamento browniano se

Tabela 3.2: Número de térmions em função da precisão η.

γ	1%		5%		10%	
η	(3.43)	χ^2	(3.43)	χ^2	(3.43)	χ^2
1 %	133128	132700	76832	76829	54120	54111
3 %	14793	14746	8538	8537	6014	6012
5 %	5325	5310	3073	3073	2165	2164

dá de maneira bem determinada (e relacionado com o passo temporal), cada passo de espaço sendo determinado *a priori*, variando apenas quando há troca de meio, enquanto que o campo de velocidades irá variar pontualmente. Devemos tomar cuidado para que os deslocamentos convectivos não excedam as dimensões dos elementos sólidos contidos na célula elementar, pois caso contrário, o térmion poderia "atravessar" os sólidos sem iteragir com os mesmos. Uma vez que o passo espacial (difusão), o passo temporal e o deslocamento total (via campo de velocidades) estão relacionados, tais cuidados são fundamentais para que não hajam interpretações físicas errôneas dos resultados obtidos.

3.4 Número de Térmions

O número total de térmions e a precisão que almejamos obter podem ser avaliadas no caso de difusão pura, usando estatística.

Conforme o desenvolvimento feito na referência [1], utilizaremos a relação:

$$n \approx n - 1 > \frac{2z_c^2}{\eta} \tag{3.43}$$

com

$$\eta = \frac{nS^2}{(n-1)\sigma^2} - 1 \tag{3.44}$$

onde n é a quantidade de térmions, z_c é chamado de valor crítico, η a precisão para variância, S^2 a variável aleatória e σ^2 a variância. Baseado nas equações (3.43), (3.44) e na lei do χ^2 obtivemos a tabela 3.2 que fornece a quantidade de térmions que devemos usar em função da precisão do cálculo da variância igual a η para um intervalo de confiança dado por γ.

Utilizando a equação de calorimetria abaixo,

$$k_{\parallel} = \varepsilon_\beta k_\beta + \varepsilon_\sigma k_\sigma \tag{3.45}$$

com os valores de porosidade $\varepsilon_\beta = 0.64$ e $\varepsilon_\sigma = 0.36$ e parâmetros térmicos $k_\beta = 1$, $k_\sigma = 2$, $(\rho c_p)_\beta = 2$ e $(\rho c_p)_\sigma = 1$ para o meio extratificado (figura 3.8), gerou-se a tabela 3.3.

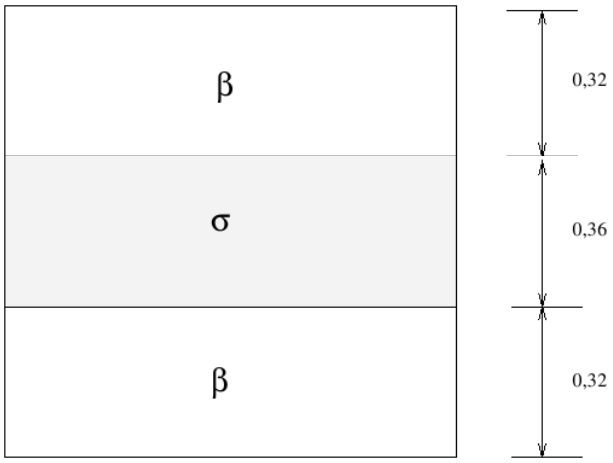

Figura 3.8: Meio extratificado e porosidade.

Tabela 3.3: $k_{\|}$ em função da quantidade de térmions.

Térmions	$k_{\|}$
100	1,1415
250	1,3232
500	1,3337
750	1,3323
1000	1,4175
2500	1,3803
5000	1,3480
7500	1,3557
10000	1,3583
25000	1,3570
50000	1,3569

Verificamos que de 7500 a 50000 térmions os valores $k_{\|}$ são práticamente equivalentes, agora recorrendo a tabela 3.2, verificamos que 7500 térmions o intervalo de confiança está entre 5% e 10%, entretanto escolhendo 10000 térmions verificamos que o intervalo de confiança estará entre 1% e 5%, com a precisão do cálculo da variância de 3%, baseado nisso usaremos a quantidade de 10000 térmions nas simulações.

Com relação ao passo dado por cada térmion adotaremos a expressão abaixo obtida da referência [22] para obter o valor máximo de incremento espacial,

$$\delta r_d^* = \frac{2}{v_{max}^*}\left[\sqrt{1 + cv_{max}^*\Delta} - 1\right] \tag{3.46}$$

onde v_{max}^* é o maior valor da velocidade no escoamento do fluido, c uma constante menor ou igual a 1 e Δ o tamanho característico do elemento da malha.

A seguir iremos analisar uma parte importante da implementação referente à localização do térmion, ou seja, quando ele se encontra no fluido ou na matriz sólida.

3.5 Análise dos Algoritimos de Localização

Foi verificado que grande parte do processamento é dedicado à localização de cada término dentro do meio poroso, verificando se ele encontra-se em um sólido ou fluido. O algoritimo original como foi desenvolvido possui um custo de localização de um dado término em qualquer posição de $O(n)$, ou seja, linear com o número de blocos e compromete em média 50% de processamento em situações típicas do ambiente bidimensional. É necessário, portanto, usar uma técnica que seja mais eficiente e abordaremos algumas possibilidades.

3.5.1 Busca Binária

A primeira estratégia intuitiva de diminuição de custo de busca é o algoritimo de busca binária que apresenta um custo menor que o seqüencial $O(log(n))$ [23], [24]. Entretanto é exigido um vetor de dados ordenado, no nosso caso utilizamos uma estrutura de alocação dinâmica (ponteiros) onde armazenamos uma matriz cúbica, ou seja, cada elemento da lista é uma tripla ordenada. Então supondo, a utilização de um vetor e desprezando o custo inicial de ordenação do mesmo temos um custo de busca de $O(3 * log(n))$.

3.5.2 Árvores e Octrees

O custo de pesquisa em árvore é similar ao da busca binária [23], [24], isto é, $O(log(n))$, e o tipo mais indicado seria a *árvore binária de busca* (figura 3.9). Neste caso não é necessário retirar e nem inserir elementos da árvore, evitando processos de rebalancemento como nas árvores AVL, e a mesma é própria para se trabalhar com alocação dinâmica, porém não é desprezado o fato que os elementos da árvore devam estar ordenados.

Outra estrutura em árvore é o *octree*, cuja a característica é a de que cada nodo ou raíz possui até oito folhas, como ilustrado na figura 3.10 ou em representação tridimensional em 3.11. Essas estruturas de dados são muito usadas para particionamento de espaço tridimensional via uma subdivisão recursiva em oito octantes.

3.5.3 Tabela HASH

A tabela HASH é a estrutura computacional que pode efetuar buscas com custo típico de $O(1)$, ou seja, constante para qualquer quantidade de dados. Entretanto ela possui algumas restrições :

Figura 3.9: Diagrama de árvore binária (f - folha, r - raíz).

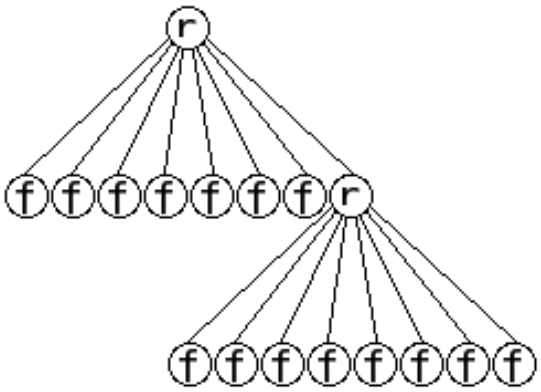

Figura 3.10: Diagrama de octree (f - folha, r - raíz).

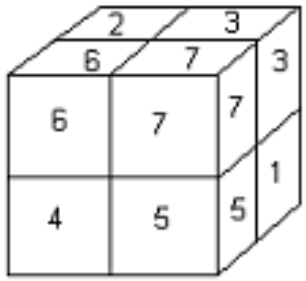

Figura 3.11: Diagrama tridimensional de octree.

- Os dados armazenados podem vir a ocupar um espaço grande ou fragmentado dentro da memória;
- A tabela *HASH* tem seu melhor desempenho quando a função conhecida como função geradora ou função de *HASH*, que é usada para montar ou buscar, não gera colisões. Uma colisão é quando dados pelo menos dois elementos x e y obtemos $f(x) = f(y)$ onde f é a função de *HASH* com $f(x)$ e $f(y)$ sendo os valores naturais que representam posições na memória de um computador. Nas colisões o *HASH* tem como alternativa montar uma subtabela que pode apresentar um desempenho $O(n)$.

Ambos os problemas acima estão diretamente ligados ao fato de uma má escolha de uma função de *HASH* vinculada ao problema ou a não possibilidade de criação de uma

função adequada.

Matematicamente para que a tabela *HASH* funcione adequadamente sem colisões a função deve ser injetora (para cada x temos uma única $f(x)$). Ser sobrejetora é interessante apenas quando é limitada a quantidade de memória obrigando a função *HASH* a aproveitar ao máximo cada espaço.

No nosso caso temos que os dados são expressos na forma (x, y, z), que representam a posição do térmion no espaço. Como vimos nos capítulos anteriores os térmions representam uma quantidade arbitrária de energia e não interagem entre si, portanto não há sentido físico em descrevermos dois termions com a mesma posição (x, y, z) no espaço.

Portanto propomos a seguinte expressão para a função de HASH:

$$P = P_x + (P_y * Q_x) + (P_z * Q_X * Q_y) \tag{3.47}$$

onde P é a posição na memória do computador, P_x, P_y e P_z representam respectivamente o valor inteiro obtido da conversão das coordenadas reais para inteiras e Q_x, Q_y e Q_z representam a quantidade de pontos em cada direção do espaço.

A função proposta obedece a injetividade que mencionamos acima. Portanto, teremos em custo computacional de $O(1)$ na pesquisa. Assim, usaremos a tabela *HASH* na presente implementação.

Esta estratégia garante que cada elemento ocupará um único espaço na memória e que para cada elemento que ocupe a posição k da memória o próximo ocupará a posição $(k+1)$ (para quantidade de térmions maior ou igual a 2). Isso evitará a desfragmentação de memória, se o sistema operacional alocar um bloco contínuo de memória. Todos os sistemas operacionais modernos trabalham com a estratégia de repaginação de memória [25], desta forma mesmo havendo a chamada fragmentação interna (a nível lógico baixo), não será perceptível pois a alocação de memória irá gerar uma repaginação de memória gerando um bloco único e com posições seqüenciadas. A seguir apresentaremos algumas análises para corroborar esta escolha de maneira mais objetiva.

3.6 Comparando as Funções de Busca

Iremos fazer uma comparação da função de busca por um térmion utilizado no algoritimo original [1] com a função baseada no método *HASH*.

Vamos denominar cada operação de busca como uma Unidade de Esforço Computacio-

nal (UEC), e chamaremos de *nblocos* a quantidade de blocos sólidos da célula fundamental, ou seja, a quantidade de partes disjuntas de sólido da célula fundamental.

Na estratégia de busca seqüencial temos os seguintes esforços maximizados :

Tabela 3.4: Ações e Custos de busca.

Ação	Custo sequencial	Custo *HASH*
Atribuições >=	$1 + (nblocos * 3)$	$5 + 3$
Loops =	1	0
Comparações >=	$(nblocos * 7)$	7
Cálculos Diretos =	0	$9 + 2$
Conversões numéricas =	0	3
Total =	$(nblocos * 3) + (nblocos * 7) + 2$	29

Fazendo *nblocos* igual a 1 temos :

Total $= 1 + 3 + 1 + 7 + 0 = 12$ UEC.

Na tabela 3.4 vemos o custo maximizado da tabela *HASH* em 29 UEC, e não é dependente de *nblocos*.

Vemos que a estratégia de busca *HASH* apresenta um esforço computacional maior para células fundamentais com até dois blocos sólidos. Contudo, como o custo do *HASH* é fixo para os casos de $N > 3$, o algoritimo de busca será mais eficiente usando HASH. Para efeito de exemplo a versão *HASH* da função de busca para uma estrutura com $nblocos = 5$ tem em torno de 44% a mais de eficiência teórica na busca por um elemento quando comparada ao seqüencial.

Devemos alertar ao leitor que esse desempenho se aplica somente no algoritimo de busca do térmion. Mesmo assim, devido à importância da localização do térmion o resultado global será significativo, como veremos posteriormente.

A implementação do algoritimo está disponível no apêndice B deste trabalho.

3.7 Conclusão

Neste capítulo descrevemos como problema termodinâmico é tratado com o uso da teoria de Eistein sobre o movimento browniano que lida com partículas. Usamos a idéia de térmion com marcha aleatória como conexão entre o movimento browniano e a idéia clássica sobre a difusão de calor, que do ponto de vista computacional essa situação pode se encarada como um autômato celular.

Mostramos como é feita a implementação do código, juntamente com o tratamento dos térmions cujos o deslocamentos são influenciados pela difusão e convecção existentes no fluido do meio composto por células fundamentais. Fizemos o tratamento de passagem fluido-sólido e sólido-fluido do térmion. Obtivemos também a quantidade de térmions necessária para que haja uma certa precisão nos valores obtidos via cálculo do tensor de dispersão.

A necessidade de aliviar o processo de busca por térmions nos levou a procurar uma melhor estratégia, e concluímos que a tabela HASH é a que tem o melhor desempenho.

Capítulo 4

Avaliando a Implementação

Faremos agora uma análise dos aspectos computacionais da implementação elaborada a partir das discussões anteriores. Inicialmente utilizamos a célula fundamental no formato de placas paralelas, em seguida fizemos simulações com célula fundamental na forma de um tubo seção de quadrada, e variações desta célula. No caso mais simples das simulações o meio poroso como um todo foi representado por um feixe de tubos como pode ser visto em recorte na figura 4.4. Todas as simulações foram feitas em máquina equipada com processador Pentium IV Hyper Threading de 2.2Ghz de clock, sob o sistema operacional Linux Fedora Core 5, e com 1Gb de memória RAM, utilizando o compilador GCC versão 4.0.2 20051125 (Red Hat 4.0.2-8).

4.1 Testes Preliminares

Usamos como célula fundamental a situação na qual temos duas placas sólidas e um fluido escoando entre elas, como mostrado em recorte bidimensional na figura 4.1. As componentes do campo vetorial v_y e v_z são nulas, e a componente v_x é calculada pela expressão (4.1) [1].

$$v_x = \frac{3}{2}\left(1 - \frac{y^2}{h^2}\right) \tag{4.1}$$

A componente longitudinal do tensor de dispersão térmica é dada pela equação,

$$K_{\parallel} = k_{\parallel} + \varepsilon_\beta^3 Pe^2 k_\beta \left[\frac{17}{140} - \frac{r_\beta}{5} + \frac{r_\beta^2}{12}\frac{r_\beta(1-r_\beta)}{12}\frac{\alpha_\beta}{\alpha_\sigma}\frac{\varepsilon_\sigma}{\varepsilon_\beta}\right] \tag{4.2}$$

obtida da referência [26], onde $r_\beta = \varepsilon_\beta(\rho c_p)_\beta/[\varepsilon_\beta(\rho c_p)_\beta + \varepsilon_\sigma(\rho c_p)_\sigma]$ e $k_{\parallel} = (\varepsilon_\beta k_\beta + \varepsilon_\sigma k_\sigma)$. Utilizado os seguintes valores de porosidade e parâmetros térmicos, $\varepsilon_\beta = 0.6$, $\varepsilon_\sigma = 0.4$, $k_\beta = 1$, $k_\sigma = 1$, $\alpha_\beta = 1$, $\alpha_\sigma = 1$, $(\rho c_p)_\beta = 1$ e $(\rho c_p)_\sigma = 1$, obteve-se a segunda coluna da

Figura 4.1: Célula fundamental.

tabela 4.1 que representa a solução analítica de $K_{\parallel}$ em função de Pe.

Utilizando campo de velocidades gerado a partir da expressão (4.1), com os mesmos parâmetros térmicos e de porosidade utilizados na obtenção da solução analítica de $K_{\parallel}$, realizamos as simulações e obtivemos a terceira coluna da tabela 4.1. O gráfico de $K_{\parallel}$ em função de Pe comparando a solução analítica e a obtida via implementação pode ser visto na figura 4.2.

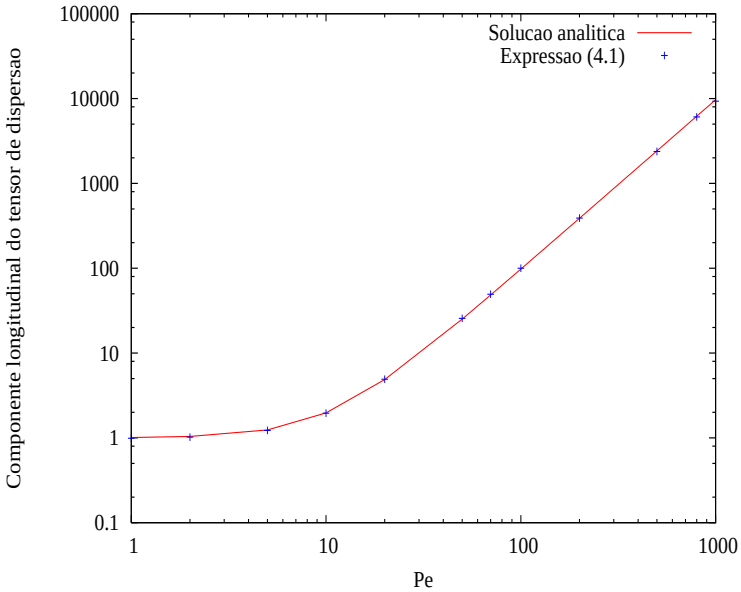

Figura 4.2: Cruzamento entre $K_{\parallel}$ analítico e $K_{\parallel}$ pela expressão (4.1) em função de Pe.

Os valores do $K_{\parallel}$ apresentados possuem variações entre 1% e 5%, como era esperado

pela análise feita no capítulo anterior.

Para o cálculo do campo vetorial deste mesmo problema usamos o software FlexPDE que pode ser encontrado em *www.pdesolutions.com*. O mesmo será usado para o cálculo do campo nas demais situações apresentadas. Na figura 4.3 é apresentado o campo vetorial calculado na interface típica do software. Apesar da trivialidade do teste procuramos averiguar como está funcionando a leitura de dados da implementação.

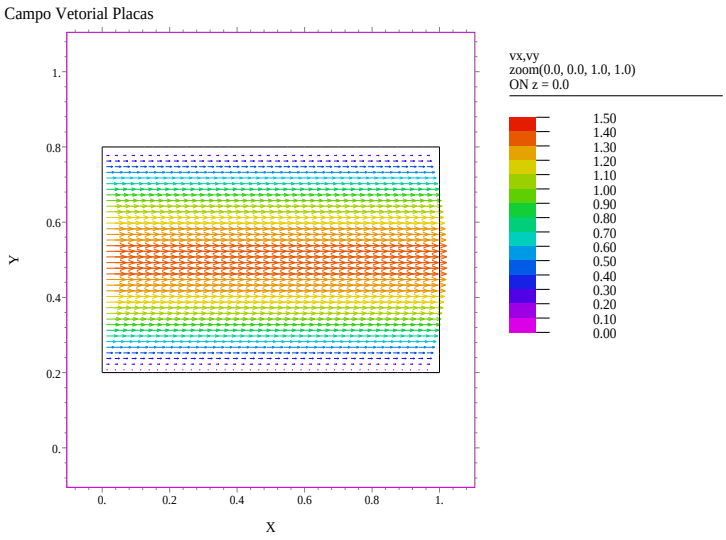

Figura 4.3: Corte XY do campo vetorial do meio extratificado.

Com os mesmos parâmetros térmicos e de porosidade utilizados na obtenção do $K_{\|}$ analítico, geramos a quarta coluna da tabela 4.1, onde as variações de dispersão estão entre 1% e 5%.

Tabela 4.1: Dispersões longitudinais em função de *Péclet*

Pe	$K_{\|}$ analítico (4.2)	$K_{\|}$ campo (4.1)	$K_{\|}$ campo FlexPDE
1	1.009669e+000	9.912940e-001	9.915079e-001
2	1.038674e+000	1.019225e+000	1.020090e+000
5	1.241714e+000	1.221903e+000	1.227350e+000
10	1.966857e+000	1.955980e+000	1.977842e+000
20	4.867429e+000	4.910186e+000	4.997594e+000
50	2.517143e+001	2.566122e+001	2.620806e+001
70	4.837600e+001	4.940226e+001	5.012096e+001
100	9.768571e+001	9.987433e+001	1.020880e+002
200	3.877429e+002	3.904925e+002	3.892248e+002
500	2.418143e+003	2.375570e+003	2.461121e+003
800	6.188886e+003	6.067638e+003	6.229259e+003
1000	9.669571e+003	9.294803e+003	9.759222e+003

Pelos testes realizados foi verificado que a implementação gerou resultados coerentes para $K_{\parallel}$, quando comparados com a solução analítica. A seguir mostraremos os resultados das simulações usando a implementação em conjunto com o software FlexPDE como ferramenta para obtenção do campo vetorial de um fluido em meios onde a célula fundamental é complexa e sendo assim a solução analítica do comportamento de $K_{\parallel}$ não é conhecida.

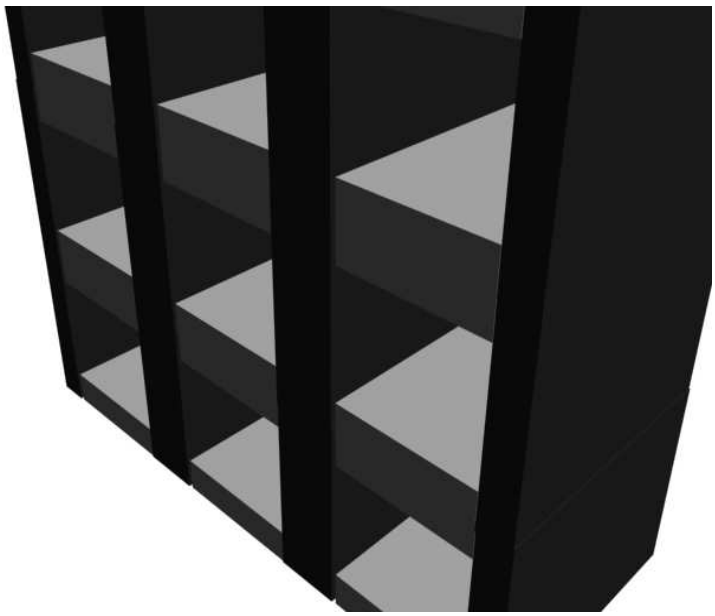

Figura 4.4: Meio constituído por tubos.

4.2 Geometrias de célula fundamental adotadas

O software utilizado para o cálculo dos campos de velocidade não é muito adequado ao caso do meio periódioco aqui estudado. Assim, as geometrias escolhidas, apesar de simples, foram as satisfatórias às análises requeridas neste trabalho. Como a nossa finalidade é de explorar os aspectos computacionais do método dos térmions, o cálculo do campo vetorial e sua complexidade são mais ilustrativas do que fundamentais neste trabalho. As geometrias foram chamadas de Tubo, Tubo com obstáculo, Tubo com obstáculo e abertura e Tubo com obstáculo e aberturas. Em todas as simulações utilizamos os seguintes valores para as propriedades térmicas, $k_\beta = 1$, $k_\sigma = 1$, $(\rho c_p)_\beta = 1$ e $(\rho c_p)_\sigma = 1$.

- **Tubo.** Esta célula fundamental constituída por 4 blocos é vista na figura 4.5. Aqui o meio poroso foi simulado como um feixe de tubos de seção quadrada como visto na figura 4.4, onde a contribuição da convectividade se deu longitudinalmente. Os

gráficos mostrando o campo de velocidades do escoamento pode ser vistos nas figuras A.1, A.2. Utilizamos os seguintes valores de porosidades $\varepsilon_\beta = 0.6$, $\varepsilon_\sigma = 0.4$, com isso obtivemos a tabela 4.2 que fornece os valores calculados da componente longitudinal do tensor de dispersão térmica ($K_\parallel$) em função no número de *Péclet* (Pe).

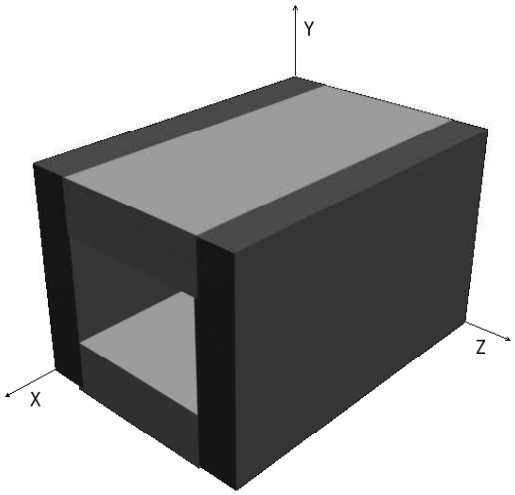

Figura 4.5: Célula tubo.

Tabela 4.2: Célula tubo.

Pe	$K_\parallel$
1	1.006536e+00
2	1.045535e+00
5	1.175834e+00
10	1.686731e+00
20	3.973521e+00
50	1.903797e+01
70	3.587344e+01
100	7.672094e+01
200	2.909409e+02
500	1.824659e+03
800	4.605522e+03
1000	7.054442e+03

- **Tubo com obstáculo.** Adicionamos à célula fundamental um obstáculo como mostrada na figura 4.6, esta nova célula é constituída por 5 blocos. Com o obstáculo teremos o fluxo mais complexo devido à assimetria transversal, e com isso espera-se uma inomogeniedade na dispersão longitudinal. Os gráficos mostrando o campo de velocidades do escoamento pode ser vistos nas figuras A.5, A.6. Foram utilizados

os seguintes valores das porosidades, $\varepsilon_\beta = 0.596$, $\varepsilon_\sigma = 0.404$, e com isso obtivemos a tabela 4.3.

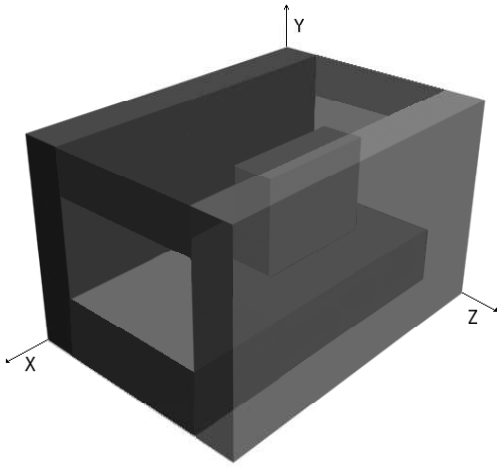

Figura 4.6: Célula tubo com obstáculo.

Tabela 4.3: Célula tubo com obstáculo.

Pe	$K_\parallel$
1	9.938758e-01
2	1.036614e+00
5	1.159779e+00
10	1.765047e+00
20	3.851574e+00
50	1.909246e+01
70	3.714999e+01
100	7.343402e+01
200	3.004560e+02
500	1.787599e+03
800	4.320824e+03
1000	6.496311e+03

- **Tubo com obstáculo e abertura.** Partindo da célula tubo com obstáculo, adicionamos aberturas como mostrada na figura 4.7, gerando assim uma nova célula constituída por 7 blocos. Criamos uma maior complexidade no fluxo do escoamento, com isso espera-se uma alteração no valor da dispersão longitudinal, nos interessa avaliar como esta situação afeta a performance da implementação. Os gráficos mostrando o campo de velocidades do escoamento pode ser vistos nas figuras A.9, A.10. Utilizamos os seguintes valores das porosidades, $\varepsilon_\beta = 0.627$, $\varepsilon_\sigma = 0.373$, que ao final das simulações obtivemos a tabela 4.4.

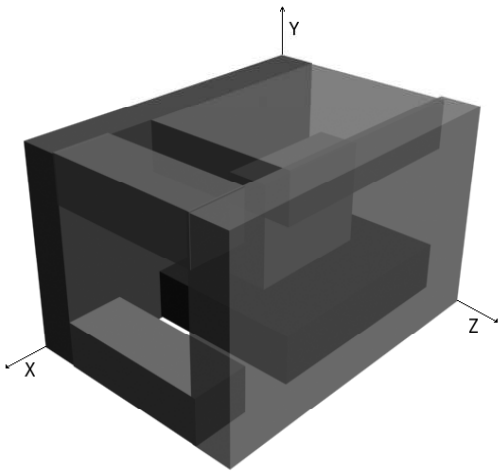

Figura 4.7: Célula tubo com obstáculo e abertura.

Tabela 4.4: Célula tubo com obstáculo e abertura.

Pe	$K_{\parallel}$
1	1.028065e+00
2	1.039820e+00
5	1.174209e+00
10	1.807177e+00
20	4.356870e+00
50	2.209312e+01
70	4.157277e+01
100	8.432117e+01
200	3.328708e+02
500	1.903001e+03
800	4.359583e+03
1000	6.429779e+03

- **Tubo com obstáculo e aberturas.** Adicionamos aberturas laterais a célula tubo com obstáculo e abertura, e assim geramos uma nova célula constituída por 7 blocos como mostrada na figura 4.8. Criamos aberturas laterais maiores que as superiores e apesar do fluxo do escoamento ser mais complexo, espera-se uma maior homogêniedade na dispersão transversal quando as simulações estiverem com valores de Pe altos. Queremos avaliar com isso o comportamento da implementação a partir de uma célula fundamental que permita dispersão em todas as direções. Os gráficos mostrando o campo de velocidades do escoamento pode ser vistos nas figuras A.13, A.14. Com os valores das porosidades, $\varepsilon_\beta = 0.626$, $\varepsilon_\sigma = 0.374$, obtivemos a tabela 4.5.

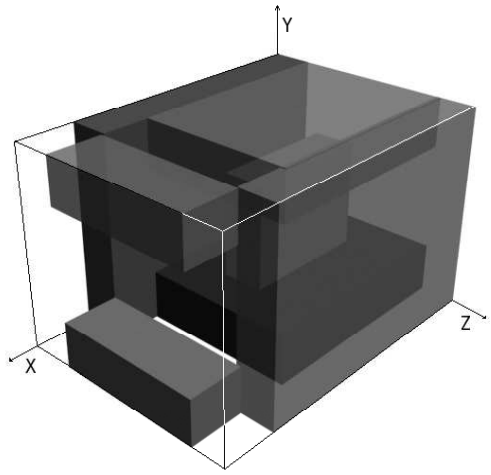

Figura 4.8: Célula tubo com obstáculo e aberturas.

Tabela 4.5: Célula tubo com obstáculo e aberturas.

Pe	$K_\parallel$
1	1.028065e+00
2	1.012890e+00
5	1.161219e+00
10	1.812327e+00
20	4.331212e+00
50	2.182332e+01
70	4.122342e+01
100	8.403734e+01
200	3.312358e+02
500	1.893201e+03
800	4.332312e+03
1000	6.434533e+03

4.3 Análises

É sabido da literatura que para $Pe < 5$ há contribuição predominantemente difusiva no processo de dispersão, em $5 < Pe < 100$, a difusão e convecção competem, e para $Pe > 100$ a maior contribuição é proveniente do tensor de dispersão hidrodinâmica, estes comportamentos são verificados nos gráficos das figuras A.3, A.7, A.11, A.15 que representam a $K_{\|}$ em função de Pe obtidos das simulações. O coeficiente longitudinal é apresentado na literatura como $K_{\|} \approx APe^m$, onde A é uma constante e m é a potência de *Péclet*. Realizadas as simulações extraimos os valores de m de cada configuração de célula fundamental, que podem ser vistos na tabela 4.6, esses valores nos informam um comportamento típico da dispersão de Taylor.

Na figura A.4 observamos o gráfico com o comportamento das componentes transversais da dispersão no caso célula fundamental tubo, e verificamos que a difusão é o único fator de dispersão transversal para todos os valores de Pe, idem para os casos, tubo com obstáculo A.8, tubo com obstáculo e abertura A.12, e tubo com obstáculo e aberturas A.16.

Tabela 4.6: Potências de *Péclet* obtidas em relação a geometria de célula fundamental.

Tipo de Geometria	Valores de m
Tubo	1.95
Tubo Obstáculo	1.94
Tubo Obstáculo Abertura	1.93
Tubo Obstáculo Aberturas	1.93

4.4 Análise do uso da função de HASH

No Capítulo 4 fizemos uma análise sobre a estratégia de busca baseada em tabela HASH e chegamos a conclusão que seu desempenho é melhor para situações onde a quantidade de blocos que compõem a célula fundamental é superior a 3. Nas simulações contidas neste capítulo as células tiveram as seguintes quantidades de blocos 4, 20 e 40 (tubo), 5, 20 e 40 (tubo com obstáculo), 7, 20 e 40 (tubo com obstáculo e abertura), 7 (tubo com obstáculo e aberturas), a tabela 4.7 fornece o percentual de diminuição no tempo de processamento em função da quantidade de blocos. Cada percentual foi obtido pela média da soma dos tempos de simulações com os números de Pe adotados. Houve uma diferença no percentual que pode ser considerada como uma flutuação, que é oriunda do fato de que estamos trabalhando com uma técnica probabilística.

O uso da busca em HASH provou ser uma técnica eficiente, nos casos testados ela manteve um percentual de diminuição de tempo de processamento bastante expressivo.

Tabela 4.7: % diminuição de tempo em relação ao tipo de célula fundamental e a quantidade de blocos.

Célula fundamental	Quantidade blocos	Diminuição tempo processamento
Tubo	4	19.89%
Tubo com obstáculo	5	20.88%
Tubo com obstáculo e abertura	7	22.88%
Tubo com obstáculo e aberturas	7	22.92%
Tubo	20	32.55%
Tubo com obstáculo	20	32.34%
Tubo com obstáculo e abertura	20	32.28%
Tubo	40	39.80%
Tubo com obstáculo	40	39.46%
Tubo com obstáculo e abertura	40	39.42%

4.5 Consumo de memória

Na implementação quando rodamos o experimentos com os parâmetros de porosidade $k_\beta = 1$, $k_\sigma = 1$, $(\rho c_p)_\beta = 1$ e $(\rho c_p)_\sigma = 1$, e $Pe = 1$, utilizando a busca seqüêncial o gasto em memória é de $17,4$ Mbytes, enquanto que utilizando a busca em tabela HASH o gasto é de 18.4 Mbytes. Modificando a condutividade para $k_\beta = 2$, o gasto de memória é de 25.6 Mbytes para a busca seqüêncial e 26.6 Mbytes para busca em tabela HASH. Mantendo o parâmetro de porosidade $k_\beta = 1$, e utilizando $Pe = 1000$ o gasto em memória é de 202.6 Mbytes para busca sequencial e 203.8 Mbytes para busca em tabela HASH, mantendo Pe com o mesmo valor e fazendo o parâmetro de porosidade $k_\beta = 2$, obteve-se 398.3 Mbytes na busca seqüêncial e 399.3 Mbytes na busca em tabela HASH. Como podemos ver o gasto de memória pelo uso de tabela HASH é acrescido em 1 Mbyte e que valores altos para Pe e valores distintos de propriedades térmicas do sólido e fluido fazem com que a implementação tenha que alocar mais memória e gaste mais tempo de processamento devido ao fato de estarem diretamente ligados ao cálculo do valor de δr_d^*.

4.6 Conclusão Geral

Neste trabalho implementamos o algoritimo da referência [1] para meios tridimensionais, entretanto a implementação original gastava em torno de 50% do tempo total de processamento para localizar um térmion. Então analisamos estratégias de diminuição

de tempo de localização e o quanto isso afetaria o consumo de memória. Constatamos que o uso da tabela HASH provou ser a melhor estratégia de localização porque além de diminuir o tempo de processamento não acarretou um gasto expressivo de memória nos casos trabalhados, contudo isso foi possível devido a utilização de uma função bijetora para a tabela HASH.

Recomendamos as seguintes propostas de continuidade deste trabalho:

- Implementação total ou parcial do algoritimo em GPUs [27], [28];
- Criação de células fundamentais mais complexas;
- Criação de uma estratégia de implementação de δr variável;
- Elaborar uma melhor estratégia para o valor máximo de incremento δr_d^* pois o número de passos reflete diretamente no tempo de processamento, como pode ser visto abaixo.

$$\delta r_d^* = \frac{2}{v_{max}^*}\left[\sqrt{1 + cv_{max}^*\Delta} - 1\right]$$

APÊNDICE A – Figuras

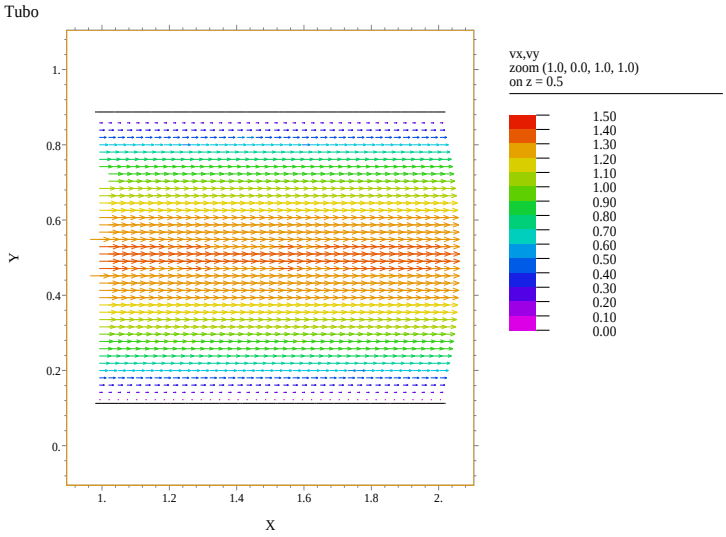

Figura A.1: Escoamento XY, Tubo.

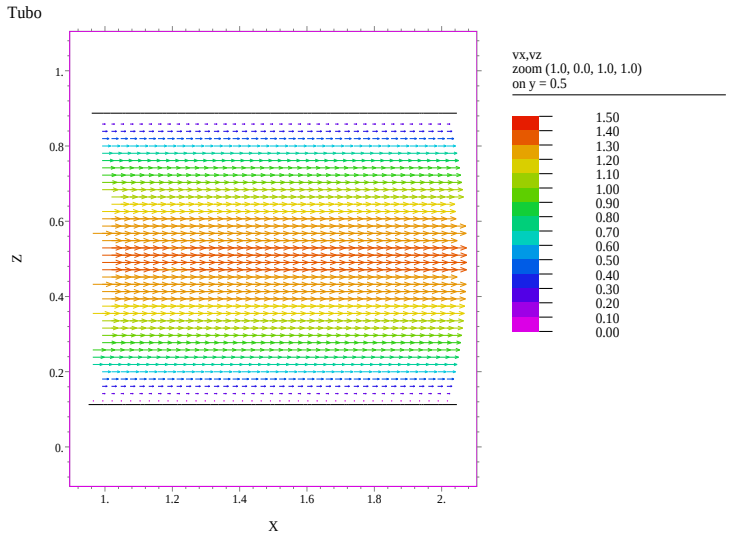

Figura A.2: Escoamento XZ, Tubo.

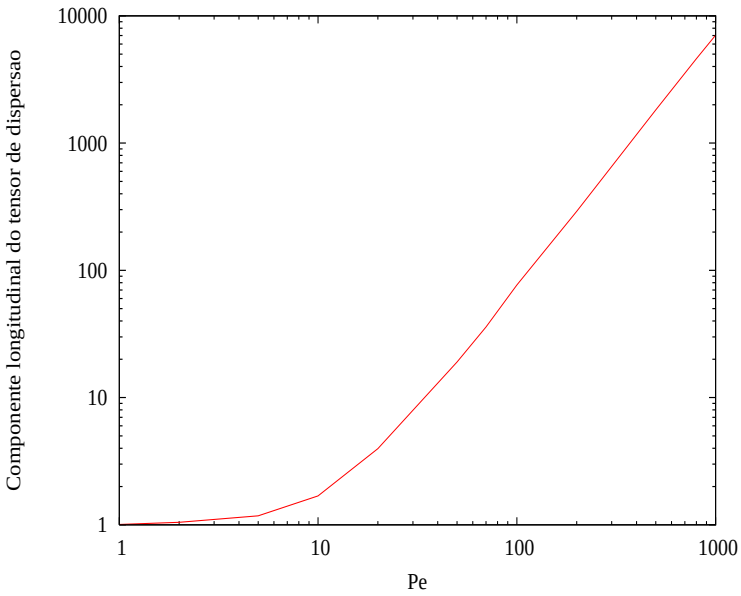

Figura A.3: $K_{\parallel}$ em função de Pe, Tubo.

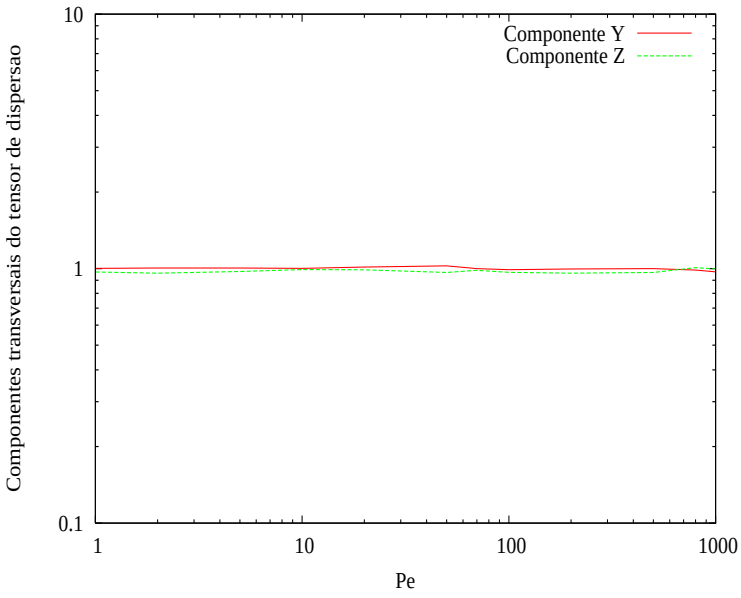

Figura A.4: Componentes transversais Y e Z em função de Pe, Tubo.

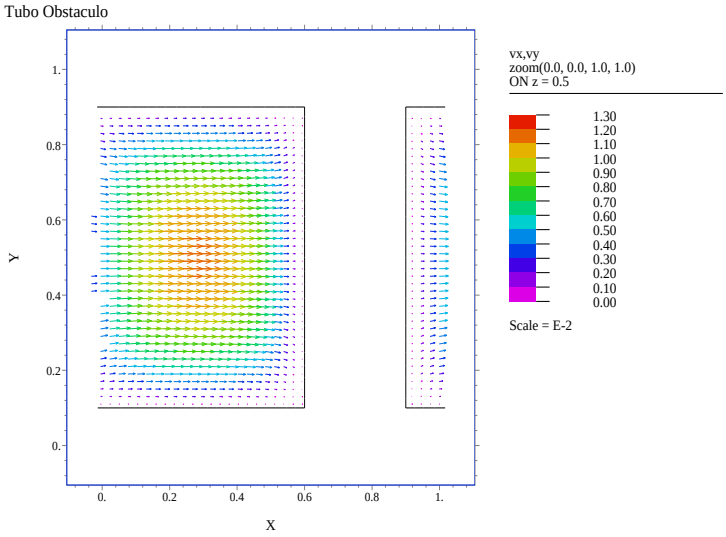

Figura A.5: Escoamento XY, Tubo com Obstáculo.

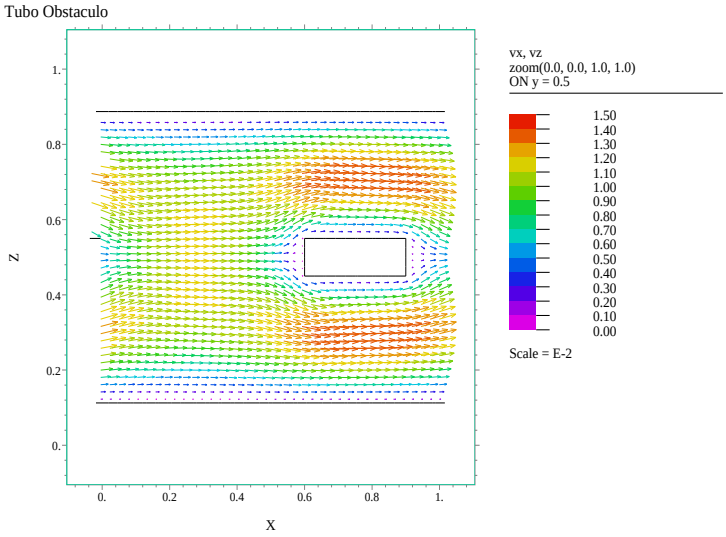

Figura A.6: Escoamento XZ, Tubo com Obstáculo.

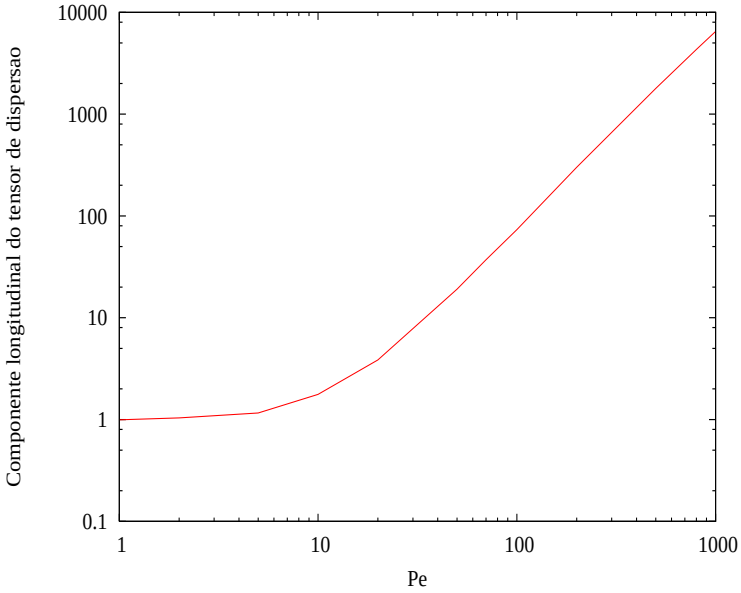

Figura A.7: $K_{\|}$ em função de Pe, Tubo com Obstáculo.

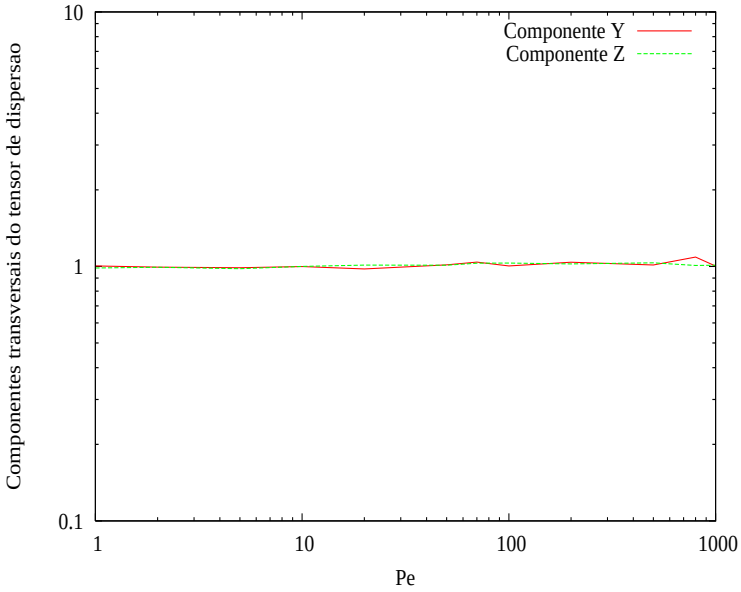

Figura A.8: Componentes transversais Y e Z em função de Pe, Tubo com Obstáculo.

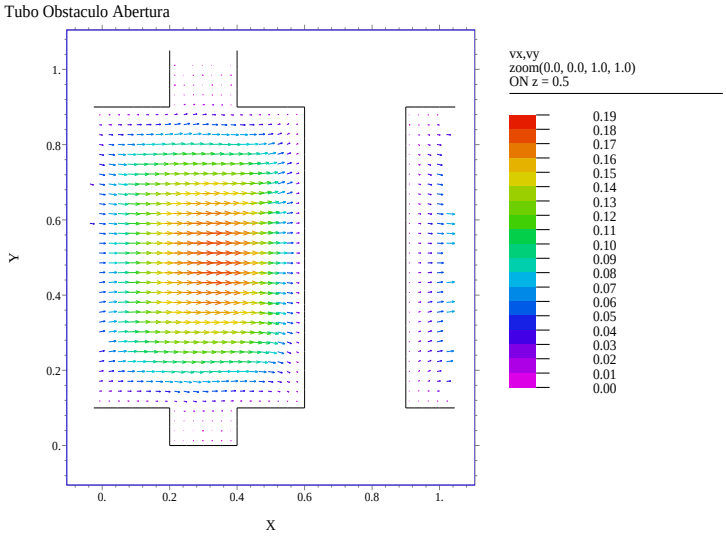

Figura A.9: Escoamento XY, Tubo com Obstáculo e Abertura.

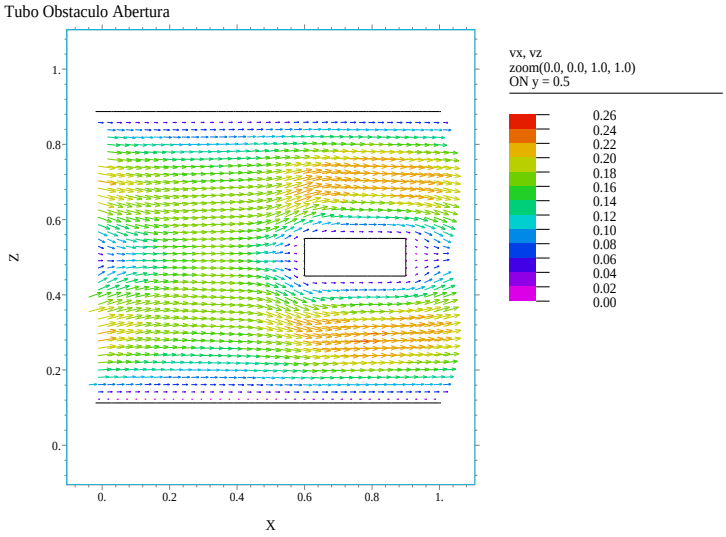

Figura A.10: Escoamento XZ, Tubo com Obstáculo e Abertura.

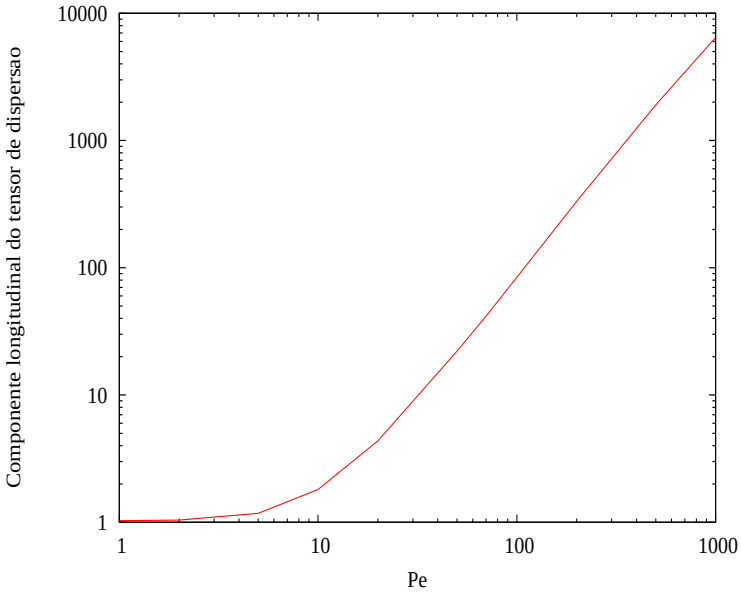

Figura A.11: $K_{\parallel}$ em função de Pe, Tubo com Obstáculo e Abertura.

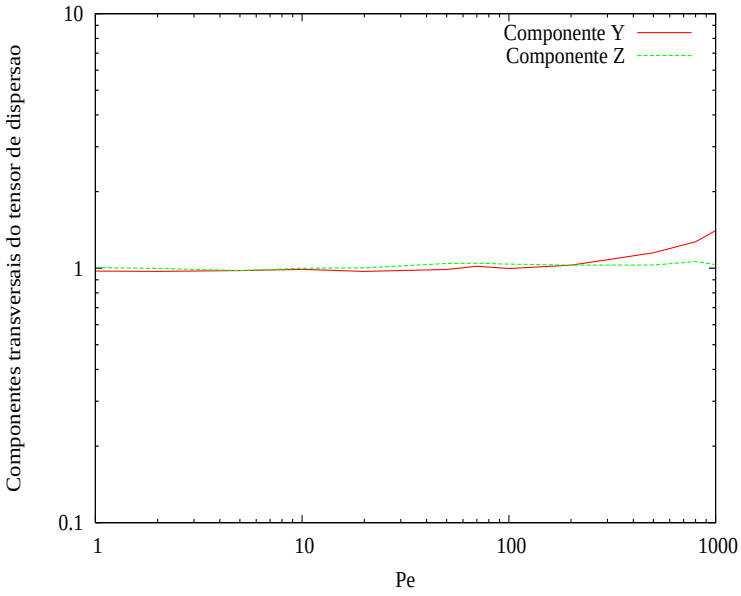

Figura A.12: Componentes transversais Y e Z em função de Pe, Tubo com Obstáculo e Abertura.

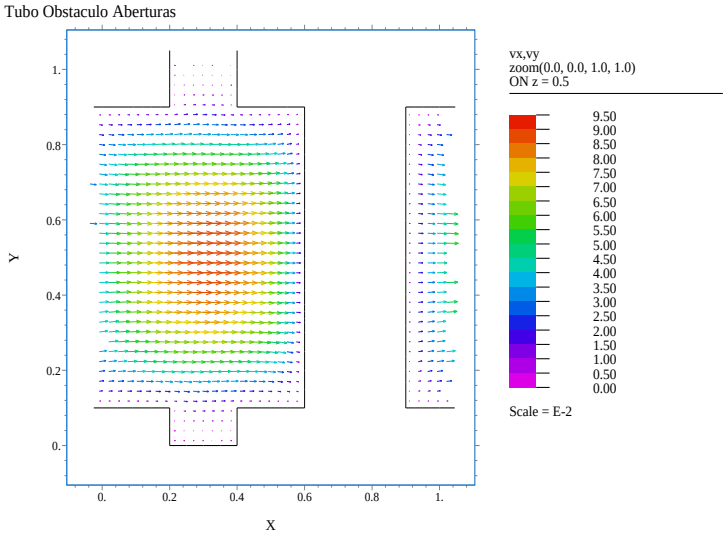

Figura A.13: Escoamento XY, Tubo com Obstáculo e Aberturas.

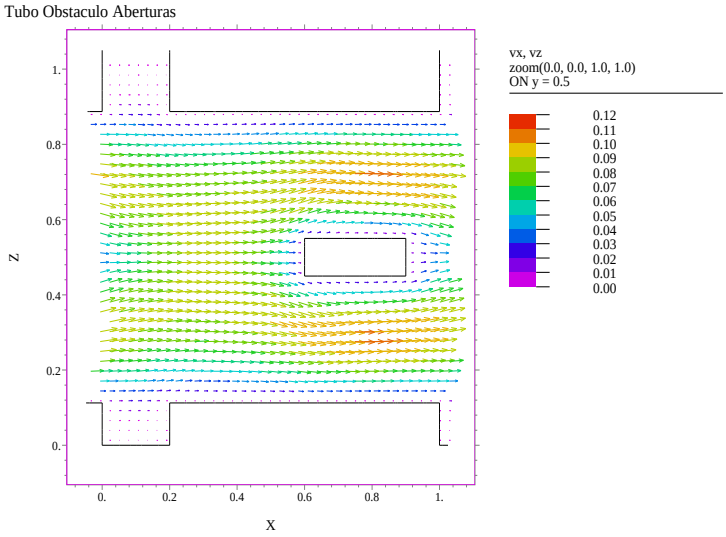

Figura A.14: Escoamento XZ, Tubo com Obstáculo e Aberturas.

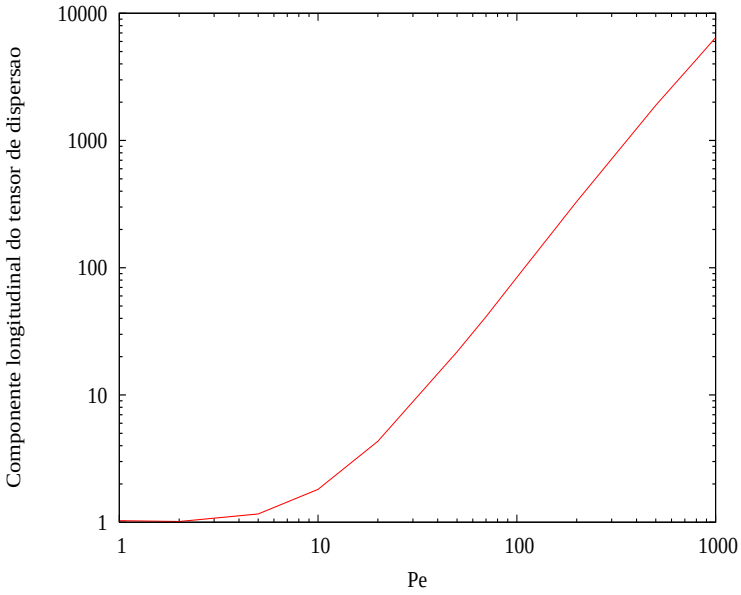

Figura A.15: $K_{\|}$ em função de Pe, Tubo com Obstáculo e Aberturas.

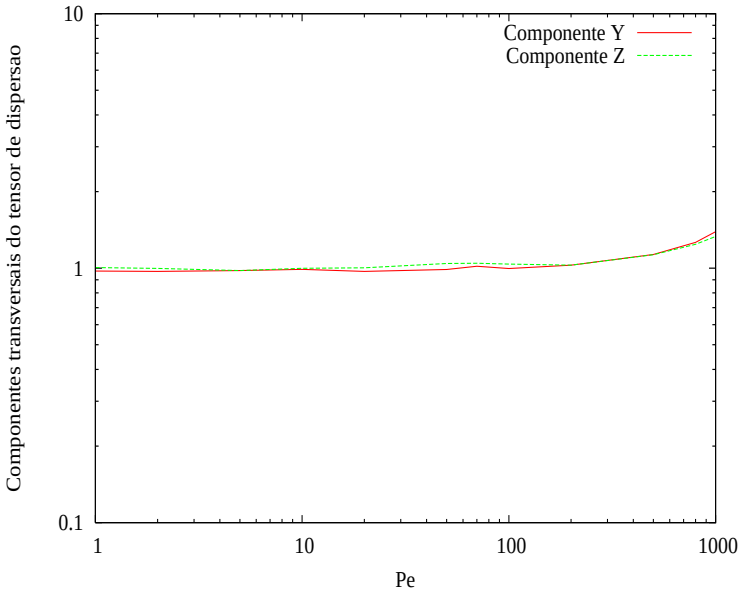

Figura A.16: Componentes transversais Y e Z em função de Pe, Tubo com Obstáculo e Aberturas.

APÊNDICE B – Codigo

Listagem B.1: termion04bbee3d-010.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <float.h>
5  #include <values.h>
6  #include <string.h>
7  #include <fcntl.h>
8  #include <unistd.h>
9
10 #define SIM 1
11 #define NAO 0
12 #define NORMAL 1
13 #define END     0
14
15 /* TIPO DE PESQUISA -> SIM = HASH, NAO = SEQUENCIAL */
16 #define HASH NAO
17 #define DIFUSAO    SIM
18 #define CONVECCAO SIM
19
20 #ifndef MAXDOUBLE
21     #define MAXDOUBLE 1.79769313486231570e+308
22 #endif
23
24 #ifndef PI
25     #define   PI    3.1415926524
26 #endif
27
28 #define   DPI   6.283185307
29 #define   ZERO 0.00000001
30 #define   MASK 1000000000.0
31 #define   LIMITE 0.00001
32 #define   PARMAX 1.0
```

```
33 #define  POSITIVO  1
34 #define  NEGATIVO -1
35
36 #define A11 16807
37 #define M11 2147483647
38 #define Q11 127773
39 #define R11 2836
40
41 #define A1 40014
42 #define A2 40692
43 #define M1 2147483563
44 #define M2 2147483399
45 #define MM1 (M1 - 1)
46 #define Q1 53668
47 #define Q2 52774
48 #define R1 12211
49 #define R2 3791
50
51 #define SEED1 -13241277
52 #define SEED2 -45678131
53
54 #define NTAB 32
55 #define EPS  1.0/MAXDOUBLE
56 #define RNMX (1.0 - EPS)
57
58 #define NMAXBLOCOS 40
59 #define N          70
60
61 #define BUFFERSIZE 10240
62
63 #define NL 0x0A
64 #define SP 0x20
65
66 /* ------------------------ Macros ------------------------ */
67 #define limpa(d) (floor(MASK * (d))/MASK + ZERO)
68 /* Elimina algarismos menos significativos de um numero */
69 /* Objetivo: Eliminar parte do ruido numerico           */
70 /* #define ffabs(x) (( x < 0.0) ? -x : x)               */
71 /* Determina o valor absoluto. Objetivo: Otimizacao do  */
72 /* tempo de execucao.                                   */
73 /* -------------------------------------------------------- */
74
75 typedef struct vector Vector;
```

```
76  typedef struct tensor2 Tensor2;
77  typedef struct tensor3 Tensor3;
78  typedef struct bloco  Bloco;
79  typedef struct cel    Cel;
80
81  typedef enum{DOUBLE, FLOAT, INT, CHAR, VECTOR, TENSOR2, NON} Dado;
82  typedef enum{FLUIDO, SOLIDO, POROSO} Fase;
83  typedef enum{UNIFORME, PONTUAL, FAIXA} Distribuicao;
84  typedef enum{X, Y, Z} Direcao;
85  typedef enum{OK, KO} Ok;
86  typedef struct es
87  {
88     Ok pos, mom, vel, disp;
89  } Es;
90
91  struct vector
92  {
93     double x, y, z;
94  };
95
96  struct tensor2
97  {
98     double xx, xy, xz,
99            yx, yy, yz,
100           zx, zy, zz;
101 };
102
103 struct tensor3
104 {
105    double xxx, xxy, xxz,
106           xyx, xyy, xyz,
107           xzx, xzy, xzz,
108           yxx, yxy, yxz,
109           yyx, yyy, yyz,
110           yzx, yzy, yzz,
111           zxx, zxy, zxz,
112           zyx, zyy, zyz,
113           zzx, zzy, zzz;
114 };
115
116 struct bloco
117 {
118    Vector ini, fin;
```

```
119 };
120
121 struct cel
122 {
123    double size;
124    Vector ini, fin;
125    Bloco  posicao[NMAXBLOCOS];
126 };
127
128 /* ------------------------------------------------------------ */
129 /*  Metodo dos momentos. Tecnica dos termions                   */
130 /*  Dispersao de calor num fluxo entre placas paralelas         */
131 /*  apresentado como meio periodico na direcao x e y            */
132 /* ------------------------------------------------------------ */
133 /* Esta versao contem:                                          */
134 /*                        Conservacao do tempo                  */
135 /*                        Direcao alternada                     */
136 /*                        Em caso de termion na fronteira       */
137 /*                        ele faz o movimento "completo"        */
138 /* ------------------------------------------------------------ */
139
140 /* Prototipos */
141 Fase   onde(Vector red, Cel celula);
142 void   postored (Vector pos, Vector *red);
143 void   postored2 (Vector pos, Vector *red);
144 double recalcula(Vector red, Direcao d, Cel celula);
145 void   *dalloc(Dado d, long int n);
146 void   velocidade_p(Vector p, Vector v[][N][N], int n, Vector d, Vector *vv
    );
147 void   autovalor(Tensor2 t, Tensor2 *k, double *ang_xy, double *ang_xz,
    double *ang_yz);
148
149 double matmax(Vector v[][N][N], int n);
150 double matmed(Vector v[][N][N], int n);
151
152 /* INI Hash */
153 unsigned int total_mem     = 0,
154              pts           = 0,
155              pontos        = N, /* QUANTIDADE DE PONTOS EM HASH + 1*/
156              discretizacao = N - 1, /* QUANTIDADE DE PONTOS EM HASH */
157              tam_char  = sizeof(char);
158 char *end_base;
159 int alocar_memoria();
```

```
160 int plotar_hash(int estado);
161 /* FIN Hash */
162
163 Fase   onde_g_hash(Vector red, Cel celula, int nblocos, int *bloco);
164 Fase   onde_g(Vector red, Cel celula, int nblocos, int *bloco);
165 Fase   testabloco(Vector red, Bloco b);
166
167 double ran1(long int *i);
168 double ran2(long int *i);
169
170 double minimo(double v[], int n);
171 double maximo(double v[], int n);
172
173 void limpanom(char nom[]);
174 FILE *openfile(char nom[], char mode[]);
175 int  acha(char o_que, char *buff, int inicio);
176 void transf(char *saida, char *buff, int inicio, int fim);
177
178 void problemas(Direcao dir, Fase d, Fase a,
179                Vector red0, Vector red, Vector pos0, Vector pos,
180                double par, double tempo);
181
182 void  miniquad(Vector v[], long int n, double *a0, double *a1);
183 double miniquad_a(Vector v[], long int n);
184 double miniquad_c(Vector v[], long int n);
185
186 /* Variaveis INI */
187    char posicao, momentos, velocidade, dispersao;
188
189    FILE *out_mom, *out_dif, *out_pos = 0, *out_vels,
190         *in_geo, *in_vel;
191
192    int in_dateco,
193        inicio, passo, tamanho, maxaj,
194        malha;
195
196    int i, j, k, bloco = 0, nblocos, ilixo, sinal;
197
198    long int nt, impr, nimpr, np, npimp;
199
200    long int it, in;
201
202    long int seed1, seed2;
```

```
203
204     double xlixo, ylixo, zlixo;
205
206     double conds, condf,
207            roc, rocs, rocf, ni,
208            eps, epf,
209            difs, diff, rdifs,
210            efs, eff,
211            dr, drs, drf,
212            probsf, probfs, fator,
213            dt, dvr = 0, deltar, temps, tmax,
214            xm, xm2, xm3,
215            ym, ym2, ym3,
216            zm, zm2, zm3,
217            xym, xm2y, xmy2,
218            xzm, xm2z, xmz2,
219            yzm, ym2z, ymz2,
220            vm, Pe, kvm, rb,
221            vmax, vmed, xmax = 0, xmin = 0, ymax = 0, ymin = 0, zmax = 0,
                   zmin = 0,
222            angulo_xy, angulo_xz, angulo_yz;
223
224     double a0, a1, a_a, a_c;
225
226     double *tps, *x, *x2, *x3,
227            *y, *y2, *y3,
228            *z, *z2, *z3,
229            *xy, *x2y, *xy2,
230            *xz, *x2z, *xz2,
231            *yz, *y2z, *yz2,
232            *xyz,
233            *vaux;
234
235     Vector pos, pos0, d, fonte, faixa,
236            red, red0, redi, vv, av,
237            *v, *ajus, *rpos = 0;
238
239     Vector vel[N][N][N];
240
241     Tensor2 *dif, a, D;
242
243     Cel celula;
244
```

```
245     Fase depart, arrivee;
246
247     Direcao dir;
248
249     Ok ondefonte;
250
251     Distribuicao como;
252
253     Es saida;
254
255     float um_terco = 1.0 / 3.0, dois_tercos = 2.0 / 3.0;
256
257     float valor;
258
259     float base = 0.0, altura = 0.0, area = 0.0, volume = 0.0, espessura =
            0.0, volume_acm = 0.0;
260
261     char nom[33],
262          nom_arq[33],
263          nom_dif[33],
264          nom_mom[33],
265          nom_pos[33],
266          nom_geo[33],
267          nom_vel[33],
268          nom_vels[33],
269          numero[6],
270          *buff;
271
272     float x_ini[5], y_ini[5], z_ini[5],
273           x_fin[5], y_fin[5], z_fin[5];
274 /* Variaveis FIM */
275
276 int main(void)
277 {
278     /* ---------------------------------- Programa ------------------------------------
            */
279     seed1 = SEED1; /* Inicializacao dos geradores de numeros aleatorios */
280     seed2 = SEED2;
281
282     fscanf(stdin, "%s\n", nom); /* Leitura do nome base do arquivo de saida
            e */
283     limpanom(nom);              /* retira a extensao do nome base, caso haja
             */
```

```
284
285     /* Gera os nomes dos arquivos de saida -> */
286     strcpy(nom_dif, nom);  strcat(nom_dif,  ".dif"); /* Arquivo com tensor
            de dispersao    */
287     strcpy(nom_mom, nom);  strcat(nom_mom,  ".mom"); /* Arquivo com momentos
                        */
288     strcpy(nom_vels, nom); strcat(nom_vels, ".vel"); /* Arquivo com a
            velocidade simulada */
289     strcpy(nom_pos,  nom); strcat(nom_pos,  ".pos"); /* Arquivo contendo
            distribuicao final de termions*/
290     /* Arquivos de saida <- */
291
292     /* ------------------------ Entrada de dados  numericos
            ------------------------ */
293     fscanf(stdin,"%ld\n%lf\n%ld\n", &np, &tmax, &nimpr);
294                                     /* Entrada de :              */
295                                     /* Numero de particulas,     */
296                                     /* tempo maximo de simulacao */
297                                     /* intervalo de amostra      */
298
299     /* Propriedades dos meios */
300     fscanf(stdin,"%le\n%le\n%le\n%le\n", &conds, &rocs, &condf, &rocf);
301                                       /* Entrada de:  */
302                                       /* Condutividade termica do solido
                                             */
303                                       /* Capacidade termica do solido
                                             */
304                                       /* Condutividade termica do fluido
                                             */
305                                       /* Capacidade termica do fluido
                                             */
306
307     fscanf(stdin,"%le\n%le\n%le\n", &vm, &Pe, &dr);
308                                    /* Entrada de:  */
309                                    /* Velocidade media do campo de
                                          velocidades */
310                                    /* Numero de Peclet
                                                                  */
311                                    /* Passo no espaco
                                                                    */
312
313     fscanf(stdin,"%le\n%le\n%le\n%le\n%le\n%le\n%le\n", &(celula.size),
```

```
314                                                    &(celula.ini.x), &(
                                                           celula.ini.y), &(
                                                           celula.ini.z),
315                                                    &(celula.fin.x), &(
                                                           celula.fin.y), &(
                                                           celula.fin.z));
316                                                    /* Entrada de:  */
317                                                    /* Tamanho da celula
                                                            e          */
318                                                    /* cantos inferior
                                                           esquerdo e */
319                                                    /* superior direito
                                                           da mesma  */
320
321     fscanf(stdin,"%le\n%le\n%le\n", &(fonte.x), &(fonte.y), &(fonte.z));
322
323     fscanf(stdin, "%le\n", &fator);
324
325     fscanf(stdin, "%s\n", nom_arq);
326
327     fscanf(stdin, "%c\n%c\n%c\n%c", &posicao, &momentos, &velocidade, &
            dispersao);
328
329     printf("\n ---------------> %c %c %c %c \n", posicao, momentos,
            velocidade, dispersao);
330
331     #if HASH == NAO
332     fprintf(stdout, "\n MODO : SEQUENCIAL\n");
333     #endif
334
335     #if HASH == SIM
336     fprintf(stdout, "\n MODO : HASH\n\n");
337     pts = pontos;
338     alocar_memoria();
339     #endif
340
341     posicao    = toupper(posicao);
342     momentos   = toupper(momentos);
343     velocidade = toupper(velocidade);
344     dispersao  = toupper(dispersao);
345
346     saida.pos  = (posicao    == 'S') ? OK: KO;
347     saida.mom  = (momentos   == 'S') ? OK: KO;
```

```
348     saida.vel  = (velocidade == 'S') ? OK: KO;
349     saida.disp = (dispersao  == 'S') ? OK: KO;
350
351     buff = (char *)dalloc(CHAR, BUFFERSIZE);
352
353     if((in_dateco = open(nom_arq, O_RDONLY)) == -1)
354     {
355        puts("main: Problemas ao abrir arquivo : \n");
356        printf("%s\n",nom_arq);
357        exit(NORMAL);
358     }
359
360     tamanho = read(in_dateco, buff, BUFFERSIZE);
361
362     buff[tamanho] = '\0';
363
364     inicio = 0;
365
366     /* Determinacao do nome do arquivo contendo o campo de velocidades */
367     inicio = acha(NL, buff, inicio);
368     inicio = acha(NL, buff, inicio);
369     passo  = acha(SP, buff, inicio);
370
371     transf(nom_vel, buff, inicio, passo - 1);
372
373     /* Determinacao do nome do arquivo contendo a geometria da celula */
374     inicio = acha(NL, buff, passo);
375     passo  = acha(SP, buff, inicio);
376
377     transf(nom_geo, buff, inicio, passo - 1);
378
379     printf("\n Lendo informacoes de: '%s''%s'\n", nom_vel, nom_geo);
380
381     /* Lendo uma das dimensoes da malha */
382     inicio = acha(NL, buff, passo);
383     inicio = acha(NL, buff, inicio);
384     inicio = acha(NL, buff, inicio);
385
386     while (isspace(buff[inicio])) inicio++;
387
388     passo = acha(SP, buff, inicio);
389
390     transf(numero, buff, inicio, passo);
```

```
391
392     malha = atoi(numero);
393
394     /* Pegando o numero de blocos solidos da celula elementar */
395     inicio = acha(NL, buff, inicio);
396
397     while (isspace(buff[inicio])) inicio++;
398
399     passo = acha(SP, buff, inicio);
400
401     transf(numero, buff, inicio, passo);
402
403     nblocos = atoi(numero);
404
405     close(in_dateco);
406
407     free(buff);
408
409     fprintf(stdout, "\n n. de blocos %d\n", nblocos);
410
411     /* "Tamanho" da celula no campo de velocidades. Caso especial de mesmas
            dimensoes */
412     d.x = d.y = d.z = celula.size/((double) malha);
413
414     fprintf(stdout, "\n d = (%le, %le, %le)\n", d.x, d.y, d.z);
415
416     in_geo = openfile(nom_geo, "r");
417
418     #if HASH == SIM
419     for (i = 0; i < nblocos; i++)
420     {
421         fscanf(in_geo, "%le %le %le", &(celula.posicao[i].ini.x), &(celula.
                posicao[i].ini.y), &(celula.posicao[i].ini.z));
422         x_ini[0] = celula.posicao[i].ini.x;
423         y_ini[0] = celula.posicao[i].ini.y;
424         z_ini[0] = celula.posicao[i].ini.z;
425
426         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
427         x_ini[1] = xlixo;
428         y_ini[1] = ylixo;
429         z_ini[1] = zlixo;
430
431         printf("\nBloco %d\n", i + 1);
```

```
432         base = xlixo - celula.posicao[i].ini.x;
433         printf("base      = %f\n", base);
434         altura = celula.posicao[i].ini.y;
435 
436         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
437         x_ini[2] = xlixo;
438         y_ini[2] = ylixo;
439         z_ini[2] = zlixo;
440 
441         altura = ylixo - altura;
442         printf("altura    = %f\n", altura);
443         area = base * altura;
444         printf("area      = %f\n", area);
445         espessura = zlixo;
446 
447         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
448         x_ini[3] = xlixo;
449         y_ini[3] = ylixo;
450         z_ini[3] = zlixo;
451 
452         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
453         x_ini[4] = xlixo;
454         y_ini[4] = ylixo;
455         z_ini[4] = zlixo;
456 
457         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
458         x_fin[0] = xlixo;
459         y_fin[0] = ylixo;
460         z_fin[0] = zlixo;
461 
462         espessura = (zlixo - espessura);
463         volume = area * espessura;
464         printf("espessura = %f\n", espessura);
465         printf("volume    = %f\n", volume);
466 
467         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
468         x_fin[1] = xlixo;
469         y_fin[1] = ylixo;
470         z_fin[1] = zlixo;
471 
472         fscanf(in_geo, "%le %le %le", &(celula.posicao[i].fin.x), &(celula.
                posicao[i].fin.y), &(celula.posicao[i].fin.z));
473         x_fin[2] = celula.posicao[i].fin.x;
```

```
474         y_fin[2] = celula.posicao[i].fin.y;
475         z_fin[2] = celula.posicao[i].fin.z;
476 
477         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
478         x_fin[3] = xlixo;
479         y_fin[3] = ylixo;
480         z_fin[3] = zlixo;
481 
482         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
483         x_fin[4] = xlixo;
484         y_fin[4] = ylixo;
485         z_fin[4] = zlixo;
486 
487         volume_acm = volume_acm + volume;
488 
489         plotar_hash(i + 1);
490     }
491     #endif
492 
493     #if HASH == NAO
494     for (i = 0; i < nblocos; i++)
495     {
496         fscanf(in_geo, "%le %le %le", &(celula.posicao[i].ini.x), &(celula.
                posicao[i].ini.y), &(celula.posicao[i].ini.z));
497 
498         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
499 
500         printf("\nBloco %d\n", i + 1);
501         base = xlixo - celula.posicao[i].ini.x;
502         printf("base      = %f\n", base);
503         altura = celula.posicao[i].ini.y;
504 
505         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
506 
507         altura = ylixo - altura;
508         printf("altura    = %f\n", altura);
509         area = base * altura;
510         printf("area      = %f\n", area);
511         espessura = zlixo;
512 
513         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
514         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
515         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
```

```
516
517         espessura = (zlixo - espessura);
518         volume = area * espessura;
519         printf("espessura = %f\n", espessura);
520         printf("volume    = %f\n", volume);
521
522         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
523         fscanf(in_geo, "%le %le %le", &(celula.posicao[i].fin.x), &(celula.
                posicao[i].fin.y), &(celula.posicao[i].fin.z));
524         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
525         fscanf(in_geo, "%le %le %le", &xlixo, &ylixo, &zlixo);
526
527         volume_acm = volume_acm + volume;
528     }
529     #endif
530
531     printf("\nVolume Solido Acumulado = %f\n\n", volume_acm);
532
533     fclose(in_geo);
534
535     for (i = 0; i < nblocos; i++)
536         fprintf(stdout, "(%le %le %le) - (%le %le %le) \n", celula.posicao[i
                ].ini.x,
537                                                             celula.posicao[
                                                                i].ini.y,
538                                                             celula.posicao[
                                                                i].ini.z,
539                                                             celula.posicao[
                                                                i].fin.x,
540                                                             celula.posicao[
                                                                i].fin.y,
541                                                             celula.posicao[
                                                                i].fin.z);
542
543     eps = volume_acm;
544     epf = 1.0 - eps;
545
546     /* ------------------------------------ */
547     /* Propriedades do meio */
548     /* solido               */
549     difs = conds/rocs;
550     efs  = sqrt(rocs * conds);
551
```

```
552     /* fluido            */
553     diff = condf/rocf;
554     eff  = sqrt(rocf * condf);
555
556     roc = (epf * rocf) + (eps * rocs); /* roc "medio" do meio */
557     rb = (epf * rocf)/roc; /* Fator r_beta */
558
559     in_vel = openfile(nom_vel,  "r");
560
561     /* Leitura do campo de velocidades */
562     printf("\n Campo de Velocidade\n");
563
564     fscanf(in_vel, "%d", &ilixo); /* <---------- Provisorio !!!! */
565     printf("\n ilixo1 = %d \n", ilixo);
566
567     fscanf(in_vel, "%d", &ilixo);
568     printf("\n ilixo2 = %d \n", ilixo);
569
570     for (k = 0; k < malha; k++)
571     {
572        for (j = 0; j < malha; j++)
573        {
574           for (i = 0; i < malha; i++)
575           {
576              fscanf(in_vel, "%le %le %le\n", &(vel[i][j][k].x), &(vel[i][j][
                    k].y), &(vel[i][j][k].z));
577           }
578        }
579     }
580
581     fclose(in_vel);
582
583     printf("\n ---------------- \n");
584
585     /* Definicao das probabilidades de "transicoes" */
586     /* probfs : fluido ----------> solido */
587     /* probsf : solido ----------> fluido */
588     probfs = efs/(eff + efs);
589     probsf = eff/(eff + efs);
590
591     /* Calculo do passo em cada meio */
592     /* Admensionalizacao da velocidade           */
593     /* Aproveita da situacao stokesiana do fluxo  */
```

```
594     kvm = (Pe * diff)/(vm * celula.size); /* Fator de escala do campo de
            velocidades */
595
596     ni = 1.0; /* Viscosidade <----------------------------!!!! */
597     for (i = 0; i < malha; i++)
598     {
599         for (j = 0; j < malha; j++)
600         {
601             for (k = 0; k < malha; k++)
602             {
603                 vel[i][j][k].x = kvm * vel[i][j][k].x * ni * celula.size/diff;
604                 vel[i][j][k].y = kvm * vel[i][j][k].y * ni * celula.size/diff;
605                 vel[i][j][k].z = kvm * vel[i][j][k].z * ni * celula.size/diff;
606             }
607         }
608     }
609
610     vmax = matmax(vel, malha); /* Determinacao da componente de valor maximo
            */
611                                /* em modulo */
612
613     vmed = matmed(vel, malha);
614
615     rdifs = difs/diff;
616
617     /* Criterio de avaliacao do passo maximo levando em consideracao o fluxo
            */
618     /* Retirado e adaptado do artigo de Salles & alli Phys. Fluids, A 5 (10)
            , Oct 1993 2348-2376 */
619     if(Pe != 0.0) /* Teste para evitar problemas com situacoes puramente
            difusivas */
620     {
621         deltar = (2.0 / vmax) * (sqrt(1.0 + fator * vmax * d.x) - 1.0);
622
623         if(deltar < dr) dr = deltar;
624     }
625
626     dt = rdifs < 1.0 ? ((dr * dr) / (6.0)) : ((dr * dr) / (6.0 * rdifs));
627
628     dt = dt / (celula.size * celula.size * celula.size);
629
630     nt = (long int)(tmax / dt); /* Calculo do numero de passos temporais */
631
```

```
632     drf = sqrt(6.0 * dt); /* Caso Tridimensional */
633     drs = sqrt(6.0 * rdifs * dt); /* Caso Tridimensional */
634
635     /*  Transformacao do campo de velocidades em "campo" em deslocamento */
636     for (i = 0; i < malha; i++)
637     {
638        for (j = 0; j < malha; j++)
639        {
640           for (k = 0; k < malha; k++)
641           {
642              vel[i][j][k].x = dt * vel[i][j][k].x;
643              vel[i][j][k].y = dt * vel[i][j][k].y;
644              vel[i][j][k].z = dt * vel[i][j][k].z;
645           }
646        }
647     }
648
649     /* ---------------- Espelho: saida de valores ------------------------------
           */
650     fprintf(stdout, "----------------\n vmax = %le vmed = %le\n
           -------------\n", vmax, vmed);
651     fprintf(stdout, "----------------> Fator de malha = %le\n", fator);
652     fprintf(stdout, "n. de termions %ld\n Intervalos de tempo %ld\n
           intervalo entre registros %ld\n", np, nt, nimpr);
653     fprintf(stdout,  "\nGeometria da celula:\nTamanho = %5.3le\nInicio =
           (%5.3le, %5.3le, %5.3le)\nFim = (%5.3le, %5.3le, %5.3le)\n",
654                      celula.size,
655                      celula.ini.x, celula.ini.y, celula.ini.z,
656                      celula.fin.x, celula.fin.y, celula.fin.z);
657
658     /* ---------------- Espelho: saida de valores ------------------------------
           */
659     fprintf(stdout, " \nSolido :");
660     fprintf(stdout, "\n Condutividade %5.3le\n Capacidade termica %5.3le\n
           Difusividade %5.3le  \n Efusividade %5.3le",
661                     conds, rocs, difs, efs);
662     fprintf(stdout, " \nFluido:");
663     fprintf(stdout, "\n Condutividade %5.3le\n Capacidade termica %5.3le\
           nDifusividade %5.3le  \n Efusividade %5.3le",
664                     condf, rocf, diff, eff);
665     fprintf(stdout, " \n Capacidade Termica Media %5.3le\n", roc);
666     fprintf(stdout, " \n Fracao volumetrica : \nSolido %5.3le\n Fluido %5.3
           le\n",
```

```
667                     eps, epf);
668     fprintf(stdout, " \n Probabilidade de transicao : \nFluido->Solido %5.3
            le\n Solido->Fluido %5.3le\n",
669                     probfs, probsf);
670     fprintf(stdout, " \nNumero de Reynolds %5.3le\nPasso espacial inicial
            %5.3le\n Passos calculados:\nNo solido %5.3le\nNo fluido %5.3le\n\n
            Passo temporal %5.3le\n Passo temporal calculado pela velocidade %5.3
            le\n", vm, dr, drs, drf, dt, dvr);
671     fprintf(stdout, "\n Posicao da fonte de termions (%le, %le, %le)\n",
            fonte.x, fonte.y, fonte.z);
672
673     /* Alocacao de memoria com inicializacao dos "vetores" com zero, via
            calloc() */
674     npimp = nt / nimpr + 1;
675
676     printf("\n############ npimp = %ld #############\n", npimp);
677
678     tps = (double *)dalloc(DOUBLE, npimp + 1);
679
680     x   = (double *)dalloc(DOUBLE, npimp + 1);
681     x2  = (double *)dalloc(DOUBLE, npimp + 1);
682     x3  = (double *)dalloc(DOUBLE, npimp + 1);
683
684     y   = (double *)dalloc(DOUBLE, npimp + 1);
685     y2  = (double *)dalloc(DOUBLE, npimp + 1);
686     y3  = (double *)dalloc(DOUBLE, npimp + 1);
687
688     z   = (double *)dalloc(DOUBLE, npimp + 1);
689     z2  = (double *)dalloc(DOUBLE, npimp + 1);
690     z3  = (double *)dalloc(DOUBLE, npimp + 1);
691
692     xy  = (double *)dalloc(DOUBLE, npimp + 1);
693     x2y = (double *)dalloc(DOUBLE, npimp + 1);
694     xy2 = (double *)dalloc(DOUBLE, npimp + 1);
695
696     xz  = (double *)dalloc(DOUBLE, npimp + 1);
697     x2z = (double *)dalloc(DOUBLE, npimp + 1);
698     xz2 = (double *)dalloc(DOUBLE, npimp + 1);
699
700     yz  = (double *)dalloc(DOUBLE, npimp + 1);
701     y2z = (double *)dalloc(DOUBLE, npimp + 1);
702     yz2 = (double *)dalloc(DOUBLE, npimp + 1);
703
```

```
704     xyz = (double *)dalloc(DOUBLE, npimp + 1);
705
706     dif = (Tensor2 *)dalloc(TENSOR2, npimp + 1);
707
708     if (saida.pos == OK) out_pos = openfile(nom_pos, "w");
709     if (saida.pos == OK) rpos = (Vector *)dalloc(VECTOR, np);
710
711     faixa.x = 0.1;
712     faixa.y = 1.0;
713     faixa.z = 0.5;
714
715     como = UNIFORME; /* Distribuicao inicial : UNIFORME, FAIXA, PONTUAL */
716
717     for (in = 0; in < np; in++) /* Contador de particulas */
718     {
719        register double dra;
720
721        switch (como)  /* Distribuicao inicial das particulas */
722        {
723           case PONTUAL: /* Posicao inicial da fonte de termions com uma
                 pertubacao aleatoria */
724                         /* dentro de um cubo de lado dr.
                                                                */
725           {
726              pos0.x = fonte.x + 2.0 * dr * (ran2(&seed2) - 0.5);
727              pos0.y = fonte.y + 2.0 * dr * (ran2(&seed2) - 0.5);
728              pos0.z = fonte.z + 2.0 * dr * (ran2(&seed2) - 0.5);
729
730              postored2(pos0, &red0);
731           }
732           break;
733           case UNIFORME: /* Distribuicao uniforme dentro da celula elementar
                  */
734           {                 /* nos meios condutores de calor
                                  */
735              ondefonte = KO;
736
737              do
738              {
739                 pos0.x = ran2(&seed2);
740                 pos0.y = ran2(&seed2);
741                 pos0.z = ran2(&seed2);
742
```

```
743                 postored2(pos0, &red0);
744
745                 if (conds == 0.0) /* Examina o caso do solido nao ter
                        condutividade */
746                 {
747                     #if HASH == NAO
748                     if (onde_g(red0, celula, nblocos, &bloco) != SOLIDO)
                            ondefonte = OK;
749                     #endif
750
751                     #if HASH == SIM
752                     if (onde_g_hash(red0, celula, nblocos, &bloco) != SOLIDO)
                            ondefonte = OK;
753                     #endif
754                 }
755                 else  ondefonte = OK;
756             }
757             while (ondefonte == KO);
758         }
759         break;
760         case FAIXA: /* Distribuicao de termions dentro de uma faixa de
                largura faixa.x */
761                         /* e altura faixa.y nos meios condutores de calor
                                        */
762         {
763             do
764             {
765                 pos0.x = fonte.x + faixa.x * ran2(&seed2);
766                 pos0.y = fonte.y + faixa.y * ran2(&seed2);
767                 pos0.z = fonte.z + faixa.z * ran2(&seed2);
768
769                 postored2(pos0, &red0); /* Examina o caso do solido nao ter
                        condutividade */
770                 if (conds == 0.0)
771                 {
772                     #if HASH == NAO
773                     if (onde_g(red0, celula, nblocos, &bloco) != SOLIDO)
                            ondefonte = OK;
774                     #endif
775
776                     #if HASH == SIM
777                     if (onde_g_hash(red0, celula, nblocos, &bloco) != SOLIDO)
                            ondefonte = OK;
```

```
778                     #endif
779                 }
780                 else ondefonte = OK;
781             }
782             while (ondefonte == KO);
783         }
784         break;
785
786         default:
787         printf("\n Tipo de distribuicao desconhecida\n");
788         exit(NORMAL);
789       }
790
791       #if HASH == NAO
792       depart = onde_g(red0, celula, nblocos, &bloco); /* Testa em qual meio
              se encontra */
793       #endif
794
795       #if HASH == SIM
796       depart = onde_g_hash(red0, celula, nblocos, &bloco); /* Testa em qual
              meio se encontra */
797       #endif
798
799       dra = depart == FLUIDO ? drf : drs; /* Da' o deslocamento dependente
              do meio */
800
801       valor = ran1(&seed1);
802
803       if((valor >= 0.0) && (valor < um_terco))
804         dir = X;
805       else
806         if((valor >= um_terco) && (valor < dois_tercos))
807           dir = Y;
808         else
809           dir = Z;
810
811       #if DIFUSAO == NAO
812       pos.x = pos0.x;
813       pos.y = pos0.y;
814       pos.z = pos0.z;
815
816       red.x = red0.x;
817       red.y = red0.y;
```

```
818         red.z = red0.z;
819         #endif
820
821         #if DIFUSAO == SIM
822         for (it = 1; it <= nt; it++)
823         {
824             /* ---------------------- Participacao aleatoria da dispersao
                    ------------------------- */
825             sinal = (ran1(&seed1) - 0.5) >= 0.0 ? NEGATIVO : POSITIVO;  /*
                    Gera uma direcao aleatoria */
826
827             if (dir == X)
828             {
829                 pos.x = pos0.x + (double)sinal * dra; /* Calcula direcao de
                        deslocamento */
830                 pos.y = pos0.y;
831                 pos.z = pos0.z;
832             }
833             else
834             {
835                 if(dir == Y)
836                 {
837                     pos.x = pos0.x;
838                     pos.y = pos0.y + (double)sinal * dra; /* Atencao! Versao
                            feia!              */
839                     pos.z = pos0.z;
840                 }
841                 else
842                 {
843                     pos.x = pos0.x;
844                     pos.y = pos0.y;
845                     pos.z = pos0.z + (double)sinal * dra;
846                 }
847             }
848
849             postored2(pos, &red);
850
851             #if HASH == NAO
852             arrivee = onde_g(red, celula, nblocos, &bloco); /* Ve em qual meio
                    o termion chegou */
853             #endif
854
855             #if HASH == SIM
```

```
856             arrivee = onde_g_hash(red, celula, nblocos, &bloco); /* Ve em qual
                    meio o termion chegou */
857             #endif
858             /*
                    ------------------------------------------------------------------------
                    */
859
860             if (arrivee != depart) /* Se o termion mudou de meio ... */
861             {
862                 register double par, /* Parametro de interseccao da trajetoria
                        do termion com as interfaces */
863                                 sqrtdti, l1, l2;
864
865                 /* Calcula o trecho de tragetoria 'a ser percorrida no outro
                        meio */
866                 /* So' funcional para o caso de tragetorias alternadas
                                */
867                 if (dir == X)
868                 {
869                     l1 = fabs(celula.posicao[bloco].ini.x - red0.x);
870                     l2 = fabs(celula.posicao[bloco].fin.x - red0.x);
871                 }
872                 else
873                 {
874                     if(dir == Y)
875                     {
876                         l1 = fabs(celula.posicao[bloco].ini.y - red0.y);
877                         l2 = fabs(celula.posicao[bloco].fin.y - red0.y);
878                     }
879                     else
880                     {
881                         l1 = fabs(celula.posicao[bloco].ini.z - red0.z);
882                         l2 = fabs(celula.posicao[bloco].fin.z - red0.z);
883                     }
884                 }
885
886                 /*
                        ------------------------------------------------------------
                        */
887
888                 if (depart == FLUIDO) /* Se fluido -> solido ... */
889                 {
890                     par = sinal == POSITIVO ? l1 : l2;
```

```
891
892                 if (ran1(&seed1) > probfs) /* Verifica se a probabilidade
                        propicia a mudanca */
893                 {  /* Se nao, e' refletida */
894                    sqrtdti = sqrt(fabs(dt - (par * par)/4.0));
895                    arrivee = FLUIDO;
896                    dra = drf;
897
898                    if (dir == X)
899                    {
900                       pos.x = pos0.x + (double)sinal * (par - sqrtdti -
                           sqrtdti);
901                    }
902                    else
903                    {
904                       if(dir == Y)
905                       {
906                          pos.y = pos0.y + (double)sinal * (par - sqrtdti -
                              sqrtdti);
907                       }
908                       else
909                       {
910                          pos.z = pos0.z + (double)sinal * (par - sqrtdti -
                              sqrtdti);
911                       }
912                    }
913                 }
914                 else /* Se sim, modifica o ponto de chegada levando em
                        consideracao */
915                 { /* as diferencas dos meios
                                                        */
916                    sqrtdti = sqrt(rdifs * fabs(dt - (par * par)/4.0));
917                    dra = drs;
918                    arrivee = SOLIDO;
919
920                    if (dir == X)
921                    {
922                       pos.x = pos0.x + (double)sinal * (par + sqrtdti +
                           sqrtdti);
923                    }
924                    else
925                    {
926                       if(dir == Y)
```

```
927                     {
928                         pos.y = pos0.y + (double)sinal * (par + sqrtdti +
                                sqrtdti);
929                     }
930                     else
931                     {
932                         pos.z = pos0.z + (double)sinal * (par + sqrtdti +
                                sqrtdti);
933                     }
934                 }
935             }
936             else  /* Se solido -> fluido ... */
937             {
938                 par = sinal == POSITIVO ? l2 : l1;
939
940                 /* Calcula o "tempo faltante" se a particula atravessar */
941                 if (ran1(&seed1) > probsf) /* Testa se a probabilidade
                        propicia a mudanca */
942                 { /* Se nao, e' refletida */
943                     sqrtdti = sqrt(rdifs * fabs(dt - (par * par)/(4.0 * rdifs
                            )));
944                     arrivee = SOLIDO;
945                     dra     = drs;
946
947                     if (dir == X)
948                     {
949                         pos.x = pos0.x + (double)sinal * (par - sqrtdti -
                                sqrtdti);
950                     }
951                     else
952                     {
953                         if(dir == Y)
954                         {
955                             pos.y = pos0.y + (double)sinal * (par - sqrtdti -
                                    sqrtdti);
956                         }
957                         else
958                         {
959                             pos.z = pos0.z + (double)sinal * (par - sqrtdti -
                                    sqrtdti);
960                         }
961                     }
962
```

```
963                 }
964                 else /* Se sim, modifica o ponto de chegada levando em
                        consideracao */
965                 { /* as diferencas dos meios  */
966                     sqrtdti = sqrt(fabs(dt - (par * par)/(4.0 * rdifs)));
967                     dra = drf;
968                     arrivee = FLUIDO;
969 
970                     if (dir == X)
971                     {
972                         pos.x = pos0.x + (double)sinal * (par + sqrtdti +
                                sqrtdti);
973                     }
974                     else
975                     {
976                         if(dir == Y)
977                         {
978                             pos.y = pos0.y + (double)sinal * (par + sqrtdti +
                                    sqrtdti);
979                         }
980                         else
981                         {
982                             pos.z = pos0.z + (double)sinal * (par + sqrtdti +
                                    sqrtdti);
983                         }
984                     }
985                 }
986             }
987             postored2(pos, &red);
988         }
989         #endif
990 
991         #if CONVECCAO == SIM
992         /* Deslocamento provocado pelo fluxo do fluido */
993 
994         #if DIFUSAO == NAO
995         postored2(pos, &red);
996         #endif
997 
998         velocidade_p(red, vel, malha, d, &vv);
999 
1000        pos.x += vv.x;
1001        pos.y += vv.y;
```

```
1002            pos.z += vv.z;
1003
1004            postored2(pos, &redi);
1005
1006            #if HASH == NAO
1007            if (onde_g(redi, celula, nblocos, &bloco) == SOLIDO)
1008            {
1009               pos.x -= vv.x;
1010               pos.y -= vv.y;
1011               pos.z -= vv.z;
1012            }
1013            else
1014            {
1015               red.x = redi.x;
1016               red.y = redi.y;
1017               red.z = redi.z;
1018            }
1019            #endif
1020
1021            #if HASH == SIM
1022            if (onde_g_hash(redi, celula, nblocos, &bloco) == SOLIDO)
1023            {
1024               pos.x -= vv.x;
1025               pos.y -= vv.y;
1026               pos.z -= vv.z;
1027            }
1028            else
1029            {
1030               red.x = redi.x;
1031               red.y = redi.y;
1032               red.z = redi.z;
1033            }
1034            #endif
1035            #endif
1036
1037            impr = it/nimpr;
1038
1039            if (it == impr * nimpr)
1040            {
1041               x[impr]   += pos.x; /* Somatorio das posicoes em x num dado
                     momento */
1042               y[impr]   += pos.y; /* Somatorio das posicoes em y num dado
                     momento */
```

```
1043            z[impr]   += pos.z; /* Somatorio das posicoes em y num dado
                    momento */
1044
1045            x2[impr]  += (pos.x * pos.x); /* Somatorio do quadrado de x
                             */
1046            y2[impr]  += (pos.y * pos.y); /* Somatorio do quadrado de y
                             */
1047            z2[impr]  += (pos.z * pos.z); /* Somatorio do quadrado de z
                             */
1048
1049            xy[impr]  += (pos.x * pos.y); /* Somatorio do  produto xy
                               */
1050            xz[impr]  += (pos.x * pos.z); /* Somatorio do  produto xz
                               */
1051            yz[impr]  += (pos.y * pos.z); /* Somatorio do  produto yz
                               */
1052
1053            x3[impr]  += (pos.x * pos.x * pos.x); /* Somatorio do cubo de x
                           */
1054            y3[impr]  += (pos.y * pos.y * pos.y); /* Somatorio do cubo de y
                           */
1055            z3[impr]  += (pos.z * pos.z * pos.z); /* Somatorio do cubo de z
                           */
1056
1057            x2y[impr] += (pos.x * pos.x * pos.y); /* Somatorio do  produto
                    xxy    */
1058            xy2[impr] += (pos.x * pos.y * pos.y); /* Somatorio do  produto
                    xyy    */
1059
1060            x2z[impr] += (pos.x * pos.x * pos.z); /* Somatorio do  produto
                    xxz    */
1061            xz2[impr] += (pos.x * pos.z * pos.z); /* Somatorio do  produto
                    xzz    */
1062
1063            y2z[impr] += (pos.y * pos.y * pos.z); /* Somatorio do  produto
                    yyz    */
1064            yz2[impr] += (pos.y * pos.z * pos.z); /* Somatorio do  produto
                    yzz    */
1065
1066            xyz[impr] += (pos.x * pos.y * pos.z); /* Somatorio do  produto
                    xyz    */
1067        }
1068
```

```
1069            #if DIFUSAO == SIM
1070            /* Geracao alternada de direcoes */
1071            valor = ran1(&seed1);
1072
1073            if((valor >= 0.0) && (valor < um_terco))
1074               dir = X;
1075            else
1076               if((valor >= um_terco) && (valor < dois_tercos))
1077                  dir = Y;
1078               else
1079                  dir = Z;
1080
1081            pos0.x = pos.x;
1082            pos0.y = pos.y;
1083            pos0.z = pos.z;
1084
1085            red0.x = red.x;
1086            red0.y = red.y;
1087            red0.z = red.z;
1088
1089            depart = arrivee;
1090            #endif
1091         }
1092
1093         /* Imprime a posicao de cada termion */
1094         if (saida.pos == OK)
1095         {
1096            fprintf(out_pos, "\n%le, %le, %le", pos.x, pos.y, pos.z);
1097            rpos[in].x = pos.x;
1098            rpos[in].y = pos.y;
1099            rpos[in].z = pos.z;
1100         }
1101      }
1102
1103      if (saida.pos == OK) fclose(out_pos);
1104
1105      dif[0].xx = dif[0].xy = dif[0].xz = dif[0].yx = dif[0].yy = dif[0].yz =
          dif[0].zx = dif[0].zy = dif[0].zz = 0.0;
1106      tps[0] = 0.0;
1107
1108      v = (Vector *)dalloc(VECTOR, npimp + 1);
1109
1110      for (it = 1; it <= npimp; it++)
```

```
1111    {
1112        register double rnp = 1.0 / (double)np;
1113
1114        temps = dt * (double)(it * nimpr);
1115
1116        tps[it] = temps; /* Tempo adimensional */
1117
1118        xm  = x[it] * rnp;      /* Calculo do primeiro momento  */
1119        ym  = y[it] * rnp;
1120        zm  = z[it] * rnp;
1121
1122        x[it] = xm;
1123        y[it] = ym;
1124        z[it] = zm;
1125
1126        xm2 = x2[it] * rnp;     /* Calculo do segundo momento   */
1127        ym2 = y2[it] * rnp;
1128        zm2 = z2[it] * rnp;
1129
1130        xym = xy[it] * rnp;
1131        xzm = xz[it] * rnp;
1132        yzm = yz[it] * rnp;
1133
1134        xm3 = x3[it] * rnp;     /* Calculo do terceiro momento   */
1135        ym3 = y3[it] * rnp;
1136        zm3 = z3[it] * rnp;
1137
1138        xm2y = x2y[it] * rnp;
1139        xmy2 = xy2[it] * rnp;
1140
1141        xm2z = x2z[it] * rnp;
1142        xmz2 = xz2[it] * rnp;
1143
1144        ym2z = y2z[it] * rnp;
1145        ymz2 = yz2[it] * rnp;
1146
1147        x2[it] = xm2 - xm * xm; /* Calculo do segundo momento centrado */
1148        y2[it] = ym2 - ym * ym;
1149        z2[it] = zm2 - zm * zm;
1150
1151        xy[it] = xym - xm * ym; /* Covarianca */
1152        xz[it] = xzm - xm * zm;
1153        yz[it] = yzm - ym * zm;
```

```
1154
1155        x3[it]  = xm3 - 3.0 * xm2 * xm + 2.0 * xm * xm * xm;  /* Calculo do
                terceiro momento centrado */
1156        y3[it]  = ym3 - 3.0 * ym2 * ym + 2.0 * ym * ym * ym;
1157        z3[it]  = zm3 - 3.0 * zm3 * zm + 2.0 * zm * zm * zm;
1158
1159        x2y[it] = xm2y - xm2 * ym - 2.0 * xm * (xym - xm * ym);
1160        xy2[it] = xmy2 - xm * ym2 - 2.0 * ym * (xym - xm * ym);
1161
1162        x2z[it] = xm2z - xm2 * zm - 2.0 * xm * (xzm - xm * zm);
1163        xz2[it] = xmz2 - xm * zm2 - 2.0 * zm * (xzm - xm * zm);
1164
1165        y2z[it] = ym2z - ym2 * zm - 2.0 * ym * (yzm - ym * zm);
1166        yz2[it] = ymz2 - ym * zm2 - 2.0 * zm * (yzm - ym * zm);
1167
1168        /* Ponto de correcao para dar os valores como numero de Pe */
1169        /* Todos os valores foram divididos por diff               */
1170
1171        v[it].x = (1.0 / rb) * xm / temps; /* Calculo da velocidade em cada
                direcao */
1172        v[it].y = (1.0 / rb) * ym / temps;
1173        v[it].z = (1.0 / rb) * zm / temps;
1174
1175        /* Ponto de correcao para dar o tensor efetivo e nao ele sobre diff
                12/07/00 */
1176        /* Todos os valores foram multiplicados por diff
                                        */
1177
1178        dif[it].xx = diff * (roc / 2.0) * x2[it] / temps;   /* Componente xx
                de D */
1179        dif[it].yy = diff * (roc / 2.0) * y2[it] / temps;   /* Componente yy
                de D */
1180        dif[it].zz = diff * (roc / 2.0) * z2[it] / temps;   /* Componente zz
                de D */
1181
1182        dif[it].xy = diff * (roc / 2.0) * xy[it] / temps;   /* Componente xy
                de D */
1183        dif[it].yx = diff * dif[it].xy;
1184
1185        dif[it].xz = diff * (roc / 2.0) * xz[it] / temps;   /* Componente xz
                de D */
1186        dif[it].zx = diff * dif[it].xz;
1187
```

```
1188        dif[it].yz = diff * (roc / 2.0) * yz[it] / temps;    /* Componente yz
                de D */
1189        dif[it].zy = diff * dif[it].yz;
1190    }
1191
1192    if (saida.pos == OK)
1193    {
1194        vaux = (double *) dalloc(DOUBLE, np);
1195
1196        for (i = 0; i < np; i++) vaux[i] = rpos[i].x;
1197
1198        xmax = maximo(vaux, np);
1199        xmin = minimo(vaux, np);
1200
1201        for (i = 0; i < np; i++) vaux[i] = rpos[i].y;
1202
1203        ymax = maximo(vaux, np);
1204        ymin = minimo(vaux, np);
1205
1206        for (i = 0; i < np; i++) vaux[i] = rpos[i].z;
1207
1208        zmax = maximo(vaux, np);
1209        zmin = minimo(vaux, np);
1210
1211        free(vaux);
1212    }
1213
1214    fprintf(stdout, "\n regiao onde se encontram os termions (%le, %le, %le)
            (%le, %le, %le)\n Volume :%le\n", xmin, ymin, zmin, xmax, ymax, zmax
            , (xmax - xmin) * (ymax - ymin) * (zmax - zmin));
1215
1216    /* ------------------------------------ Saida de valores
            ------------------------- */
1217    printf("\n Saida de valores <================================== \n");
1218
1219    maxaj = 6;
1220
1221    ajus = (Vector *)dalloc(VECTOR, npimp);
1222
1223
1224    for (i = 1; i <= maxaj; i++)
1225    {
1226        inicio = npimp - npimp/i + 1;
```

```
1227
1228        /* ------------------------------ Momentos de ordem 1
                ------------------------------------ */
1229        k = 0;
1230        for (j = inicio; j < npimp; j++)
1231        {
1232            ajus[k].x = tps[j];
1233            ajus[k].y = (1.0/rb) * x[j];
1234            k++;
1235        }
1236        k--;
1237
1238        miniquad(ajus, k, &a0, &a1); /* Calculo dos coeficientes de ajuste
                linear */
1239        a_a = miniquad_a(ajus, k);
1240
1241        av.x = a1;
1242
1243        k = 0;
1244        for (j = inicio; j < npimp; j++)
1245        {
1246            ajus[k].x = tps[j];
1247            ajus[k].y = v[j].x;
1248            k++;
1249        }
1250        k--;
1251
1252        a_c = miniquad_c(ajus, k);
1253
1254        fprintf(stdout, "\n t = %le - v.x = %le - a0 = %le \n  Alternativo de
                momento = %le - Alternativo de tensor = %le - no. de pontos =%d",
1255        dt * inicio, a1, a0, a_a, a_c, k);
1256
1257        k = 0;
1258        for (j = inicio; j < npimp; j++)
1259        {
1260            ajus[k].x = tps[j];
1261            ajus[k].y = (1.0/rb) * y[j];
1262            k++;
1263        }
1264        k--;
1265
```

```
1266        miniquad(ajus, k, &a0, &a1); /* Calculo dos coeficientes de ajuste
                linear */
1267
1268        av.y = a1;
1269
1270        a_a = miniquad_a(ajus, k);
1271
1272        k = 0;
1273        for (j = inicio; j < npimp; j++)
1274        {
1275           ajus[k].x = tps[j];
1276           ajus[k].y = v[j].y;
1277           k++;
1278        }
1279        k--;
1280
1281        a_c = miniquad_c(ajus, k);
1282
1283        fprintf(stdout, "\n t = %le - v.y = %le - a0 = %le \n  Alternativo de
                momento = %le - Alternativo de tensor = %le - no. de pontos =%d",
1284        dt * inicio, a1, a0, a_a, a_c, k);
1285
1286        /* Inicio Acrescimo da componente Z*/
1287        k = 0;
1288        for (j = inicio; j < npimp; j++)
1289        {
1290           ajus[k].x = tps[j];
1291           ajus[k].y = (1.0/rb) * z[j];
1292           k++;
1293        }
1294        k--;
1295
1296        miniquad(ajus, k, &a0, &a1); /* Calculo dos coeficientes de ajuste
                linear */
1297
1298        av.z = a1;
1299
1300        a_a = miniquad_a(ajus, k);
1301
1302        k = 0;
1303        for (j = inicio; j < npimp; j++)
1304        {
1305           ajus[k].x = tps[j];
```

```
1306            ajus[k].y = v[j].z;
1307            k++;
1308        }
1309        k--;
1310
1311        a_c = miniquad_c(ajus, k);
1312
1313        fprintf(stdout, "\n t = %le - v.z = %le - a0 = %le \n  Alternativo de
                momento = %le - Alternativo de tensor = %le - no. de pontos =%d",
1314        dt * inicio, a1, a0, a_a, a_c, k);
1315        /* Fim Acrescimo da componente Z*/
1316
1317        fprintf(stdout, "\n >>>>>>> Resultante = %le <<<<<<<<<<\n", sqrt(av.x
                * av.x + av.y * av.y + av.z * av.z));
1318
1319        /* ------------------------------ Momentos de ordem 2
                ---------------------------------- */
1320        /* XX */
1321        k = 0;
1322        for (j = inicio; j < npimp; j++)
1323        {
1324            ajus[k].x = tps[j];
1325            ajus[k].y = diff * (roc/2.0) * x2[j];
1326            k++;
1327        }
1328        k--;
1329
1330        miniquad(ajus, k, &a0, &a1); /* Calculo dos coeficientes de ajuste
                linear */
1331        a_a = miniquad_a(ajus, k);
1332
1333        a.xx = a1;
1334
1335        k = 0;
1336        for (j = inicio; j < npimp; j++)
1337        {
1338            ajus[k].x = tps[j];
1339            ajus[k].y = dif[j].xx;
1340            k++;
1341        }
1342        k--;
1343
1344        a_c = miniquad_c(ajus, k);
```

```
1345
1346        fprintf(stdout, "\n t = %le - D.xx = %le - a0 = %le \n  Alternativo
                de momento = %le - Alternativo de tensor = %le - no. de pontos =%d
                ",
1347        dt * inicio, a1, a0, a_a, a_c, k);
1348        /* XX */
1349
1350        /* YY */
1351        k = 0;
1352        for (j = inicio; j < npimp; j++)
1353        {
1354            ajus[k].x = tps[j];
1355            ajus[k].y = diff * (roc/2.0) * y2[j];
1356            k++;
1357        }
1358        k--;
1359
1360        miniquad(ajus, k, &a0, &a1); /* Calculo dos coeficientes de ajuste
                linear */
1361        a_a = miniquad_a(ajus, k);
1362
1363        a.yy = a1;
1364
1365        k = 0;
1366        for (j = inicio; j < npimp; j++)
1367        {
1368            ajus[k].x = tps[j];
1369            ajus[k].y = dif[j].yy;
1370            k++;
1371        }
1372        k--;
1373
1374        a_c = miniquad_c(ajus, k);
1375
1376        fprintf(stdout, "\n t = %le - D.yy = %le - a0 = %le \n  Alternativo
                de momento = %le - Alternativo de tensor = %le - no. de pontos =%d
                ",
1377        dt * inicio, a1, a0, a_a, a_c, k);
1378        /* YY */
1379
1380        /* ZZ */
1381        k = 0;
1382        for (j = inicio; j < npimp; j++)
```

```
1383        {
1384            ajus[k].x = tps[j];
1385            ajus[k].y = diff * (roc/2.0) * z2[j];
1386            k++;
1387        }
1388        k--;
1389
1390        miniquad(ajus, k, &a0, &a1); /* Calculo dos coeficientes de ajuste
                linear */
1391        a_a = miniquad_a(ajus, k);
1392
1393        a.zz = a1;
1394
1395        k = 0;
1396        for (j = inicio; j < npimp; j++)
1397        {
1398            ajus[k].x = tps[j];
1399            ajus[k].y = dif[j].zz;
1400            k++;
1401        }
1402        k--;
1403
1404        a_c = miniquad_c(ajus, k);
1405
1406        fprintf(stdout, "\n t = %le - D.zz = %le - a0 = %le \n  Alternativo
                de momento = %le - Alternativo de tensor = %le - no. de pontos =%d
                ",
1407        dt * inicio, a1, a0, a_a, a_c, k);
1408        /* ZZ */
1409
1410        /* XY, YX */
1411        k = 0;
1412        for (j = inicio; j < npimp; j++)
1413        {
1414            ajus[k].x = tps[j];
1415            ajus[k].y = diff * (roc/2.0) * xy[j];
1416            k++;
1417        }
1418        k--;
1419
1420        miniquad(ajus, k, &a0, &a1); /* Calculo dos coeficientes de ajuste
                linear */
1421        a_a = miniquad_a(ajus, k);
```

```
1422
1423        a.xy = a.yx = a1;
1424
1425        k = 0;
1426        for (j = inicio; j < npimp; j++)
1427        {
1428            ajus[k].x = tps[j];
1429            ajus[k].y = dif[j].xy;
1430            k++;
1431        }
1432        k--;
1433
1434        a_c = miniquad_c(ajus, k);
1435
1436        fprintf(stdout, "\n t = %le - D.xy = %le - a0 = %le \n  Alternativo
                de momento = %le - Alternativo de tensor = %le - no. de pontos =%d
                ",
1437        dt * inicio, a1, a0, a_a, a_c, k);
1438        /* XY, YX */
1439
1440        /* XZ, ZX */
1441        k = 0;
1442        for (j = inicio; j < npimp; j++)
1443        {
1444            ajus[k].x = tps[j];
1445            ajus[k].y = diff * (roc/2.0) * xz[j];
1446            k++;
1447        }
1448        k--;
1449
1450        miniquad(ajus, k, &a0, &a1); /* Calculo dos coeficientes de ajuste
                linear */
1451        a_a = miniquad_a(ajus, k);
1452
1453        a.xz = a.zx = a1;
1454
1455        k = 0;
1456        for (j = inicio; j < npimp; j++)
1457        {
1458            ajus[k].x = tps[j];
1459            ajus[k].y = dif[j].xz;
1460            k++;
1461        }
```

```
1462        k--;
1463
1464        a_c = miniquad_c(ajus, k);
1465
1466        fprintf(stdout, "\n t = %le - D.xz = %le - a0 = %le \n  Alternativo
                de momento = %le - Alternativo de tensor = %le - no. de pontos =%d
                ",
1467        dt * inicio, a1, a0, a_a, a_c, k);
1468        /* XZ, ZX */
1469
1470        /* YZ, ZY */
1471        k = 0;
1472        for (j = inicio; j < npimp; j++)
1473        {
1474            ajus[k].x = tps[j];
1475            ajus[k].y = diff * (roc/2.0) * yz[j];
1476            k++;
1477        }
1478        k--;
1479
1480        miniquad(ajus, k, &a0, &a1); /* Calculo dos coeficientes de ajuste
                linear */
1481
1482        a_a = miniquad_a(ajus, k);
1483
1484        a.yz = a.zy = a1;
1485
1486        k = 0;
1487        for (j = inicio; j < npimp; j++)
1488        {
1489            ajus[k].x = tps[j];
1490            ajus[k].y = dif[j].yz;
1491            k++;
1492        }
1493        k--;
1494
1495        a_c = miniquad_c(ajus, k);
1496
1497        fprintf(stdout, "\n t = %le - D.yz = %le - a0 = %le \n  Alternativo
                de momento = %le - Alternativo de tensor = %le - no. de pontos =%d
                ", dt * inicio, a1, a0, a_a, a_c, k);
1498        /* YZ, ZY */
1499
```

```
1500        autovalor(a, &D, &angulo_xy, &angulo_xz, &angulo_yz);
1501
1502        fprintf(stdout, "\n Girado : >>>>>>> D.xx = %le D.yy = %le D.zz = %le
                angulo XY = %le angulo XZ = %le angulo YZ = %le <<<<<<< \n", D.xx
               , D.yy, D.zz, angulo_xy, angulo_xz, angulo_yz);
1503    }
1504
1505    /* Abertura dos arquivos de saida */
1506    fprintf(stdout, "\nPreparando as saidas dos solicitados:\n\n");
1507
1508    if (saida.vel == OK)
1509    {
1510       fprintf(stdout, "\nPrimeiro momento como velocidade\n");
1511       out_vels   = openfile(nom_vels,   "w");
1512       for (it = 1; it < npimp; it++)
1513          fprintf(out_vels, "%5.3le %5.3le %5.3le %5.3le\n",
1514                            tps[it],
1515                            v[it].x, v[it].y, v[it].z);
1516       fclose(out_vels);
1517    }
1518
1519    if (saida.disp == OK)
1520    {
1521       fprintf(stdout, "\nSegundo momento como tensor de dispersao\n");
1522       out_dif    = openfile(nom_dif,    "w");
1523       for (it = 1; it < npimp; it++)
1524          fprintf(out_dif, "%5.3le %5.3le %5.3le %5.3le %5.3le %5.3le %5.3le
                  \n",
1525                           tps[it],
1526                           dif[it].xx, dif[it].yy, dif[it].zz, dif[it].xy,
                               dif[it].xz, dif[it].yz);
1527
1528       fclose(out_dif);
1529    }
1530
1531    if (saida.mom == OK)
1532    {
1533       fprintf(stdout, "\nMomentos centrados\n");
1534       out_mom    = openfile(nom_mom,    "w");
1535
1536       /* Saida dos momentos  -> */
1537       for (it = 1; it < npimp; it++)
```

```
1538            fprintf(out_mom, "%5.3le %5.3le %5.3le %5.3le %5.3le %5.3le %5.3le
                    %5.3le %5.3le %5.3le\n",
1539                                tps[it],
1540                                x[it], y[it], z[it],
1541                                x2[it], y2[it], z2[it],
1542                                xy[it], xz[it], yz[it]);
1543
1544        fclose(out_mom);
1545    }
1546
1547    free(tps);
1548    free(x); free(y); free(z);
1549    free(x2); free(y2); free(z2);
1550    free(xy); free(xz); free(yz);
1551    free(dif);
1552
1553    return(0);
1554 }
1555
1556 int alocar_memoria()
1557 {
1558    unsigned int temp;
1559    char *caracter;
1560
1561    if(pts != pontos) pontos = pts;
1562
1563    total_mem = ((pontos * pontos * pontos) * sizeof(char));
1564    discretizacao = (pontos - 1);
1565    end_base = malloc(total_mem);
1566    temp = total_mem;
1567    caracter = end_base;
1568
1569    while(temp > 0)
1570    {
1571        *caracter = 0;
1572        caracter = caracter + tam_char;
1573        temp--;
1574    }
1575
1576    printf(" Dimensoes de cada celula (%d,%d, %d)\n", pontos, pontos, pontos
        );
1577    printf(" Neste ambiente [char] tem o tamanho de %d byte(s)\n",tam_char);
1578    printf(" Alocado %d bytes\n", total_mem);
```

```
1579
1580     return(0);
1581 }
1582
1583 int plotar_hash(int estado)
1584 {
1585     char *caracter;
1586
1587     int p0_x_ini_int = (int)(round(x_ini[0] * pontos));
1588     int p0_y_ini_int = (int)(round(y_ini[0] * pontos));
1589     int p0_z_ini_int = (int)(round(z_ini[0] * pontos));
1590     int p1_x_ini_int = (int)(round(x_ini[1] * pontos));
1591     int p1_y_ini_int = (int)(round(y_ini[1] * pontos));
1592     int p1_z_ini_int = (int)(round(z_ini[1] * pontos));
1593     int p2_x_ini_int = (int)(round(x_ini[2] * pontos));
1594     int p2_y_ini_int = (int)(round(y_ini[2] * pontos));
1595     int p2_z_ini_int = (int)(round(z_ini[2] * pontos));
1596     int p3_x_ini_int = (int)(round(x_ini[3] * pontos));
1597     int p3_y_ini_int = (int)(round(y_ini[3] * pontos));
1598     int p3_z_ini_int = (int)(round(z_ini[3] * pontos));
1599     int p4_x_ini_int = (int)(round(x_ini[4] * pontos));
1600     int p4_y_ini_int = (int)(round(y_ini[4] * pontos));
1601     int p4_z_ini_int = (int)(round(z_ini[4] * pontos));
1602
1603     int p0_x_fin_int = (int)(round(x_fin[0] * pontos));
1604     int p0_y_fin_int = (int)(round(y_fin[0] * pontos));
1605     int p0_z_fin_int = (int)(round(z_fin[0] * pontos));
1606     int p1_x_fin_int = (int)(round(x_fin[1] * pontos));
1607     int p1_y_fin_int = (int)(round(y_fin[1] * pontos));
1608     int p1_z_fin_int = (int)(round(z_fin[1] * pontos));
1609     int p2_x_fin_int = (int)(round(x_fin[2] * pontos));
1610     int p2_y_fin_int = (int)(round(y_fin[2] * pontos));
1611     int p2_z_fin_int = (int)(round(z_fin[2] * pontos));
1612     int p3_x_fin_int = (int)(round(x_fin[3] * pontos));
1613     int p3_y_fin_int = (int)(round(y_fin[3] * pontos));
1614     int p3_z_fin_int = (int)(round(z_fin[3] * pontos));
1615     int p4_x_fin_int = (int)(round(x_fin[4] * pontos));
1616     int p4_y_fin_int = (int)(round(y_fin[4] * pontos));
1617     int p4_z_fin_int = (int)(round(z_fin[4] * pontos));
1618
1619     int tam_z = (p0_z_fin_int - p0_z_ini_int);
1620     int tam_y = (p2_y_ini_int - p1_y_ini_int);
1621     int tam_x = (p1_x_ini_int - p0_x_ini_int);
```

```
1622
1623    int x, y, z;
1624
1625    for(x = 0; x < tam_x; x++)
1626    {
1627        for(y = 0; y < tam_y; y++)
1628        {
1629            for(z = 0; z < tam_z; z++)
1630            {
1631                caracter = end_base + ((p0_z_ini_int + z) * pontos * pontos) +
1632                                      ((p0_y_ini_int + y) * pontos) +
1633                                      ((p0_x_ini_int + x));
1634                *caracter = estado;
1635            }
1636        }
1637    }
1638    return(0);
1639 }
1640
1641 Fase onde_g_hash(Vector red, Cel celula, int nblocos, int *bloco)
1642 {
1643    unsigned int x_inteiro, y_inteiro, z_inteiro;
1644
1645    Bloco b;
1646
1647    Fase f = FLUIDO;
1648
1649    char *caracter;
1650
1651    x_inteiro = (int)(red.x * discretizacao);
1652    y_inteiro = (int)(red.y * discretizacao);
1653    z_inteiro = (int)(red.z * discretizacao);
1654
1655    caracter = end_base + x_inteiro + (y_inteiro * pontos) + (z_inteiro *
         pontos * pontos);
1656
1657    if(*caracter != 0)
1658    {
1659        b = celula.posicao[*caracter - 1];
1660
1661        if (red.z >= b.ini.z)
1662        {
1663            if (red.z <= b.fin.z)
```

```
1664            {
1665                if (red.y >= b.ini.y)
1666                {
1667                    if (red.y <= b.fin.y)
1668                    {
1669                        if (red.x >= b.ini.x)
1670                        {
1671                            if (red.x <= b.fin.x)
1672                            {
1673                                f = SOLIDO;
1674                                *bloco = *caracter - 1;
1675                            }
1676                        }
1677                    }
1678                }
1679            }
1680        }
1681    }
1682    return (f);
1683 }
1684
1685 Fase onde_g(Vector red, Cel celula, int nblocos, int *bloco)
1686 {
1687    register int i;
1688    Fase f = FLUIDO;
1689    Bloco b;
1690
1691    /* Dada a posicao pos e os parametros da celula, determina onde   */
1692    /* pos esta' (solido ou liquido) e se estiver num solido devolve  */
1693    /* qual o bloco onde se encontrava (necessario para o calculo de  */
1694    /* tragetoria).                                                   */
1695    /* Celula com origem no canto inferior esquerdo                   */
1696    for (i = 0; i < nblocos; i++)
1697    {
1698        b = celula.posicao[i];
1699
1700        if(red.z >= b.ini.z)
1701        {
1702            if(red.z <= b.fin.z)
1703            {
1704                if(red.y >= b.ini.y)
1705                {
1706                    if(red.y <= b.fin.y)
```

```
1707                {
1708                    if(red.x >= b.ini.x)
1709                    {
1710                        if(red.x <= b.fin.x)
1711                        {
1712                            f = SOLIDO;
1713                            *bloco = i;
1714                        }
1715                    }
1716                }
1717            }
1718        }
1719      }
1720      if (f == SOLIDO) break;
1721    }
1722
1723    return f;
1724 }
1725
1726 void postored2 (Vector pos, Vector *red)
1727 {
1728   /* Calcula as coordenadas reduzidas, ou seja, correspondentes 'a celula
        */
1729   /* Supoem-se que a celula esta' contida num quadrado de canto inferior
        */
1730   /* esquerdo na origem e canto superior direito em (1, 1).
        */
1731    red->x = pos.x - (double) ((long int) pos.x);
1732
1733    if (pos.x <= 0.0)
1734    {
1735       if (red->x != 0.0) red->x += 1.0;
1736    }
1737    else
1738    {
1739       if (red->x == 0.0) red->x = 1.0;
1740    }
1741
1742    red->y = pos.y - (double) ((long int) pos.y);
1743
1744    if (pos.y <= 0.0)
1745    {
1746       if (red->y != 0.0) red->y += 1.0;
```

```
1747     }
1748     else
1749     {
1750         if (red->y == 0.0) red->y = 1.0;
1751     }
1752
1753     red->z = pos.z - (double) ((long int) pos.z);
1754
1755     if (pos.z <= 0.0)
1756     {
1757         if (red->z != 0.0) red->z += 1.0;
1758     }
1759     else
1760     {
1761         if (red->z == 0.0) red->z = 1.0;
1762     }
1763 }
1764
1765 void velocidade_p(Vector p, Vector v[][N][N], int n, Vector d, Vector *vv)
1766 {
1767     /* Devolve a velocidade do fluido num determinado ponto especificado por
          p */
1768     /* Usa a velocidade do "centro" do volume como velocidade para qualquer
           */
1769     /* termion que esteja em qualquer ponto do volume
                                    */
1770     register int i, j, k;
1771
1772     i = (int)(floor(p.x / d.x));
1773     j = (int)(floor(p.y / d.y));
1774     k = (int)(floor(p.z / d.z));
1775
1776     if (i == n) --i;
1777     if (j == n) --j;
1778     if (k == n) --k;
1779
1780     vv->x = v[i][j][k].x;
1781     vv->y = v[i][j][k].y;
1782     vv->z = v[i][j][k].z;
1783 }
1784
1785 double ran1(long int *i)
1786 {
```

```
1787    /* Funcao geradora de numeros pseudo-aleatorios */
1788    /* Retirada no livro Numerical Recipes for C - 2a. edicao */
1789    register int j;
1790    register long int k;
1791    long int NDIV;
1792
1793    static long int y = 0;
1794    static long int v[NTAB];
1795
1796    double temp;
1797
1798    if ((*i <= 0) || (y == 0))
1799    {
1800        if (-(*i) < 1) *i = 1;
1801        else           *i = -(*i);
1802
1803        for (j = NTAB + 7; j >=0; j--)
1804        {
1805            k  = (*i) / Q11;
1806            *i = A11 * (*i - k * Q11) - R11 * k;
1807
1808            if (*i < 0) *i = (*i) + M11;
1809
1810            if (j < NTAB) v[j] = *i;
1811        }
1812        y = v[0];
1813    }
1814    k = (*i) / Q11;
1815
1816    *i = A11 * (*i - k * Q11) - R11 * k;
1817
1818    if (*i < 0) *i = (*i) + M11;
1819
1820    NDIV = 1 + (M11 - 1) / NTAB;
1821
1822    j = y / NDIV;
1823
1824    y = v[j];
1825
1826    v[j] = *i;
1827
1828    temp = ((double) y) / ((double) M11);
1829
```

```
1830     if (temp > RNMX) return RNMX;
1831     else             return temp;
1832 }
1833
1834 double ran2(long int *i)
1835 {
1836     /* Funcao geradora de numeros pseudo-aleatorios */
1837     /* Retirada no livro Numerical Recipes for C - 2a. edicao */
1838     register int j;
1839     register long int k;
1840     long int NDIV;
1841
1842     static long i2 = 123456789;
1843     static long int y = 0;
1844     static long int v[NTAB];
1845
1846     double temp;
1847
1848     if ((*i <= 0) || (y == 0))
1849     {
1850         if (-(*i) < 1) *i = 1;
1851         else           *i = -(*i);
1852
1853         for (j = NTAB + 7; j >= 0; j--)
1854         {
1855             k  = (*i) / Q1;
1856             *i = A1 * (*i - k * Q1) - R1 * k;
1857
1858             if (*i < 0) *i = (*i) + M1;
1859
1860             if (j < NTAB) v[j] = *i;
1861         }
1862         y = v[0];
1863     }
1864     k = (*i) / Q1;
1865
1866     *i = A1 * (*i - k * Q1) - R1 * k;
1867
1868     if (*i < 0) *i = (*i) + M1;
1869
1870     k = i2 / Q2;
1871
1872     i2 = A2 * (i2 - k * Q2) - k * R2;
```

```
1873
1874    if(i2 < 0) i2 += M2;
1875
1876    NDIV = 1 + MM1 / NTAB;
1877
1878    j = y / NDIV;
1879
1880    y = v[j] - i2;
1881
1882    v[j] = *i;
1883
1884    if(y < 1) y += MM1;
1885
1886    temp = ((double) y) / ((double) M1);
1887
1888    if (temp > RNMX) return RNMX;
1889    else             return temp;
1890 }
1891
1892 void miniquad(Vector v[], long int n, double *a0, double *a1)
1893 {
1894    /* Faz ajuste por minimos quadrados de um vetor dado
1895       por um polinomio do primeiro grau */
1896    register long int i;
1897    double sx  = 0.0,
1898           sy  = 0.0,
1899           sx2 = 0.0,
1900           sxy = 0.0,
1901           det;
1902
1903    for (i = 0; i < n; i++)
1904    {
1905       sx  += v[i].x;
1906       sy  += v[i].y;
1907       sxy += v[i].x * v[i].y;
1908       sx2 += v[i].x * v[i].x;
1909    }
1910
1911    det = (double)(n + 1) * sx2 - sx * sx;
1912
1913    if (fabs(det) < ZERO) puts("\n miniquad: determinante nulo");
1914
1915    *a0 = (sy * sx2 - sx * sxy) / det;
```

```
1916     *a1 = ((double) (n + 1) * sxy - sx * sy) / det;
1917 }
1918
1919 double miniquad_a(Vector v[], long int n)
1920 {
1921     /* Faz ajuste por minimos quadrados pela funcao y = ax */
1922     register long int i;
1923     double sx2 = 0.0,
1924            sxy = 0.0;
1925
1926     for (i = 0; i < n; i++)
1927     {
1928        sxy += v[i].x * v[i].y;
1929        sx2 += v[i].x * v[i].x;
1930     }
1931
1932     if (fabs(sx2) < ZERO) puts("\n miniquad_a: Valores de x nulos!");
1933
1934     return(sxy / sx2);
1935 }
1936
1937 double miniquad_c(Vector v[], long int n)
1938 {
1939     /* Ajusta por minimos quadrados uma funcao do tipo y = a */
1940     register long int i;
1941     double sy  = 0.0;
1942
1943     for (i = 0; i < n; i++) sy  += v[i].y;
1944
1945     if ( n  < ZERO) puts("\n miniquad_c: numero de pontos negativo!");
1946
1947     return(sy / (double)n);
1948 }
1949
1950 void autovalor(Tensor2 t, Tensor2 *k, double *ang_xy, double *ang_xz,
        double *ang_yz)
1951 {
1952     /* Esta funcao retorna os autovalores de uma matriz simetrica, isto e
1953        apenas valores reais, pelo metodo de tartaglia aplicado no polinomio
1954        caracteristico de uma martiz 3x3.
1955        Devolve tambem os angulos de rotacao necessarios para a
1956        diagonalizacao em graus */
1957
```

```
1958    double m[3][3], a, b, c, d, x1, x2, x3,
1959           M, r, f,
1960           u, v,
1961           u3,
1962           v3,
1963           Delta,
1964           Real,
1965           L,
1966           Imag;
1967
1968    m[0][0] = t.xx; m[0][1] = t.xy; m[0][2] = t.xz;
1969    m[1][0] = t.yx; m[1][1] = t.yy; m[1][2] = t.yz;
1970    m[2][0] = t.zx; m[2][1] = t.zy; m[2][2] = t.zz;
1971
1972    a = -1.0;
1973
1974    b =  m[0][0] + m[1][1] + m[2][2];
1975
1976    c = -m[0][0] * m[1][1] -
1977         m[0][0] * m[2][2] -
1978         m[1][1] * m[2][2] +
1979         m[0][1] * m[1][0] +
1980         m[1][2] * m[2][1] +
1981         m[0][2] * m[2][0];
1982
1983    d =  m[0][0] * m[1][1] * m[2][2] -
1984         m[0][1] * m[1][0] * m[2][2] -
1985         m[1][2] * m[2][1] * m[0][0] -
1986         m[0][2] * m[2][0] * m[1][1] +
1987         m[0][1] * m[1][2] * m[2][0] +
1988         m[0][2] * m[1][0] * m[2][1];
1989
1990    double A = b / a,
1991           B = c / a,
1992           C = d / a,
1993           p = B - A * A / 3.0,
1994           q = C - A * B / 3.0 + 2.0 * A * A * A / 27.0,
1995           D = q * q / 4.0 + p * p * p / 27.0;
1996
1997    if(d == 0.0)
1998    {
1999        x1 = 0.0;
2000        x2 = (-b + sqrt(b * b - 4.0 * a * c)) / 2.0 * a;
```

```
2001        x3 = (-b - sqrt(b * b - 4.0 * a * c)) / 2.0 * a;
2002    }
2003    else
2004    {
2005        if(D < 0.0)
2006        {
2007            M  = sqrt(-D),
2008            r  = sqrt(q * q / 4.0 + M * M),
2009            f  = acos(-q / 2.0 / r),
2010
2011            x1 = 2.0 * pow(r , 1.0 / 3.0) * cos(f / 3.0) - A / 3.0,
2012            x2 = 2.0 * pow(r , 1.0 / 3.0) * cos((f + 2.0 * M_PI) / 3.0) - A /
                    3.0,
2013            x3 = 2.0 * pow(r , 1.0 / 3.0) * cos((f + 4.0 * M_PI) / 3.0) - A /
                    3.0;
2014        }
2015        else
2016        {
2017            u3 = -q / 2.0 + sqrt(D);
2018
2019            if(u3 < 0.0) u = -pow(-u3 , 1.0 / 3.0);
2020            else u = pow(u3 , 1.0 / 3.0);
2021
2022            v3 = -q / 2.0 - sqrt(D);
2023
2024            if(v3 < 0.0) v = -pow(-v3 , 1.0 / 3.0);
2025            else v = pow(v3 , 1.0 / 3.0);
2026
2027            x1 = u + v - A / 3.0;
2028
2029            Delta =  (A + x1) * (A + x1) + 4.0 * C / x1;
2030
2031            Real  = -(A + x1) / 2.0;
2032
2033            L     = fabs(Delta);
2034
2035            Imag = sqrt(L) / 2.0;
2036
2037            x2 = Real + Imag,
2038            x3 = Real - Imag;
2039        }
2040    }
2041
```

```
2042     k->xx = x1;
2043     k->yy = x2;
2044     k->zz = x3;
2045 
2046     k->xy = k->yx = k->xz = k->zx = k->yz = k->zy = 0.0;
2047 
2048     if ((t.xx - t.yy) == 0.0)
2049        *ang_xy = 90.0;
2050     else
2051        *ang_xy = (180.0/PI) * 0.5 * atan((2.0 * t.xy)/(t.xx - t.yy));
2052 
2053     if ((t.xx - t.zz) == 0.0)
2054        *ang_xz = 90.0;
2055     else
2056        *ang_xz = (180.0/PI) * 0.5 * atan((2.0 * t.xz)/(t.xx - t.zz));
2057 
2058     if ((t.yy - t.zz) == 0.0)
2059        *ang_yz = 90.0;
2060     else
2061        *ang_yz = (180.0/PI) * 0.5 * atan((2.0 * t.yz)/(t.yy - t.zz));
2062 }
2063 
2064 double maximo(double v[], int n)
2065 {
2066     register int i;
2067     register double vmax = v[0];
2068 
2069     for (i = 0; i < n; i++)
2070     {
2071        if (v[i] > vmax) vmax = v[i];
2072     }
2073     return vmax;
2074 }
2075 
2076 double minimo(double v[], int n)
2077 {
2078     register int i;
2079     register double vmin = v[0];
2080 
2081     for (i = 0; i < n; i++)
2082     {
2083        if (v[i] < vmin) vmin = v[i];
2084     }
```

```
2085     return vmin;
2086 }
2087
2088 double matmax(Vector v[][N][N], int n)
2089 {
2090     /* Devolve o maior elemento em modulo de uma matriz de Vector */
2091     register int i, j, k;
2092     register double vmaxx = fabs(v[0][0][0].x),
2093                     vmaxy = fabs(v[0][0][0].y),
2094                     vmaxz = fabs(v[0][0][0].z);
2095
2096     for (i = 0; i < n; i++)
2097         for (j = 0; j < n; j++)
2098             for (k = 0; k < n; k++)
2099                 if (fabs(v[i][j][k].x) > vmaxx) vmaxx = fabs(v[i][j][k].x);
2100
2101     for (i = 0; i < n; i++)
2102         for (j = 0; j < n; j++)
2103             for (k = 0; k < n; k++)
2104                 if (fabs(v[i][j][k].y) > vmaxy) vmaxy = fabs(v[i][j][k].y);
2105
2106     for (i = 0; i < n; i++)
2107         for (j = 0; j < n; j++)
2108             for (k = 0; k < n; k++)
2109                 if (fabs(v[i][j][k].z) > vmaxz) vmaxz = fabs(v[i][j][k].z);
2110
2111     if((vmaxx > vmaxy) && (vmaxx > vmaxz)) return(vmaxx);
2112     if((vmaxy > vmaxx) && (vmaxy > vmaxz)) return(vmaxy);
2113     if((vmaxz > vmaxx) && (vmaxz > vmaxy)) return(vmaxz);
2114 }
2115
2116 double matmed(Vector v[][N][N], int n)
2117 {
2118     /* Devolve o maior elemento em modulo de uma matriz de Vector */
2119     register int i, j, k;
2120     register double soma = 0.0;
2121
2122     for (i = 0; i < n; i++)
2123         for (j = 0; j < n; j++)
2124             for (k = 0; k < n; k++)
2125                 soma += fabs(v[i][j][k].x);
2126
2127     for (i = 0; i < n; i++)
```

```
2128        for (j = 0; j < n; j++)
2129            for (k = 0; k < n; k++)
2130                soma += fabs(v[i][j][k].y);
2131
2132    for (i = 0; i < n; i++)
2133        for (j = 0; j < n; j++)
2134            for (k = 0; k < n; k++)
2135                soma += fabs(v[i][j][k].z);
2136
2137    return (soma / (double) (n * n * n));
2138 }
2139
2140 void problemas(Direcao dir, Fase d, Fase a,
2141                Vector red0, Vector red, Vector pos0, Vector pos, double par
                       , double tempo)
2142 {
2143    fprintf(stdout, "\n-> Dentro: par = %5.3le\n # tempo = %5.3le", par,
           tempo);
2144    fprintf(stdout, "\n reduzidas ->(%le, %le, %le) - (%le, %le, %le)", red0
           .x, red0.y, red0.z, red.x, red.y, red.z);
2145    fprintf(stdout, "\n originais ->(%le, %le, %le) - (%le, %le, %le)\n",
           pos0.x, pos0.y, pos0.z, pos.x, pos.y, pos.z);
2146    if(dir == X) fprintf(stdout, "\n Direcao do movimento %c", 'X');
2147    else fprintf(stdout, "\n Direcao do movimento %c", dir == Y ? 'Y' : 'Z')
           ;
2148    fprintf(stdout, "\n %s -> %s", d == FLUIDO ? "FLUIDO" : "SOLIDO", a ==
           FLUIDO ? "FLUIDO" : "SOLIDO");
2149    fprintf(stdout, "\n Diferencas %le %le %le %le, %le, %le\n", red0.x -
           0.1, red0.x - 0.9, red.y - 0.1, red.y - 0.9, red.z - 0.1, red.z -
           0.9);
2150 }
2151
2152 int acha(char o_que, char *buff, int inicio)
2153 {
2154    int i = inicio;
2155
2156    while(buff[i] != o_que) i++;
2157
2158    i++;
2159    return i;
2160 }
2161
2162 void transf(char *saida, char *buff, int inicio, int fim)
```

```
2163 {
2164     int i,
2165         j = 0;
2166 
2167     for (i = inicio; i < fim; i++)
2168     {
2169         saida[j] = buff[i];
2170         j++;
2171     }
2172 
2173     saida[j] = '\0';
2174 }
2175 
2176 void *dalloc(Dado d, long int n)
2177 {
2178     /* Alocador geral de memoria                    */
2179     /* Aloca espaco para vetores de varios formatos */
2180     long int sz;
2181     void *p;
2182 
2183     switch (d)
2184     {
2185         case CHAR:
2186             sz = sizeof(char);
2187             break;
2188         case INT:
2189             sz = sizeof(int);
2190             break;
2191         case FLOAT:
2192             sz = sizeof(float);
2193             break;
2194         case DOUBLE:
2195             sz = sizeof(double);
2196             break;
2197         case VECTOR:
2198             sz = sizeof(Vector);
2199             break;
2200         case TENSOR2:
2201             sz = sizeof(Tensor2);
2202             break;
2203         default:
2204             puts("\n dalloc: Tipo de dados invalido\n");
2205             return NULL;
```

```
2206     }
2207
2208     if ((p = calloc(n, sz)) == NULL)
2209     {
2210         puts("\n dalloc: Nao ha' mais espaco em memoria \n");
2211         exit(NORMAL);
2212     }
2213     return(p);
2214 }
2215
2216 void limpanom(char nom[])
2217 {
2218     /* Dado um nome de arquivo, elimina a extensao (se houver) */
2219     while ((*nom != '.') && (*nom)) nom++;
2220
2221     *nom = '\0';
2222 }
2223
2224 FILE *openfile(char nom[], char mode[])
2225 {
2226     /* Devolve um ponteiro para um arquivo dado num determinado modo     */
2227     /* com tratamento de problemas de abertura. ATENCAO: versao limitada */
2228     FILE *f;
2229
2230     if ((f = fopen(nom, mode)) == NULL)
2231     {
2232         puts("Nao foi possivel abrir o arquivo :");
2233         puts(nom);
2234         exit(NORMAL);
2235     }
2236     return f;
2237 }
```

Referências

[1] SILVEIRA, O. T. da. *Dispersão Térmica em Meios Porosos Periódicos. Um Estudo Numérico.* [S.l.]: Intituto Politécnico do Rio de Janeiro (IPRJ / UERJ), 2001.

[2] BEAR, J. *Dynamics of Fluids in Porous Media.* [S.l.]: Dover, 1988.

[3] KAVIANY, M. *Principles of Heat Transfer in Porous Media.* [S.l.]: Springer-Verlag, 1992.

[4] BRENNER, H. Dispersion resulting from flow through spatial periodic porous media. *Phil. Trans. Roy. Soc.*, v. 297, p. 81–133, 1980.

[5] SANCHEZ-PALENCIA, E. *Non-Homogenious Media and Vibration Theory.* [S.l.]: Lectures Notes in Physics, 1980.

[6] PAULI, W. *Statical Mechanics.* [S.l.]: The MIT Press, 1977.

[7] SOMMERFELD, A. *Thermodinamics and Statical Mechanics.* [S.l.]: Academic Press, 1967.

[8] STACKEL, J. *Eistein´s Miraculous Year.* [S.l.]: Princeton University Press, 1988.

[9] TAYLOR, G. Dispersion of solube matter in solvent flowing slowly through a tube. *Proc. Roy. Soc.*, v. 219, p. 186–203, 1953.

[10] SOUTO, H. P. A. Introdução à técnica da média volumétrica. *I Escola em Modelagem Computacional Multiescala*, v. 1, p. 7–9, 2005.

[11] CARBONELL, R. G.; WHITAKER, S. *Heat and Mass Transfer in Porous Media. Fundamentals of Transport Phenomena in Porous Media.* [S.l.]: Martinus Nijhoff Publishers, 1984.

[12] QUINTARD, M.; WHITAKER, S. Transport in ordered and disordered porous media 1 : The celular average and the use of weighting functions. *Transport in Porous Media*, v. 14, p. 163–177, 1994.

[13] QUINTARD, M.; WHITAKER, S. Transport in ordered and disordered porous media 2 : Generalized volume averaging and the use of weighting functions. *Transport in Porous Media*, v. 14, p. 179–206, 1994.

[14] QUINTARD, M.; WHITAKER, S. Transport in ordered and disordered porous media 3 : Closure and comparison between theory and experiment. *Transport in Porous Media*, v. 15, p. 31–49, 1994.

[15] QUINTARD, M.; WHITAKER, S. Transport in ordered and disordered porous media 4 : Computer generated porous media for three-dimensional systems. *Transport in Porous Media*, v. 15, p. 51–70, 1994.

[16] QUINTARD, M.; WHITAKER, S. Transport in ordered and disordered porous media 5 : Geometrical results for two-dimensional systems. *Transport in Porous Media*, v. 15, p. 183–196, 1994.

[17] MARLE, C. M. On macroscopic equations governing multiphase flow with diffusion and chemical reactions in porous media. *Int. J. Engng. Sci*, v. 50, p. 643–662, 1982.

[18] SPIEGEL., M. R. *Probabilidade e Estatística. Coleção Schaum.* [S.l.]: Mc Graw Hill, 1978.

[19] SPIEGEL., M. R. *Análise Vetorial. Coleção Schaum.* [S.l.]: Mc Graw Hill, 1974.

[20] ANDERSON, D. A.; TANNEHILL, J. C.; PLETCHER, R. H. *Computational Fluid Mechanics and Heat Transfer.* [S.l.]: Hemisphere Publishing Corporation, 1984.

[21] WOLFRAN, S. *Cellular Automata and Complexity.* [S.l.]: Westview Press, 2002.

[22] THOVERT, J. S.-F. et al. Taylor dispersion in porous media. determination of the dispersion tensor. *Physics of Fluids*, v. 2348-2376, 1993.

[23] CORMEM, T. H.; LEISERSON, C. E.; RIVEST, R. L. *Introdução à Algorítimos.* [S.l.]: Campus Editora, 2001.

[24] SZWARCFITER, J. L.; MARKENZON, L. *Estrutura de Dados e seus Algorítimos.* [S.l.]: LTC, 1994.

[25] TANENBAUM, A. S. *Sistemas de Computação.* [S.l.]: Campus Editora, 1997.

[26] MOYNE, C. et al. Thermal dispersion in porous medium: One-equation model. *International Journal of Heat and Mass Transfer*, v. 43:3853-3867, 2000.

[27] ZAMITH ESTEBAN WALTER GONZALES CLUA, A. C. Marcelo Panaro de M.; MONTENEGRO, A. Parallel processing between gpu and cpu: Concepts in a game architecture. *Computer Graphics Imaging and Visualisation CGIV*, v. 1, 2007.

[28] ANDERSONA, W. A. G. I. A. G.; SCHRöDERB, P. Quantum monte carlo on graphical processing units. *Computer Physics Communications*, v. 177, 2007.