

Métodos Computacionais da Física

Versão SCILAB



Claudio Scherer

2^a Edição

Livro
de
Física



Métodos Computacionais da Física

Livraria
da

Física



Editora

Métodos Computacionais da Física

Claudio Scherer

2^a Edição

Livraria
da
Física



Editora

Copyright © 2010 Editora Livraria da Física
2^a Edição

Direção Editorial
José Roberto Marinho

Capa
artquatum

Diagramação
Claudio Scherer

Dados Internacionais de Catalogação na Publicação (CIP)

(Câmara Brasileira do Livro, SP, Brasil)

Scherer, Cláudio
Métodos computacionais da física / Cláudio
Scherer. -- 2. ed. -- São Paulo : Editora
Livraria da Física, 2010.
Bibliografia.

ISBN 978-85-7861-062-3

1. Física - Métodos computacionais I. Título.

10-01519

CDD-530.0285

Índices para catálogo sistemático:

1. Física : Métodos computacionais 530.0285



Sumário

PREFÁCIO	v
1 INTRODUÇÃO AOS MÉTODOS NUMÉRICOS	1
1.1 Raízes de Funções	1
1.1.1 Iteração	2
1.1.2 Funções Escalares de uma Variável	3
1.1.3 Funções Vetoriais de Variável Vetorial	9
1.1.4 Método de Newton Generalizado	16
1.2 Aproximações Numéricas de Funções	16
1.2.1 Interpolação Polinomial	17
1.2.2 Interpolação por Segmentos	20
1.2.3 Ajuste	25
1.3 Integração Numérica	34
1.3.1 Regra do Trapézio	34
1.3.2 Outras Aproximações para Integração Numérica	36
1.3.3 Integrais Impróprias	37
1.4 Transformada de Fourier	39
1.4.1 Propriedades das Transformadas de Fourier . .	40
1.4.2 A Delta de Dirac	43
1.4.3 Transformada de Fourier Discreta	44
1.4.4 Transformada de Fourier Rápida	50
2 EQUAÇÕES DIFERENCIAIS ORDINÁRIAS	61
2.1 Equações de Primeira Ordem	61
2.1.1 Derivada Numérica	61
2.1.2 Erro na Derivação Numérica	64

2.1.3	Sistema de Equações	65
2.2	Equações de Segunda Ordem	67
2.2.1	Algoritmo de Verlet	68
2.2.2	O Algoritmo “Leap-Frog” (Pulo de Rã)	73
2.2.3	O Algoritmo “Velocity-Verlet”	74
2.2.4	Métodos Runge-Kutta	75
2.2.5	Monitoração da Precisão pela Energia	80
2.2.6	Incrementos Adaptativos	80
3	EQUAÇÕES DIFERENCIAIS PARCIAIS	87
3.1	Equações Uni-Dimensionais no Espaço	87
3.1.1	Equação da Difusão	88
3.1.2	Equação de Deriva	91
3.1.3	Defeitos por Radiação	92
3.1.4	Solução Estacionária	95
3.1.5	Procedimentos “Explícito”, “Implícito” e “Crank-Nicholson”	97
3.1.6	Equação de Schrödinger	105
3.2	Relaxação em Duas ou Mais Dimensões	110
3.2.1	Algoritmo de Jacobi em Duas Dimensões Espaciais	111
3.2.2	Algoritmo de Jacobi em Três Dimensões Espaciais	113
3.2.3	Algoritmo de Gauss-Seidel	115
3.2.4	Algoritmo “Super-relaxação” (Overrelaxation) .	116
4	PROBABILIDADE, . . .	119
4.1	Introdução à Teoria de Probabilidade	119
4.1.1	O Conceito de Probabilidade	119
4.1.2	Probabilidade Condicional	121
4.1.3	Eventos Independentes	122
4.2	Variável Aleatória	122
4.2.1	Média	122
4.2.2	Momentos, Variância e Covariância	123
4.2.3	Variável Aleatória Contínua	124
4.2.4	Distribuições Especiais	126
4.2.5	Geração de Números Aleatórios para Outras Distribuições	129

4.3	Processos Estocásticos (PE)	135
4.3.1	Processos Estocásticos Contínuos	139
4.3.2	PE Discretos: Equação Master	141
5	DINÂMICA MOLECULAR	145
5.1	Um Programa de Dinâmica Molecular	146
5.2	Um Modelo Simples	147
5.3	Abordagem Nível 0	148
5.3.1	O Programa Principal: dinamol0.sce	148
5.3.2	Inicialização	150
5.3.3	Cálculo das Forças	151
5.3.4	Integração das Equações de Movimento	154
5.3.5	Cálculo das Variáveis Dinâmicas Relevantes .	156
5.4	Abordagem Nível 1	157
5.4.1	O Programa Principal: dinamol1.sce	158
5.4.2	Células e Listas	159
5.4.3	Inicialização	160
5.4.4	Identificação das Células Vizinhas	161
5.4.5	Listas dos Átomos Presentes em Cada Célula	164
5.4.6	Cálculo das Forças	165
5.4.7	Integração das Equações de Movimento	167
5.4.8	Cálculo das Variáveis Dinâmicas Relevantes .	168
5.5	Condições de Contorno Periódicas	170
6	MÉTODO MONTE CARLO	181
6.1	Elementos de Mecânica Estatística	181
6.1.1	O Objeto da Mecânica Estatística	182
6.1.2	Conceitos Fundamentais da Mecânica Estatística	183
6.1.3	Porque a Mecânica Estatística Funciona . . .	186
6.1.4	A Distribuição de Boltzmann	187
6.1.5	Energia Interna e Calor Específico	188
6.1.6	O Princípio do Balanço Detalhado	188
6.2	Média de Ensemble por Amostragem	190
6.2.1	Média por Amostragem Tendenciosa	191
6.3	O Algoritmo de Metropolis	191
6.3.1	Primeiro Exemplo: O Modelo de Ising	196

6.3.2 Segundo Exemplo: O Gás Bi-Dimensional	204
6.3.3 O Método do Histograma	216
7 DINÂMICA ESTOCÁSTICA	219
7.1 Ruído Branco e Processo de Wiener	219
7.2 Derivada de PE e Equação de Langevin	222
7.3 EDE com Ruído Aditivo	224
7.4 EDE com Ruído Multiplicativo	228
7.4.1 Cálculo de Ito	229
7.4.2 Cálculo de Stratonovich	240
7.5 Cálculo Estocástico com Vários Ruídos	244
7.5.1 Sobre as Abordagens de Ito e de Stratonovich em Simulações Numéricas	246
7.5.2 Solução da Equação de Langevin com Ruído multiplicativo até $O(\Delta t^{\frac{5}{2}})$	249
7.5.3 Movimento Browniano em Coordenadas Polares	255
7.6 A Equação de Fokker-Planck	259
7.6.1 Relação de Einstein	262
7.6.2 Equação de Fokker-Planck para PE Vetorial com Vários Ruídos	263
7.7 Teoria da Resposta Linear para PE	265
INTRODUÇÃO A SCILAB	271
TRADUÇÃO SCILAB-MATLAB	287
Referências Bibliográficas	289
Índice	292

PREFÁCIO

Este livro foi compilado e editado a partir das notas de aula da disciplina "Métodos Computacionais da Física", lecionada para alunos do curso de Bacharelado em Física da Universidade Federal do Rio Grande do Sul. Hoje esta disciplina está dividida em três semestres, "Métodos Computacionais A, B e C". Em "Métodos Computacionais A" o aluno se familiariza com o computador, com o sistema operacional LINUX, com o processador de texto LATEX, e outros recursos úteis para os físicos. Em "Métodos Computacionais B" o aluno aprende alguns métodos numéricos, inclusive a obter soluções de equações diferenciais ordinárias, para o que os capítulos 1 e 2 deste livro podem ser utilizados. Em "Métodos Computacionais C" o aluno é apresentado a problemas e métodos mais avançados, que são apresentados nos capítulos 3 a 7 deste livro, que tratam, respectivamente de "Equações Diferenciais a Derivadas Parciais", "Teoria de Probabilidade, Variáveis Aleatórias e Processos Estocásticos", "Dinâmica Molecular", "Método Monte Carlo em Física Estatística" e "Dinâmica Estocástica".

Para estabelecer uma linguagem de programação comum a todos os alunos (muitos já sabem programar em alguma linguagem, como FORTRAN, C, C++, PASCAL, etc.), que seja fácil e rápida de aprender, com grande facilidade de interação entre o operador e o computador, e com possibilidade de apresentar resultados em forma gráfica, mesmo durante a execução do programa, optamos por SCILAB. Na primeira edição havíamos optado por MATLAB, mas, devido ao alto custo para se obter o software correspondente, e considerando que SCILAB é "software livre" e pode ser baixado gratuitamente da Internet, nesta segunda edição todos os programas estão

escritos para SCILAB. Apresentamos também um apêndice (Apêndice A), onde se ensina, em poucas páginas, a programar em SCILAB. Ao longo do livro são apresentados muitos programas, prontos para serem executados, todos em SCILAB, e muitos exercícios são sugeridos para o leitor escrever seus próprios programas ou modificar os programas apresentados. Para que o leitor não precise digitar os programas prontos, uma página foi criada na internet, que contém os mesmos:

<http://www.if.ufrgs.br/~cscherer>

Ao leitor que não está familiarizado com a linguagem de programação SCILAB sugerimos que inicie seu estudo pelo Apêndice A, e ao professor que adotar este livro sugerimos também que inicie seu curso pelo Apêndice A.

Um segundo apêndice (Apêndice B) apresenta um "dicionário" das principais diferenças entre SCILAB e MATLAB, de modo que quem preferir trabalhar com MATLAB pode usar os programas do livro e fazer as poucas modificações indicadas nesse apêndice.

Sobre todos os temas tratados neste livro existem livros escritos, alguns dos quais estão relacionados nas "Referências Bibliográficas", no final deste. Cabe, por isso, a pergunta: Por que mais um livro sobre esses temas? Por dois motivos: Primeiro porque, ao que sabemos, nenhum livro publicado trata de todos estes assuntos, e é muito inconveniente, principalmente para os alunos, que uma disciplina adote 6 ou 7 livros textos; segundo, porque um livro escrito em Português facilita bastante o entendimento, pelos estudantes brasileiros, principalmente no caso daqueles que não dominam muito bem a língua inglesa. Além disso, este livro foi escrito com preocupação didática, de modo que os estudantes possam aprender os métodos computacionais nele tratados sem muito auxílio de professor.

Esta segunda edição trás, em relação à primeira, várias melhorias. A mudança de MATLAB para SCILAB nos programas apresentados facilita aos estudantes o uso do "software", pois podem baixar SCILAB da Internet, tanto para LINUX como para WINDOWS ou

MACINTOSH. Ao longo de todo o livro foram substituídos textos, que não nos pareciam estar em forma muito clara para o estudante, por outros que acreditamos serem mais didáticos. Algum material que não constava da primeira edição foi incluído. Alguns erros de digitação da primeira edição foram corrigidos. As novas regras de ortografia da língua portuguesa, em vigor desde janeiro de 2009, foram usadas nesta segunda edição.

Capítulo 1

INTRODUÇÃO AOS MÉTODOS NUMÉRICOS

Muitos bons livros têm sido escritos sobre métodos numéricos, chamados também de Cálculo Numérico, Análise Numérica, etc. Alguns deles estão listados nas referências bibliográficas [1, 2, 3, 4, 5, 6]. Não é nossa pretensão competir com esses livros ou escrever algo melhor. Pretendemos apenas fornecer ao leitor que não teve oportunidade de se familiarizar com métodos numéricos um pouco mais que um resumo de seus tópicos mais importantes, que poderão ser úteis para o entendimento dos métodos computacionais mais usados em Física, que passaremos a apresentar a partir do Capítulo II, além de abordar alguns tópicos importantes para as ciências exatas e engenharias, como “Transformada de Fourier Numérica”, que deixam de ser apresentados em vários livros tradicionais de Cálculo Numérico.

1.1 Raízes de Funções

Nesta seção consideramos uma variável x , possivelmente vetorial, $x = (x_1, x_2, \dots, x_n)$, e uma função f , vetorial se x for vetorial, $f = (f_1, f_2, \dots, f_n)$. O problema que vamos abordar consiste em encontrar pontos da variável x tais que $f(x) = c$, onde c é um valor (vetorial) dado. Este problema pode sempre ser reduzido a encontrar raízes de uma função, neste caso, da função $f(x) - c$. Por isso podemos abordar simplesmente, sem perda de generalidade, o problema de

2 CAPÍTULO 1. INTRODUÇÃO AOS MÉTODOS NUMÉRICOS

encontrar soluções para equações do tipo $f(x) = 0$, ou, em outras palavras, resolver sistemas de n equações, $f_1(x) = 0, \dots, f_n(x) = 0$, com n incógnitas x_1, \dots, x_n .

1.1.1 Iteração

Os procedimentos mais usuais na busca de soluções para equações do tipo $f(x) = 0$ envolvem um processo *iterativo*, como passamos a explicar.

Consideremos uma transformação $x \rightarrow x'$, que representamos por $T(x) = x'$. Vamos aplicar sucessivamente tal transformação, partindo de um ponto inicial x_0 :

$$\begin{aligned} x_1 &= T(x_0) \\ x_2 &= T(x_1) \\ &\vdots \\ x_{k+1} &= T(x_k) \\ &\vdots \end{aligned} \tag{1.1}$$

Se existem pontos x_p tais que $T(x_p) = x_p$, estes se chamam *pontos fixos* da transformação T . O procedimento iterativo acima pode convergir ou não a um ponto fixo. Quando converge se tem

$$\lim_{k \rightarrow \infty} x_k = x_p. \tag{1.2}$$

Geralmente o comportamento da iteração, se converge ou não e, se converge, para qual ponto fixo, depende do ponto inicial x_0 .

Exercício 1.1

Considere a transformação

$$T(x) = x(3 - x^2)/2; \tag{1.3}$$

Partindo de diferentes pontos iniciais, x_0 , efetue o processo iterativo, como indicado na Eq.(1.1), verifique a convergência para os pontos fixos $x = +1$ e $x = -1$. Há ainda um terceiro ponto fixo, $x = 0$, pois $T(0) = 0$. Entretanto a iteração nunca converge para $x = 0$;

por mais próximo de 0 que seja o ponto inicial, x sempre se afastará de 0, indo para +1 ou -1. Por isso o ponto $x = 0$ é um ponto fixo *instável*, enquanto que $x = +1$ ou -1 são *estáveis*. Se o ponto inicial for maior que $\sqrt{5}$, o processo iterativo *diverge*, i.e., vai a infinito.

1.1.2 Funções Escalares de uma Variável

Iniciamos a busca de soluções para equações do tipo $f(x) = 0$ pela situação mais simples, em que f é uma função escalar, real, da variável escalar, real, x . Com os modernos computadores é muito fácil traçar o gráfico da função $f(x)$ e ler, diretamente do gráfico, os pontos onde $f(x) = 0$. Entretanto, pode ocorrer que precisemos das raízes de f com grande precisão, que não pode ser obtida do gráfico, ou pode ser que precisemos de um processo mais automático, para determinar as raízes de f dentro de um programa maior. Dentre os diversos procedimentos possíveis, talvez o mais usado seja o método de Newton-Raphson, que passamos a apresentar.

O Método de Newton-Raphson

Para facilitar a explicação do método de Newton-Raphson (ou simplesmente “método de Newton”) vamos usar a figura 1.1. Vemos que a função $f(x)$ possui duas raízes no intervalo $[0,5]$, $x = a$ e $x = b$. O método de Newton para encontrar as raízes consiste num procedimento iterativo. Digamos que queremos encontrar a raiz $x = b$, ou seja, o valor de b . Iniciamos a iteração a partir de algum ponto “próximo” a b , digamos o ponto x_1 ; traçamos a reta L_1 , tangente ao gráfico da f em $P_1 = (x_1, f(x_1))$; essa reta intercepta o eixo x no ponto x_2 ; traçamos a reta L_2 , tangente ao gráfico da f em $P_2 = (x_2, f(x_2))$; L_2 intercepta o eixo x no ponto x_3 , que já está bastante próximo a b ; continuando o processo chegaremos a um x_k tal que $f(x_k) = 0$, dentro do erro aceitável; x_k é, então, a raiz b .

O procedimento matemático é como segue. A inclinação de L_1 é

$$\left. \frac{df}{dx} \right|_{x_1} = f'(x_1) = \frac{f(x_1) - 0}{x_1 - x_2}. \quad (1.4)$$

4 CAPÍTULO 1. INTRODUÇÃO AOS MÉTODOS NUMÉRICOS

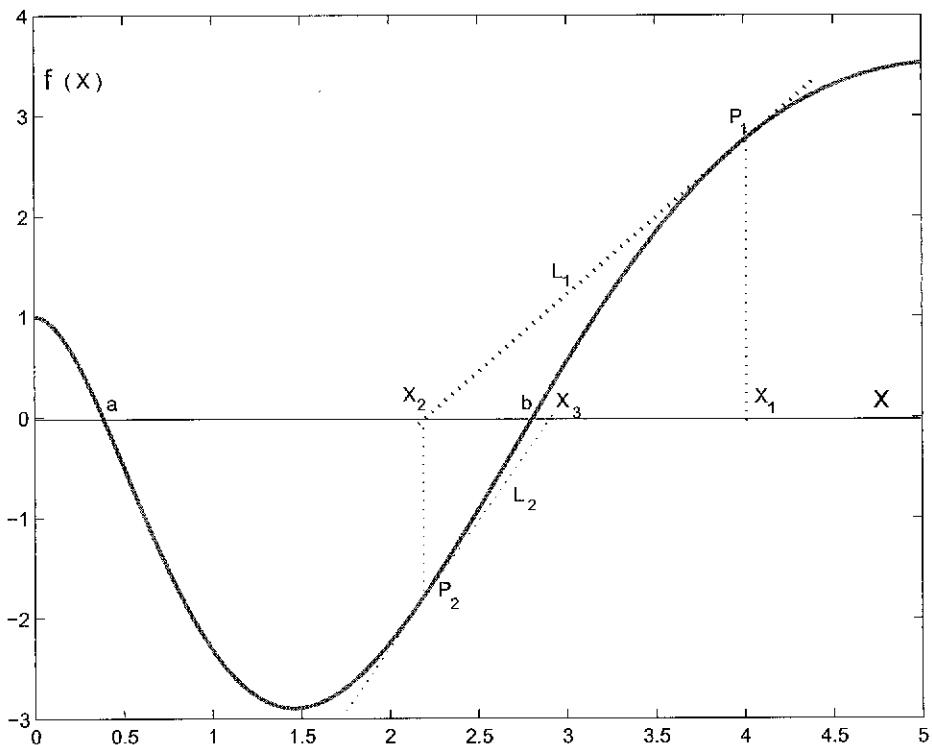


Figura 1.1: Esquema gráfico do método de Newton-Raphson

Dai segue

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}. \quad (1.5)$$

Portanto a transformação $T(x)$ usada no processo iterativo do método de Newton é

$$T(x) = x - \frac{f(x)}{f'(x)}. \quad (1.6)$$

Dissemos acima que o ponto x inicial deve ser “próximo” à raiz procurada. Que significa “próximo”? Observemos novamente a figura 1.1. Se escolhermos como x inicial, por exemplo, $x = 1$ o processo nos levará à raiz $x = a$, ou, talvez, a alguma raiz em $x < 0$; se escolhermos $x > 5$, possivelmente o processo será divergente.

Exercício 1.2

A figura 1.1 representa a função

$$f(x) = (x^4 - 10x^2) \exp(-x) + 1 ; \quad (1.7)$$

obtenha uma expressão para a derivada $f'(x)$ e aplique o método de Newton para encontrar as raízes da f no limite da precisão numérica do computador.

Observação sobre a obtenção da derivada $f'(x)$: O método de Newton é também muito usado em situações em que a $f(x)$ não é uma simples função analítica, mas uma função cujos valores sejam obtidos por algum processo complexo de computação. Nestes casos não se tem uma expressão analítica, como no exemplo acima, para a derivada $f'(x)$. O procedimento, então, consiste em usar para a derivada uma *diferença finita dividida de primeira ordem*, como explicaremos a seguir.

Diferenças Finitas

Invocando a definição de derivada da função $f(x)$, no ponto x ,

$$\frac{df(x)}{dx} = f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \quad (1.8)$$

o procedimento numérico para a derivada consiste em escolher um incremento pequeno Δx tal que o segmento do gráfico da f , entre x e $x + \Delta x$ possa ser considerado um segmento de reta, e definir a derivada numericamente por

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} . \quad (1.9)$$

A diferença $f(x + \Delta x) - f(x)$ é denominada, em cálculo numérico, “diferença finita de primeira ordem da f , em x ”, e a razão que aparece à direita da igualdade na Eq.(1.9), que é uma aproximação para a derivada primeira da f , em x , se chama “diferença finita dividida de primeira ordem”. A Eq.(1.9) é a derivada numérica à direita, porque

6 CAPÍTULO 1. INTRODUÇÃO AOS MÉTODOS NUMÉRICOS

o incremento em x é tomado à direita do ponto x ; semelhantemente, a derivada numérica à esquerda é

$$f'(x) = \frac{f(x) - f(x - \Delta x)}{\Delta x} . \quad (1.10)$$

Para funções contínuas e deriváveis, no limite $\Delta x \rightarrow 0$, as Eqs. (1.9) e (1.10) levam ao mesmo resultado, mas para Δx finito elas correspondem a aproximações distintas. Quando se precisa de mais precisão na derivada, a forma *centrada*, abaixo, é geralmente melhor:

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2 \Delta x} . \quad (1.11)$$

Semelhantemente se define a diferença finita dividida de segunda ordem,

$$f''(x) = \frac{f'(x + \Delta x) - f'(x)}{\Delta x} = \frac{f(x + \Delta x) + f(x - \Delta x) - 2f(x)}{\Delta x^2} . \quad (1.12)$$

onde usamos a Eq.(1.10) para as f' no numerador Eq.(1.12) obtendo assim uma forma centrada. Obviamente esta equação é uma aproximação numérica para a derivada segunda da f . Nos capítulos II e III, sobre equações diferenciais, usaremos as equações acima com muita frequência.

A função “raiz.sci”

SCILAB não possui uma função intrínseca (como a função “fzero” de MATLAB) que encontra raízes de uma função qualquer, não polinomial. Por isso apresentamos, a seguir, o programa “raiz.sci”, que cria a “function” “raiz(fun,x0)”, onde ‘fun’ é o nome da função cuja raiz próxima a x_0 queremos obter. ‘fun’ pode ser uma função intrínseca de SCILAB, ou uma “function” definida num arquivo, e disponibilizada no “workspace” por “getf” ou “exec” ou uma “function” criada na linha de comando. O programa usa derivada numérica centrada para encontrar df/dr .

Programa: raiz.sci

```

function r=raiz(fun,r0)
// raiz.sci : encontra raiz de uma função
// qualquer, 'fun', próxima ao real x0;
r=r0;
dr=1.e-10;
f=fun(r);
tol=100*%eps;
n=0;
while(abs(f) > tol & n < 10)
    n=n+1;
    f1=fun(r-dr); f2=fun(r+dr);
    dfdr=(f2-f1)/(2*dr);
    r=r-f/dfdr;
    f=fun(r);
end
disp(n,'n=',f,' f= ',r,' raiz=')
endfunction

```

Exercício 1.3:

Use “raiz” para encontrar as raízes da função $f(x)$ definida no Exercício 1.2. Dica: copie o programa acima para um arquivo “raiz.sci”, disponibilize a “function” raiz executando “getf raiz.sci”, crie na linha de comando a função func(x),

```

function f=func(x);
f=(x^4-10*x^2)*exp(-x)+1;
endfunction;

```

Então comande: “raiz(func,x0)”, onde x_0 é um valor de x próximo a uma raiz que você vê no gráfico. Como resultado você recebe o valor da raiz, o valor da função naquele ponto (que dificilmente será exatamente 0) e o número n de ciclos do método de Newton que foram executados para chegar a esse resultado.

Raízes de Polinômios

Um polinômio de grau n ,

$$P(x) = a_1x^n + a_2x^{n-1} + \cdots + a_nx + a_{n+1}, \quad (1.13)$$

possui n raízes, sendo que algumas delas podem ser iguais e também podem ser complexas. Que significa duas raízes serem iguais? Seu valor não indicaria apenas uma raiz? A resposta está associada à decomposição do polinômio num produto:

$$P_n(x) = a_1(x - x_1)(x - x_2) \cdots (x - x_n), \quad (1.14)$$

onde x_1, x_2, \dots, x_n são as n raízes. Por exemplo, o polinômio

$$P_2(x) = 2x^2 - 4x + 2$$

se decompõe na forma

$$P_2(x) = 2(x - 1)(x - 1),$$

possuindo, portanto, duas raízes iguais $x_1 = x_2 = 1$. Dizemos que $x = 1$ é uma *raiz dupla* de P_2 . No caso de várias raízes iguais se fala em *raiz múltipla*. O polinômio

$$P_2(x) = x^2 - 2x + 2$$

se decompõe na forma

$$P_2(x) = (x - (1 + i))(x - (1 - i)),$$

possuindo, portanto, duas raízes complexas distintas, $x_1 = 1 + i$ e $x_2 = 1 - i$.

O método de Newton, visto acima, pode ser usado para encontrar as raízes reais dos polinômios. Ele não pode ser usado para as raízes complexas e não funciona bem no caso de raiz múltipla. A determinação das raízes dos polinômios tem sido objeto de muito estudo e ocupa muitas páginas dos livros de álgebra e dos de métodos numéricos. Não pretendemos nos alongar neste assunto. Preferimos, alternativamente, indicar a excelente rotina SCILAB que encontra

todas as raízes com eficiência e precisão:

Rotina SCILAB: roots(a)

O argumento a é o vetor cujos elementos são os coeficientes do polinômio, i.e., $a = [a_1, a_2, \dots, a_n, a_{n+1}]$, sendo a_1 o coeficiente de x^n e a_{n+1} o termo independente de x . Sugerimos ao leitor que escolha alguns polinômios e encontre as raízes usando “roots”.

1.1.3 Funções Vetoriais de Variável Vetorial

Seja x uma variável vetorial de n componentes, $x = (x_1, x_2, \dots, x_n)$ e $f(x)$ uma função vetorial de x , de n componentes, i.e., $f = (f_1, f_2, \dots, f_n)$. O problema que pretendemos abordar consiste em encontrar os vetores x para os quais $f(x) = 0$. Dividimos nossa abordagem em duas etapas, a primeira para as situações em que f é uma função linear de x e a segunda que trata de funções não lineares.

Sistemas lineares

Pretendemos encontrar as raízes de equações do tipo $f(x) = 0$ quando $f = Ax - b$, onde A é uma matriz $n \times n$ de elementos constantes e b é um vetor constante, ou seja, procuramos x tal que

$$Ax = b. \quad (1.15)$$

Explicitamente, em termos dos elementos de matriz, escrevemos

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad (1.16)$$

ou, em forma de um sistema de equações lineares,

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \quad (1.17)$$

10 CAPÍTULO 1. INTRODUÇÃO AOS MÉTODOS NUMÉRICOS

No ensino médio o estudante aprende a resolver sistemas como o da Eq.(1.17) pela *Regra de Cramer*, que exige o cálculo de $n + 1$ determinantes. Cada determinante, calculado conforme sua definição, contém $n!$ termos, o que torna este procedimento impraticável para n grande. Outra maneira, muito usada em álgebra linear, consiste em calcular a matriz inversa, A^{-1} , de A , definida por

$$A A^{-1} = A^{-1} A = I, \quad (1.18)$$

onde I é a matriz identidade de dimensão n . Multiplicando-se a Eq.(1.15), à esquerda, por A^{-1} obtém-se

$$x = A^{-1}b. \quad (1.19)$$

Embora tenha uma aparência muito simpática, esta forma de resolver o sistema sofre do mesmo problema que a regra de Cramer, pois encontrar a matriz inversa também exige o cálculo de muitos determinantes. Existem procedimentos matemáticos que permitem calcular determinantes de matrizes grandes sem se fazer uso direto de sua definição, mas apenas de algumas de suas propriedades, e que tornam o cálculo muito mais rápido. Tais procedimentos estão, geralmente, baseados num método chamado *eliminação de Gauss*. Entretanto, por este método pode-se obter a solução do sistema linear diretamente, i.e., sem usar a regra de Cramer ou a inversão da matriz A e, portanto, sem calcular determinantes. Passamos, por isso, a apresentá-lo.

Eliminação de Gauss

Este método está baseado na propriedade dos sistemas lineares, Eq.(1.17), de que sua solução não se altera quando se realiza qualquer uma das seguintes operações:

- Multiplicar qualquer equação por uma constante não nula;
- Sumar a uma das equações um múltiplo de outra;
- Trocar a ordem das equações.

O procedimento consiste em aplicar as operações acima até deixar o sistema na forma

$$\begin{array}{ccccccccc}
 k_{11}x_1 & + & k_{12}x_2 & + & k_{13}x_3 & + & \cdots & + & k_{1n}x_n = c_1 \\
 0 & + & k_{22}x_2 & + & k_{13}x_3 & + & \cdots & + & k_{2n}x_n = c_2 \\
 0 & + & 0 & + & k_{23}x_3 & + & \cdots & + & k_{3n}x_n = c_3 \\
 \cdots & & \cdots & & \cdots & & \cdots & & \cdots \\
 0 & + & 0 & + & 0 & + & \cdots & + & k_{nn}x_n = c_n
 \end{array} \quad (1.20)$$

onde os coeficientes k_{ij} e os c_j são obtidos dos a_{ij} e dos b_j na forma como explicaremos a seguir. Os x_j são então obtidos facilmente da Eq.(1.20) por substituição de baixo para cima: da última linha se obtém $x_n = c_n/k_{nn}$; substituindo este valor de x_n na penúltima linha se obtém x_{n-1} , e assim sucessivamente até obter x_1 da primeira linha. O procedimento para obter os coeficientes do sistema transformado, Eq.(1.20) é o seguinte:

- 1) Chamamos de “linha pivô” a primeira linha e o elemento a_{11} se chama “elemento pivô” para a seguinte operação: adicionamos a cada linha, da segunda até a última, um múltiplo da linha pivô, tal que o coeficiente de x_1 fique nulo; simbolicamente,

$$a_{ij} \rightarrow a_{ij} - a_{1j}a_{i1}/a_{11}$$

$$b_i \rightarrow b_i - b_1a_{i1}/a_{11}$$

- 2) Escolhemos a segunda linha, na sua nova forma, como linha pivô; o novo elemento a_{22} é o novo pivô; adicionamos a cada linha, da terceira até a última, um múltiplo da linha pivô, tal que o coeficiente de x_2 fique nulo;

- 3) Procedemos analogamente, escolhendo a linha 3 como pivô, e assim sucessivamente até tornar nulos todos os elementos à esquerda dos pivôs.

Observação: se um pivô for nulo não podemos proceder da mesma forma por que as transformações contém divisão pelo pivô; neste caso se permuta a linha pivô com uma das linhas seguintes, que será então a nova linha pivô, tal que o elemento pivô não seja mais nulo. Esta operação, conhecida por “pivotamento”, também deve ser feita

12 CAPÍTULO 1. INTRODUÇÃO AOS MÉTODOS NUMÉRICOS

quando o pivô for muito pequeno (em relação aos demais elementos), de maneira que as divisões pelo pivô possam trazer importantes erros numéricos.

Exemplo 1.1:

Vamos usar o método da *Eliminação de Gauss* para resolver o seguinte sistema linear:

$$\begin{array}{rclllll} 2x_1 & + & x_2 & + & x_3 & + & x_4 = 11 \\ x_1 & + & 0 & - & 3x_3 & + & 2x_4 = 0 \\ 2x_1 & - & 2x_2 & - & 20x_3 & + & 0 = -62 \\ 0 & + & x_2 & + & 0 & - & x_4 = -2 \end{array} \quad (1.21)$$

A linha pivô é a primeira e o elemento pivô é 2. Da segunda linha subtraímos $1/2$ da pivô, da terceira subtraímos a pivô e a quarta deixamos inalterada, para obter

$$\begin{array}{rclllll} 2x_1 & + & x_2 & + & x_3 & + & x_4 = 11 \\ 0 & - & \frac{1}{2}x_2 & - & \frac{7}{2}x_3 & + & \frac{3}{2}x_4 = -\frac{11}{2} \\ 0 & - & 3x_2 & - & 21x_3 & - & x_4 = -73 \\ 0 & + & x_2 & + & 0 & - & x_4 = -2 \end{array} \quad (1.22)$$

Agora a linha pivô passa a ser a segunda e o elemento pivô $-\frac{1}{2}$. Da terceira linha subtraímos 6 vezes a pivô e à quarta somamos 2 vezes a pivô, para obter

$$\begin{array}{rclllll} 2x_1 & + & x_2 & + & x_3 & + & x_4 = 11 \\ 0 & - & \frac{1}{2}x_2 & - & \frac{7}{2}x_3 & + & \frac{3}{2}x_4 = -\frac{11}{2} \\ 0 & + & 0 & - & 0 & - & 10x_4 = -40 \\ 0 & + & 0 & + & -7x_3 & + & 2x_4 = -13 \end{array} \quad (1.23)$$

Não podemos tomar a terceira linha como nova pivô, pois o elemento pivô seria 0. Por isso permutamos a terceira linha com a quarta, para obter

$$\begin{array}{rclllll} 2x_1 & + & x_2 & + & x_3 & + & x_4 = 11 \\ 0 & - & \frac{1}{2}x_2 & - & \frac{7}{2}x_3 & + & \frac{3}{2}x_4 = -\frac{11}{2} \\ 0 & + & 0 & + & -7x_3 & + & 2x_4 = -13 \\ 0 & + & 0 & - & 0 & - & 10x_4 = -40 \end{array} \quad (1.24)$$

Com isto a matriz dos coeficientes já tem a forma triangular desejada. Da quarta linha obtemos $x_4 = 4$; substituindo este valor na terceira linha obtemos $x_3 = 3$; substituindo estes dois valores na segunda linha obtemos $x_2 = 2$ e da primeira linha obtemos $x_1 = 1$, concluindo assim a solução do sistema.

Caso Especial: Matriz Tridiagonal

Um caso especial de sistemas lineares, que ocorre com frequência em problemas de Física é quando a matriz A da Eq.(1.15) tem a forma “tridiagonal”, ou seja

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \cdots & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & \cdots & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & a_{n-1,n} \\ 0 & 0 & 0 & 0 & \cdots & a_{n,n-1} & a_{n,n} \end{pmatrix} \quad (1.25)$$

onde os únicos elementos não nulos são os da diagonal principal e os das duas “subdiagonais”, contíguas à diagonal principal.

Este caso especial é importante na solução de equações diferenciais parciais de segunda ordem, como veremos no Capítulo III além de outras situações que envolvem derivadas de segunda ordem, como a interpolação por “spline cúbico”, como veremos na seção 1.2 deste capítulo. A aplicação do processo de eliminação de Gauss, neste caso, torna-se muito simples porque recai numa fórmula de recorrência.

Não seria lógico, no caso de matrizes grandes (digamos $n \sim 100$) reservar memória do computador para armazenar todos os zeros da matriz A . Por isso vamos mudar a notação, chamando de b_i os elementos da diagonal principal e de a_i e c_i os elementos das duas subdiagonais. Com esta notação escrevemos a seguir o sistema de equações lineares a ser resolvido, chamando de x_i as variáveis incógnitas e d_i os termos independentes:

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & 0 & \cdots & 0 \\ 0 & a_3 & b_3 & c_3 & \cdots & 0 \\ 0 & 0 & a_4 & b_4 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & c_{n-1} \\ 0 & 0 & 0 & \cdots & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \cdots \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ \cdots \\ d_n \end{pmatrix} \quad (1.26)$$

Vamos aplicar o método de eliminação de Gauss à Eq.(1.26). Observamos que no primeiro passo do processo precisamos alterar apenas a segunda fila, pois as demais já possuem zeros na primeira coluna. No segundo passo precisamos alterar apenas a terceira fila, pois as seguintes já possuem zero na segunda coluna, e assim sucessivamente. Introduzindo a notação

$$\begin{aligned} \beta_1 &= b_1 \\ \beta_{i+1} &= b_{i+1} - a_{i+1}c_i/\beta_i & i = 1, \dots, n-1 \\ \delta_1 &= d_1 \\ \delta_{i+1} &= d_{i+1} - a_{i+1}\delta_i/\beta_i & i = 1, \dots, n-1 \end{aligned} \quad (1.27)$$

a equação transformada fica

$$\begin{pmatrix} \beta_1 & c_1 & 0 & 0 & \cdots & 0 \\ 0 & \beta_2 & c_2 & 0 & \cdots & 0 \\ 0 & 0 & \beta_3 & c_3 & \cdots & 0 \\ 0 & 0 & 0 & \beta_4 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & c_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & \beta_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \cdots \\ x_n \end{pmatrix} = \begin{pmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \\ \cdots \\ \delta_n \end{pmatrix} \quad (1.28)$$

Agora a obtenção da solução torna-se trivial:

$$\begin{aligned} x_n &= \delta_n/\beta_n \\ x_i &= (\delta_i - c_i x_{i+1})/\beta_i & i = n-1, n-2, \dots, 1. \end{aligned} \quad (1.29)$$

É claro que no procedimento computacional não se precisa escrever as Eqs. (1.26) e (1.28), bastando usar as fórmulas de recorrência para os β_i , δ_i e x_i , Eqs. (1.27) e (1.29).

Funções Vetoriais Não Lineares

O problema de encontrar raizes de equações vetoriais do tipo $f(x) = 0$, quando as componentes de f são funções não lineares das componentes de x é bem mais complexo do que no caso dos sistemas lineares, que acabamos de ver. Os diversos métodos conhecidos são variantes da procura de pontos fixos de transformações (veja seção 1.1.1) por procedimento iterativo. Vamos apresentar aqui apenas o método que consiste na extensão do método de Newton (seção 1.1.2) para várias variáveis. No caso de função escalar de uma variável usamos sua derivada para traçar um segmento de reta até a abscissa do gráfico da função. No caso de várias variáveis temos que usar a *Matriz Jacobiana*, definida como segue: seja $f(x) = [f_1(x_1, \dots, x_n), f_2(x_1, \dots, x_n), \dots, f_n(x_1, \dots, x_n)]$; os elementos J_{ij} da matriz Jacobiana são definidos como as derivadas parciais das f_i em relação às x_j ; explicitamente, a matriz J é definida por

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \quad (1.30)$$

No caso escalar se escolhe um valor para o erro tolerado, tol , para que quando $|f(x)| < tol$, onde $|f(x)|$ é o valor absoluto de $f(x)$, se aceita x como raiz de f . Em várias dimensões $|f(x)|$ é a norma do vetor $f(x)$, que pode ser definida como o máximo entre os valores absolutos das componentes de $f(x)$ ou como o módulo de $f(x)$, i.e., a raiz quadrada da soma dos quadrados das componentes.

Abaixo apresentamos os passos do método de Newton para função vetorial em estreito paralelismo com o método para função escalar.

1.1.4 Método de Newton Generalizado

Caso de uma variável	Caso de várias variáveis
escolhe valor inicial $x^{(1)}$	escolhe vetor inicial $x^{(1)}$
calcula a função $f(x^{(1)})$ e sua derivada $f'(x^{(1)})$	calcula função vetorial $f(x^{(1)})$ e a matriz Jacobiana $J(x^{(1)})$
calcula deslocamento $s :$ $s = -f(x^{(1)})/f'(x^{(1)})$	calcula deslocamento $s :$ $s = -J^{-1}(x^{(1)}) f(x^{(1)})$
novo valor de $x :$ $x^{(2)} = x^{(1)} + s$	novo vetor $x :$ $x^{(2)} = x^{(1)} + s$
calcula $x^{(3)}, \dots, x^{(k)}$ até que $ f(x^{(k)}) < tol$	calcula $x^{(3)}, \dots, x^{(k)}$ até que $ f(x^{(k)}) < tol$
assume $x^{(k)}$ como raiz e encerra.	assume $x^{(k)}$ como raiz e encerra.

Um exemplo de uso do esquema acima pode ser encontrado na seção 1.2.3 (Ajuste), subseção "Ajuste a uma curva qualquer"

1.2 Aproximações Numéricas de Funções

Uma função real de variável real é representada em cálculo numérico computacional por um conjunto de pares ordenados de números reais, um número para a variável e outro para a função. Assim, embora a variável real x , definida, por exemplo, no intervalo $[0, 2\pi]$, e a função $f(x) = \sin(x)$ sejam variáveis contínuas, sua representação numérica é feita em um número finito, arbitrário, de valores de x . Por várias razões pode ocorrer que o número de pontos usados

para representar uma função seja muito menor do que o que seria necessário para que o resultado das operações que desejamos executar sobre a mesma tenha a precisão desejada. Por exemplo, os pontos foram obtidos por um procedimento experimental que gerou apenas um pequeno número deles ou mesmo por procedimento computacional que demanda muito tempo de CPU. Ou então, o procedimento que queremos executar exige uma expressão analítica para a função. Há muitas maneiras, denominadas genericamente de *interpolação*, de se obter uma função contínua, mais ou menos suave, cujos valores representam uma aproximação aceitável para a função representada por um conjunto de pontos dados. Algumas delas são apresentadas nas subseções seguintes.

1.2.1 Interpolação Polinomial

Dado um conjunto de $n + 1$ pontos no plano (gráfico da função $f(x)$), tais como $[(x_1, f_1), (x_2, f_2), \dots, (x_{n+1}, f_{n+1})]$, há um polinômio de grau n , $P_n(x)$, cujo gráfico passa pelos $n + 1$ pontos dados, ou seja, $P_n(x_i) = f_i$, $i = 1, \dots, n + 1$. Assim, por um ponto único passa uma reta horizontal (polinômio de grau zero = constante), por dois pontos passa uma reta (polinômio de primeiro grau), por três pontos passa uma parábola (polinômio de segundo grau), etc. Há muitas maneiras pela qual se pode encontrar o polinômio de grau n que passa pelos $n + 1$ pontos dados, embora ele seja único, ou seja, o polinômio pode ser escrito de diversas formas equivalentes. Talvez a forma mais simples seja a que se chama *Polinômio Interpolador de Lagrange*:

$$P_n(x) = \sum_{i=1}^{n+1} p_i(x) f_i \quad (1.31)$$

onde cada $p_i(x)$ é um polinômio de grau n , definido por

$$p_i(x) = \frac{(x - x_1)(x - x_2) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_{n+1})}{(x_i - x_1)(x_i - x_2) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_{n+1})}. \quad (1.32)$$

Note que $p_i(x_j) = \delta_{i,j}$ e, portanto, segue da Eq.(1.31) que $P(x_j) = f_j$. Como o polinômio de grau n que passa pelos $n + 1$ pontos é único,

18 CAPÍTULO 1. INTRODUÇÃO AOS MÉTODOS NUMÉRICOS

segue que o polinômio interpolador de Lagrange é esse polinômio.

O programa “lagrange.sci”, a seguir, é uma “function” que obtém o polinômio interpolador de Lagrange para um conjunto de n pontos (x_i, f_i) dados.

Programa: lagrange.sci

```
function P=lagrange(x,f,X)
// Polinômio Interpolador de Lagrange; no workspace
// deve haver os vetores x=(x1, x2, ..., xn),
// f=(f1, f2, ..., fn), e X=(X1, X2, ..., XN);
// (xi,fi) são os pontos por onde deve passar a função
// P(X); Usualmente escolhe-se N>>n, X1=x1 e XN=xn;

[k1,k2]=size(f);           // as 4 linhas iniciais são
if(k1==1); f=f'; end;      // para transformar os vetores
[k1,k2]=size(X);          // f e X em colunas, caso eles
if(k1==1); X=X'; end;     // sejam filas no workspace.
n=length(x);
N=length(X);
p=ones(N,n);
for i=1:n;
    for j=1:i-1;
        p(:,i)=p(:,i).*(X-x(j))/(x(i)-x(j));
    end
    for j=i+1:n;
        p(:,i)=p(:,i).*(X-x(j))/(x(i)-x(j));
    end
end
P=p*f;
clf
plot(x,f,'o',X,P,'r')
```

Exercício 1.4

Use “lagrange” para interpolar os pontos definidos por $x=[-5, -3, -1, 1, 3, 5]$ e $f=[2, 3, 5, 4, 2, 2]$, em $X=-5:0.1:5$.

Exemplo 1.2:

Dados os 10 pontos (x_i, f_i) destacados na Figura 1.2 e o vetor $X = x_1 : .01 : x_{10}$, e disponibilizado no workspace “lagrange.sci”, comandando-se “P=lagrange(x,f,X)”, serão produzidos os valores numéricos do polinômio P, de grau 9, em todos os valores de X e que passa pelos pontos (x_i, f_i) e apresentado o gráfico correspondente à linha cheia e os pontos destacados na figura. A linha tracejada é uma interpolação por “spline cúbico”, como veremos na próxima seção. Observa-se que nas extremidades inicial e final do gráfico a interpolação por polinômio de Lagrange deixa muito a desejar. Isto costuma ocorrer sempre que o polinômio for de grau relativamente alto, como neste caso. Na próxima seção veremos “interpolação por segmentos”, que requerem polinômios de baixo grau e dão resultados bem mais satisfatórios.

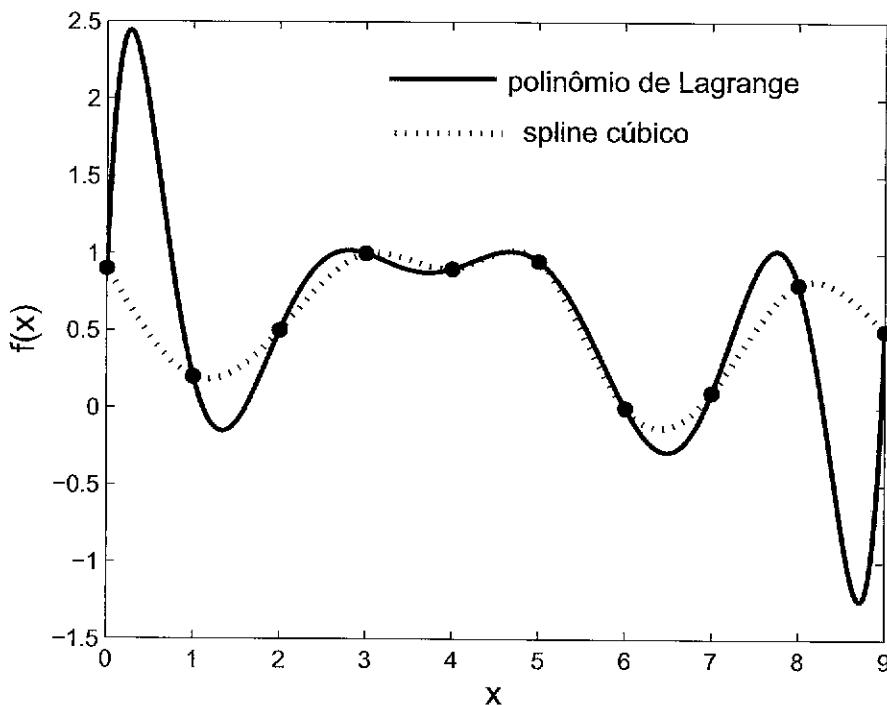


Figura 1.2: 10 pontos arbitrários (o), interpolados por polinômio de Lagrange de grau 9 (linha cheia) e pela função “cubico(x,f,X)” (linha pontilhada)

1.2.2 Interpolação por Segmentos

A interpolação polinomial, que acabamos de ver, tem a qualidade de ser uma função analítica, i.e., possuir todas derivadas contínuas. Entretanto, se o número de pontos interpolados é grande, o polinômio interpolador é de grau muito alto. Isto pode resultar em curvas que, embora passem por todos os pontos interpolados, apresentam comportamento estranho entre os mesmos, especialmente nas regiões próximas aos pontos extremos. Uma maneira alternativa de interpolar consiste em usar um polinômio diferente, de baixo grau, para cada segmento da função entre dois pontos consecutivos. As duas maneiras mais usuais de se fazer interpolação por segmentos são a interpolação linear e a forma chamada “spline cúbico”, que passamos a descrever.

Interpolação Linear

Suponhamos que a função $f(x)$ seja conhecida nos pontos x_i , $i = 1, 2, \dots, n + 1$, onde $f(x_i) = f_i$. Temos então n intervalos entre pontos consecutivos (x_i, x_{i+1}) . A maneira mais simples de unir estes pontos por uma linha contínua é traçar uma reta entre cada dois pontos consecutivos. Vamos chamar de $g_i(x)$ a reta que une os pontos (x_i, f_i) e (x_{i+1}, f_{i+1}) . A equação da g_i é, então,

$$g_i(x) = f_i + (x - x_i) \frac{f_{i+1} - f_i}{x_{i+1} - x_i} \equiv f_i + (x - x_i) \frac{\Delta f_i}{\Delta x_i}. \quad (1.33)$$

A figura 1.3 mostra a interpolação linear (linha tracejada) para um oscilador harmônico amortecido quando os pontos são dados a cada 1.5 radianos. A linha cheia também é uma interpolação linear, mas entre pontos que distam apenas 0.15 radianos. Como se vê, a interpolação linear é uma aproximação grosseira à curva exata quando os pontos são bastante distantes, mas pode ser tão próxima da exata quanto se queira se pudermos usar pontos arbitrariamente próximos. O comando “plot” de SCILAB, assim como outras rotinas gráficas para computador, usam, geralmente, a interpolação linear como “default”.

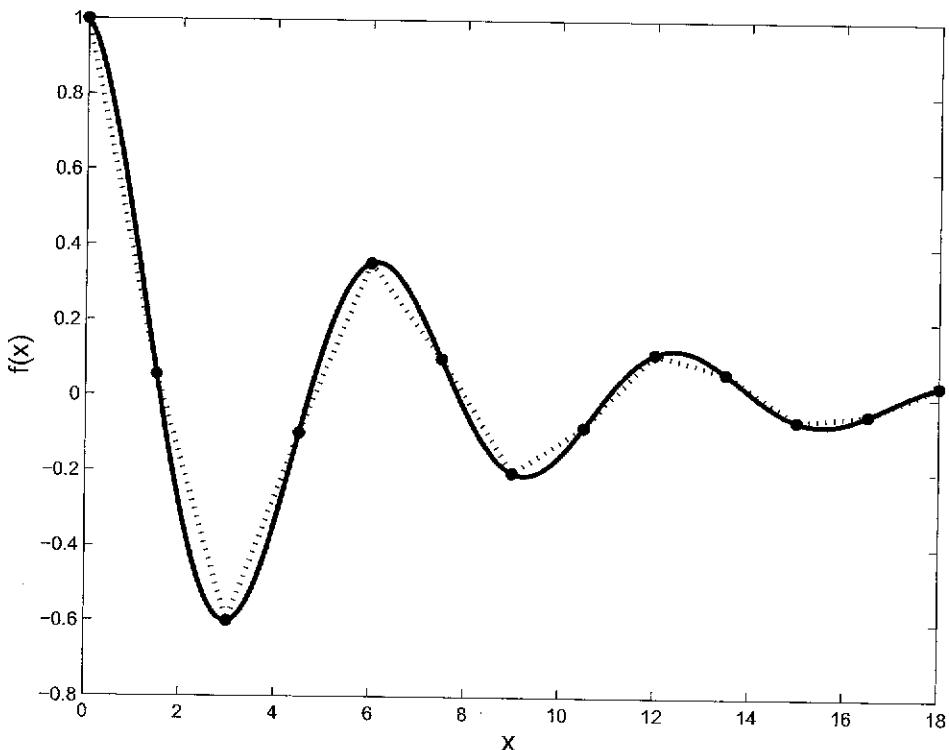


Figura 1.3: Oscilador Harmônico Amortecido: interpolação linear entre os pontos destacados (linha pontilhada) e interpolação linear entre pontos 10 vezes mais próximos (linha cheia)

Splines

A palavra inglesa “spline” significa uma régua flexível, usada para traçar linhas “suaves”, passando por pontos que não se encontram sobre uma reta. Em português ela é conhecida como “curva francesa”. Em interpolação numérica se fala em “spline de grau p ” quando cada dois pontos consecutivos são ligados por um polinômio de grau p . O mais usado é o “spline cúbico”. Com a mesma notação da seção anterior (Interpolação Linear), define-se o “spline cúbico” como a linha contínua, $G(x)$, formada pela união das funções

$$g_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i , \quad (1.34)$$

22 CAPÍTULO 1. INTRODUÇÃO AOS MÉTODOS NUMÉRICOS

que interpola entre x_i e x_{i+1} , onde as constantes a_i, b_i, c_i e d_i são escolhidas de maneira a que se satisfaçam as seguintes propriedades:

- 1) $g_i(x_i) = f_i$ isto é, a função $G(x)$ passa pelos pontos (x_i, f_i)
- 2) $g_i(x_i) = g_{i-1}(x_i)$, isto é, a função $G(x)$ é contínua;
- 3) $g'_i(x_i) = g'_{i-1}(x_i)$, isto é, a derivada primeira da $G(x)$ também é contínua;
- 4) $g''_i(x_i) = g''_{i-1}(x_i)$, isto é, a derivada segunda da $G(x)$ também é contínua.

Há um total de n funções g_i e, portanto, $4n$ constantes a serem determinadas. A propriedade 1 determina as n constantes $d_i = f_i$, devendo ainda serem determinadas $3n$ constantes a_i, b_i e c_i . As propriedades 2, 3 e 4 se aplicam aos pontos x_2, x_3, \dots, x_n e fornecem, portanto, $3(n-1)$ equações. Além disso, $g_n(x_{n+1}) = f_{n+1}$, que é mais uma equação a ser usada na determinação das constantes. Temos assim $3n - 2$ equações para determinar $3n$ constantes. Esta liberdade que temos, de arbitrar mais duas condições, é geralmente usada na determinação da forma dos segmentos nas extremidades. Uma escolha usual é $g''_1(x_1) = 0$ e $g''_n(x_{n+1}) = 0$, ou seja, a função interpoladora chega às extremidades do intervalo como uma função linear.

O procedimento algébrico para se chegar ao sistema linear que determina todas as constantes não é difícil mas bastante trabalhoso. Há várias formas alternativas de fazê-lo, que podem ser encontradas, por exemplo, nos livros de cálculo numérico, listados nas Referências Bibliográficas, no final deste livro. Vamos apresentar aqui apenas as linhas gerais do procedimento que é apresentado por DeVries[4].

Definimos alguns símbolos:

$$\begin{aligned} h_i &= x_{i+1} - x_i \\ G_i &= G(x_i) = f_i \\ G'_i &= G'(x_i) \\ G''_i &= G''(x_i) \end{aligned} \tag{1.35}$$

Da Eq.(1.34) e suas propriedades 1 a 4 segue:

$$\begin{aligned}
 G''(x_i) &= 2b_i \\
 \text{e portanto } b_i &= G''_i/2 \\
 G''(x_{i+1}) &= 6a_i h_i + 2b_i \\
 \text{e portanto } a_i &= (G''_{i+1} - G''_i)/6h_i \\
 G_{i+1} &= f_{i+1} = a_i h_i^3 + b_i h_i^2 + c_i h_i + f_i \\
 \text{e portanto } c_i &= (f_{i+1} - f_i)/h_i - (h_i G''_{i+1} + 2h_i G''_i)/6.
 \end{aligned} \tag{1.36}$$

Logo, todas as constantes podem ser obtidas, facilmente, do conjunto dos G''_i , $i = 1, \dots, n+1$, que devem, portanto, ser obtidos em primeiro lugar. Por procedimento algébrico, partindo da Eq.(1.34) e usando as propriedades 1 a 4 das g_i e a escolha $g''_1(x_1) = 0$ e $g''_{n+1}(x_{n+1}) = 0$ se obtém (veja[4])

$$\left(\begin{array}{cccccc|c}
 1 & 0 & 0 & \cdots & 0 & G''_1 \\
 0 & 2(h_1 + h_2) & h_2 & \cdots & 0 & G''_2 \\
 0 & h_2 & 2(h_2 + h_3) & \cdots & 0 & G''_3 \\
 0 & 0 & h_3 & \cdots & 0 & G''_4 \\
 \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
 0 & 0 & \cdots & 2(h_{n-1} + h_n) & 0 & G''_n \\
 0 & 0 & \cdots & 0 & 1 & G''_{n+1}
 \end{array} \right) = \left(\begin{array}{c} 0 \\ k_2 \\ k_3 \\ k_4 \\ \vdots \\ \vdots \\ k_n \\ 0 \end{array} \right) \tag{1.37}$$

onde

$$k_i = \frac{6(f_{i+1} - f_i)}{h_i} - \frac{6(f_i - f_{i-1})}{h_{i-1}} \quad i = 2, \dots, n \tag{1.38}$$

Claramente, essas expressões se simplificam se as distâncias h_i entre pontos consecutivos forem todas iguais, $h_i = h$.

A linha pontilhada da Figura 1.2 é uma interpolação por “spline cúbico”.

Programa: cubico.sci

O programa que segue é uma “function” que calcula spline cúbico pelo procedimento que acabamos de descrever, com as condições $g''_1(x_1) = 0$ e $g''_{n+1}(x_{n+1}) = 0$. Os vetores x e f , que definem os pontos (x_i, f_i) a serem interpolados devem estar definidos no workspace. As distâncias entre os x_i consecutivos não precisam ser iguais.

24 CAPÍTULO 1. INTRODUÇÃO AOS MÉTODOS NUMÉRICOS

```
function G=cubico(x,f,X);
// cubico(x,f,X) é uma function que interpola por
// spline cúbico, nos pontos X_j de X os pontos
// (x_i,f_i) da função f(x); para se obter uma curva
// suave deve-se escolher os pontos X_j muito mais
// próximos que os x_i.
n=length(x)-1;
N=length(X);
h(1:n)=x(2:n+1)-x(1:n);
b=ones(1,n+1);
b(2:n)=2*(h(1:n-1)+h(2:n));
a=zeros(1,n+1);
a(3:n)=h(2:n-1);
c=zeros(1,n+1);
c(2:n-1)=h(2:n-1);
k=zeros(1,n+1);
k(2:n)=6*(f(3:n+1)-f(2:n))./h(2:n) ...
    -6*(f(2:n)-f(1:n-1))./h(1:n-1);
bet=ones(1,n+1);
delta=zeros(1,n+1);
for i=2:n;
    bet(i)=b(i)-a(i)*c(i-1)/bet(i-1);
    delta(i)=k(i)-a(i)*delta(i-1)/bet(i-1);
end
Gll=zeros(1,n+1);
for i=n:-1:1;
    Gll(i)=(delta(i)-c(i)*Gll(i+1))/beta(i);
end
B(1:n)=Gll(1:n)/2;
A(1:n)=(Gll(2:n+1)-Gll(1:n))./(6*h(1:n));
C(1:n)=(f(2:n+1)-f(1:n))./h(1:n) ...
    -h(1:n).*(Gll(2:n+1)+2*Gll(1:n))/6;
G=zeros(1,N);
for i=1:n;
    lista=find(X>=x(i) & X <= x(i+1));
    G(lista)=A(i)*(X(lista)-x(i)).^3+ ...
        B(i)*(X(lista)-x(i)).^2+C(i)*(X(lista)-x(i))+ f(i);
```

```

end
plot(x,f,'ro',X,G)
endfunction

```

Rotina SCILAB: A “function” “smooth(F, DX)” executa a interpolação tipo spline cúbico. O argumento F é uma matriz $2 \times n$, definida por $F = [x; f]$, e DX é o intervalo entre dois valores consecutivos da nova abscissa, $DX = X_{j+1} - X_j$. O resultado de smooth é uma matriz $2 \times N$, cuja primeira linha contém os N valores da nova abscissa e a segunda linha os valores da ordenada. As condições nos extremos não são as mesmas usadas em nosso programa “cubico”, e por isso os resultados obtidos com “smooth” e com “cubico” podem ser diferentes nos segmentos próximos das extremidades.

Exercício 1.5:

Produza no workspace um conjunto de pontos (x_i, f_i) onde $f_i = f(x_i)$, sendo $f(x)$ uma função com bastante oscilação (por exemplo, escolha para f um conjunto de valores aleatórios entre 0 e 1) Para diferentes conjuntos de x_i , e escolhendo um conjunto de X_j , com muito mais pontos que os x_i , executa “cubico(x,f,X)” e “smooth(F, DX)”, com $F = [x, f]$ e $DX = X_{j+1} - X_j$ e compare os resultados num mesmo gráfico. Qual das linhas de interpolação te parece preferível?

1.2.3 Ajuste

Dados experimentais geralmente correspondem a um conjunto de pares de números, (x_i, m_i) , $i = 1, 2, \dots, N$, onde x_i são valores de alguma variável regulável (temperatura, pressão, campo aplicado, etc.) e m_i os resultados obtidos nas medidas correspondentes de alguma variável dependente de x (dilatação térmica, compressibilidade, suscetibilidade magnética, condutividade elétrica, etc.). Usualmente o físico experimental é capaz de estimar o erro Δm_i de cada medida, de maneira que se costuma escrever o resultado de uma medida na forma $y_i = m_i \pm \Delta m_i$, sendo y o nome da variável que se está medindo e $y_i = y(x_i)$. Na maioria dos casos há alguma razão teórica para se esperar que $y(x)$ seja uma função de forma conhecida, dependente de

alguns parâmetros de valores desconhecidos. Por exemplo, a potência dissipada numa experiência de ressonância, em função da frequência ω do campo aplicado, pode ter a forma de uma função Lorentziana,

$$P(\omega) = \frac{A}{(\omega - \omega_0)^2 + \Gamma^2}, \quad (1.39)$$

onde a amplitude A , a frequência de ressonância ω_0 e a largura Γ são os parâmetros de valores desconhecidos.

Vamos introduzir o “vetor de parâmetros” $q = (q_1, q_2, \dots, q_n)$ e chamar a função a ser usada para interpolar os resultados experimentais de $y(x; q)$. Neste caso a palavra interpolação não tem exatamente o mesmo significado que tinha nas seções anteriores. Naquelas a linha interpoladora passava sobre os pontos dados, mas aqui ela passa “o mais próximo possível” dos pontos dados. Com efeito, se temos N pontos e $n < N$ parâmetros ajustáveis, em geral não é possível encontrar valores dos parâmetros para os quais a $y(x; q_1, q_2, \dots, q_n)$ passe pelos N pontos. Mas o que significa “o mais próximo possível”? Para responder a esta pergunta vamos introduzir o conceito “desvio quadrático”, χ^2 (pronuncia-se quiquadrado):

$$\chi^2 = \sum_{i=1}^N \left(\frac{y(x_i; q) - m_i}{\Delta m_i} \right)^2. \quad (1.40)$$

A “linha de ajuste” $y(x; q)$ passa o mais próximo possível dos pontos experimentais (x_i, m_i) quando χ^2 é mínimo em relação a variações dos parâmetros q . Encontrar o melhor ajuste significa encontrar os valores q_1, q_2, \dots, q_n que minimizam χ^2 .

Sabemos do Cálculo que um mínimo de uma função de várias variáveis ocorre em pontos onde as derivadas parciais da função são nulas. Definindo

$$f_j(q) = \frac{1}{2} \frac{\partial \chi^2}{\partial q_j} \quad j = 1, 2, \dots, n \quad (1.41)$$

e definindo o “vetor”

$$f(q) = [f_1(q_1, \dots, q_n); f_2(q_1, \dots, q_n); \dots; f_n(q_1, \dots, q_n)],$$

o problema se reduz a encontrar raízes (geralmente uma única) de $f(q)$. O fator $1/2$ na definição de f foi introduzido somente para simplificar o fator 2 que surge quando se deriva χ^2 .

Muitas vezes os erros Δm_i são considerados iguais, digamos Δm . Então o fator $(1/\Delta m)^2$, na Eq.(1.40), é apenas um fator constante. Como os pontos de mínimos de uma função não dependem de um fator constante, nada se perde se definirmos, neste caso, χ^2 simplesmente por

$$\chi^2 = \sum_{i=1}^N (y(x_i; q) - m_i)^2 . \quad (1.42)$$

O mesmo procedimento é usado quando os erros das medidas não são especificados.

Por simplicidade vamos apresentar o procedimento para a obtenção do mínimo de χ^2 como definido pela Eq.(1.42). A generalização para o caso da Eq.(1.40) é trivial, bastando acrescentar os fatores $1/\Delta m_i^2$ aos termos da f e da matriz Jacobiana.

Ajuste Linear

Antes de apresentarmos o procedimento geral para ajustar uma curva qualquer a um conjunto de pontos dados, vamos ver um caso particular, muito simples mas de grande importância, o *ajuste linear*, também conhecido por *regressão linear* por razões históricas. Neste caso a curva a ser ajustada aos pontos é uma reta,

$$y(x) = a + b x , \quad (1.43)$$

O problema consiste em encontrar a e b que minimizem

$$\chi^2 = \sum_{i=1}^N (a + b x_i - m_i)^2 . \quad (1.44)$$

Isto se obtém igualando a zero as derivadas de χ^2 em relação aos parâmetros a e b :

$$\frac{\partial \chi^2}{\partial a} = 2 \sum_{i=1}^N (a + b x_i - m_i) = 0 \quad (1.45)$$

28 CAPÍTULO 1. INTRODUÇÃO AOS MÉTODOS NUMÉRICOS

e

$$\frac{\partial \chi^2}{\partial b} = 2 \sum_{i=1}^N (a + b x_i - m_i) x_i = 0 \quad (1.46)$$

Definindo

$$\begin{aligned} A &= \sum_{i=1}^N x_i \\ B &= \sum_{i=1}^N m_i \\ C &= \sum_{i=1}^N (x_i)^2 \\ D &= \sum_{i=1}^N m_i x_i , \end{aligned} \quad (1.47)$$

o sistema de equações Eq.(1.45) e Eq.(1.46) fica

$$\begin{aligned} N a + A b &= B \\ A a + C b &= D \end{aligned} \quad (1.48)$$

A solução deste sistema nas incógnitas a e b é

$$a = \frac{B C - A D}{N C - A^2} \quad (1.49)$$

e

$$b = \frac{N D - A B}{N C - A^2} . \quad (1.50)$$

Exercício 1.6: Gerar um conjunto de N pontos aleatórios, próximos a uma reta. Por ex., em SCILAB você pode escrever
 $x=0:10;$
 $m=2-0.5*x+2*rand(1,11);$
 Encontre a e b e faça um gráfico que contenha os pontos gerados e a reta de ajuste.

Ajuste a uma curva qualquer

Derivando a Eq.(1.42) em relação a q_j obtemos (cf. Eq.(1.41))

$$f_j = \sum_{i=1}^N (y(x_i; q) - m_i) y_j(x_i), \quad j = 1, \dots, n, \quad (1.51)$$

onde

$$y_j(x_i) = \frac{\partial y(x_i; q)}{\partial q_j}. \quad (1.52)$$

Para encontrarmos uma raiz de $f(q)$ precisaremos da matriz Jacobiana (veja seção 1.1.3.2), de elementos

$$J_{jk} = \frac{\partial f_j(q)}{\partial q_k}. \quad (1.53)$$

Como as f_j contém derivadas primeiras de y , os elementos J_{jk} conterão derivadas segundas,

$$y_{jk}(x_i) = \frac{\partial^2 y(x_i; q)}{\partial q_j \partial q_k} = y_{kj}(x_i). \quad (1.54)$$

Portanto

$$J_{jk} = \sum_{i=1}^N \{y_j(x_i)y_k(x_i) + (y(x_i) - m_i)y_{jk}(x_i)\}. \quad (1.55)$$

Agora basta escolher um ponto $q^{(1)} = q_1, \dots, q_n$, razoavelmente próximo do que esperamos ser raiz de $f(q)$, ou seja, que produz uma curva $y(x; q^{(1)})$ próxima dos pontos experimentais (x_i, m_i) , e seguir a “receita” da seção 1.1.4, com $x^{(1)} = q^{(1)}$.

Exemplo 1.3:

Suponhamos que os dados da tabela seguinte representem valores medidos da potência dissipada, $P(\omega)$, para um conjunto de valores da frequência, $\omega = \omega_1, \dots, \omega_n$. A figura 1.4 é o gráfico dos pontos da tabela 1 juntamente com a linha Lorentziana ajustada. Vamos usar o procedimento acima para encontrar os parâmetros A , ω_0 e Γ da Eq.(1.39) que produzem o melhor ajuste para os pontos dados.

ω_i	$m_i = P(\omega_i)$	ω_i	$m_i = P(\omega_i)$
0.0	1.0	5.5	11.0
1.0	1.5	6.0	7.0
2.0	3.0	7.0	3.0
3.0	4.5	8.0	2.0
3.5	8.0	9.0	0.7
4.0	12.0	10.0	1.0
4.5	16.0	11.0	0.5
5.0	15.5	12.0	0.0

Tabela 1: Pontos representados na figura 1.4.

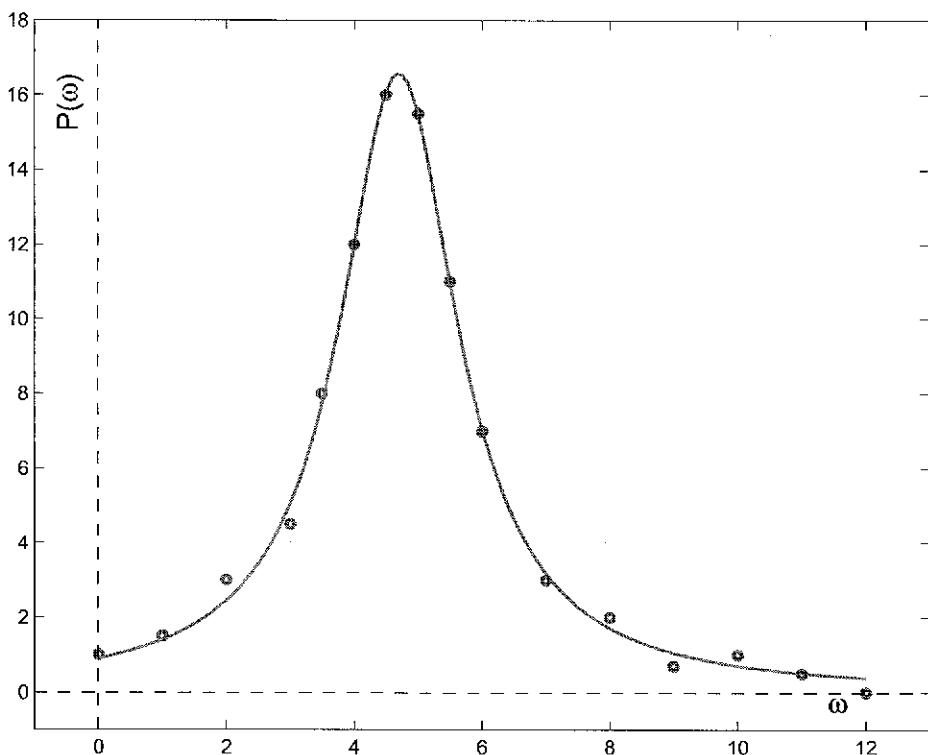


Figura 1.4: Pontos medidos em uma experiência de ressonância

O programa “lorentz.sci” é uma função que executa o procedimento descrito acima. Olhando para a figura 1.4 tem-se uma ideia

aproximada dos valores dos parâmetros. Por exemplo, podemos escolher, como valores iniciais, $\omega_0 = 5$, $\Gamma = 1$ e $A = 16$ (lembre que A/Γ^2 é o valor do máximo). Como tolerância (tol) podemos escolher, por exemplo, 10^{-8} . O programa apresenta inicialmente um gráfico da Lorentziana obtida com estes parâmetros e permite que o operador altere os mesmos, tantas vezes quantas for preciso para obter uma curva que esteja bastante próxima dos pontos experimentais. Se o programa for rodado sem este cuidado é bem possível que não converja para os valores corretos. O programa apresenta uma sequência de gráficos para o operador poder seguir a convergência ao ajuste ideal. Após cada gráfico apresentado, dê um “enter” para continuar. No final são apresentados os parâmetros do melhor ajuste, o valor de χ^2 e o “erro”, que é a diferença entre o penúltimo e o último χ^2 . Quanto menor for χ^2 melhor é o ajuste.

Programa: lorentz.sci

```

function L=lorentz(w,m)
// Quiquadrado para Lorentziana. No workspace deve
// haver o conjunto de pontos (w,m)=(w(j),m(j)), j=1:N;
// w e m devem ser vetores fila, 1 x N;
plot(w,m,'o')
par=input('deh ponto inicial e tol [A,w0,gama,tol] \n');
A=par(1); w0=par(2); gama2=par(3)^2; tol=par(4);
q=[A;w0;gama2]; // vetor coluna dos parâmetros;
lw=length(w);
ww=linspace(0,w(lw),100);
while(length(par)>0);
    Pw=A*ones(ww)./((ww-w0).^2+gama2);
    clf; plot(w,m,'.',ww,Pw)
    par=input('troca parametros [A,w0,gama]? \n');
    if(length(par)==3)
        A=par(1); w0=par(2); gama2=par(3)^2;
        q=[A;w0;gama2]; // vetor coluna dos parâmetros;
    end
end
dw=w-w0; dw2=dw.^2;
D=ones(w)./(dw2+gama2);

```

32 CAPÍTULO 1. INTRODUÇÃO AOS MÉTODOS NUMÉRICOS

```
D2=D.^2; D3=D2.*D;
P=A*D; Pm=P-m;
xi2=Pm*Pm';
for n=1:10;
// definições
P1=D'; P2=(2*A*dw.*D2)';
P3=(-A*D2)';
// calcula f=[f1;f2;f3] e testa;
f1=Pm*P1; f2=Pm*P2; f3=Pm*P3;
f=[f1;f2;f3];
// calcula a matriz J
P12=(2*dw.*D2)';
P13=(-D2)';
P22=(-2*A*D2+8*A*(dw.^2).*D3)';
P23=(-4*A*dw.*D3)';
P33=(2*A*D3)';
// Matriz Jacobiana
J11=P1'*P1;
J12=P1'*P2+Pm*P12;
J13=P1'*P3+Pm*P13;
J22=P2'*P2+Pm*P22;
J23=P2'*P3+Pm*P23;
J33=P3'*P3+Pm*P33;
J=[J11, J12, J13; J12, J22, J23; J13, J23, J33];
// cálculo do vetor deslocamento s e do novo q:
s=-J\f;
q=q+s;
A=q(1);
w0=q(2);
gama2=q(3);
clf
Pw=A*ones(ww)./((ww-w0).^2+gama2);
plot(w,m,'.',ww,Pw)
halt()
dw=w-w0; dw2=dw.^2;
D=ones(w)./(dw2+gama2);
D2=D.^2; D3=D2.*D;
```

```

P=A*D; Pm=P-m;
chi2=Pm*Pm';
erro=xi2-chi2;
disp(erro, 'erro=')
if(erro < tol ); break; disp(n, 'n='), end
xi2=chi2;
end
gama=sqrt(gama2);
P=A*ones(w)./((w-w0).^2+gama2);
disp(['      A   ', ' w0   ', ...
      ' Gama   ', ' chi2   ', ' erro   '])
disp([A, w0, gama, chi2,erro])
L=[ww;Pw];
endfunction

```

Exercício 1.7:

- a) Execute o programa “lorentz.sci” e encontre o melhor ajuste para os pontos da tabela 1.
- b) Escreva um programa “Gaussiana.sci” para ajustar pontos experimentais a uma função Gaussiana,

$$P_G(\omega) = A \exp\left(\frac{-(\omega - \omega_0)^2}{2\Gamma^2}\right) . \quad (1.56)$$

É claro que antes de escrever o programa você deve obter, analiticamente, os elementos da matriz Jacobiana. Aplicando seu programa aos dados da tabela 1 você obterá um ajuste não muito satisfatório (χ^2 grande), o que mostra que estes dados experimentais não correspondem a uma Gaussiana.

- c) Gere um novo conjunto de valores $\{m_i\}$ usando a própria função Gaussiana, Eq.(1.56), para um conjunto de frequências $\{\omega_i\}$. Para este par de vetores (ω, m) seu programa “Gaussiana.sci” deve produzir um ajuste perfeito, com $\chi^2 = 0$.

1.3 Integração Numérica

Consideremos uma integral definida, de uma função escalar $f(x)$ da variável real x . Seja (a, b) o intervalo de integração, no qual definimos um conjunto de pontos, $x_1 = a$, $x_2 = x_1 + \Delta x_1$, $x_3 = x_2 + \Delta x_2$, ..., $x_{i+1} = x_i + \Delta x_i$, ..., $x_n = b$. Por definição

$$I_{ab} = \int_a^b f(x)dx = \lim_{\{\Delta x_i \rightarrow 0\}} \sum_{i=1}^n f(x_i)\Delta x_i . \quad (1.57)$$

Quando se conhece uma função $F(x)$ tal que $F'(x) = f(x)$, então, pelo teorema fundamental do cálculo, sabemos que

$$I_{ab} = F(b) - F(a) . \quad (1.58)$$

O uso de integração numérica é feito quando não se conhece a $F(x)$. A Eq.(1.57) é então calculada, não no limite $\Delta x_i \rightarrow 0$, mas para Δx_i “suficientemente pequenos”. Mas que significa “suficientemente pequenos”? Significa que não faz diferença relevante no valor calculado para a integral se eles forem ainda menores. No caso de funções $f(x)$ “bem comportadas” não há por que não escolher os $\Delta x_i = \Delta x$, ou seja, todos iguais. Com esta escolha a integral, Eq.(1.57), será

$$I_{ab} = \Delta x \sum_{i=1}^n f_i . \quad (1.59)$$

onde $f_i = f(x_i)$.

Geometricamente esta fórmula corresponde a construir-se um retângulo de altura f_i e largura Δx em torno de cada x_i e aproximar a área entre a curva $f(x)$ e a abscissa, x , como a soma das áreas dos retângulos.

1.3.1 Regra do Trapézio

Uma aproximação melhor se obtém usando, em vez de retângulos, trapézios, como na figura 1.5.

Isto corresponde a fazer-se uma interpolação linear entre os pontos (x_i, f_i) consecutivos. A área do i^{mo} trapézio é $0.5(f_i + f_{i+1})\Delta x$. Somando-se as áreas de todos os trapézios se obtém

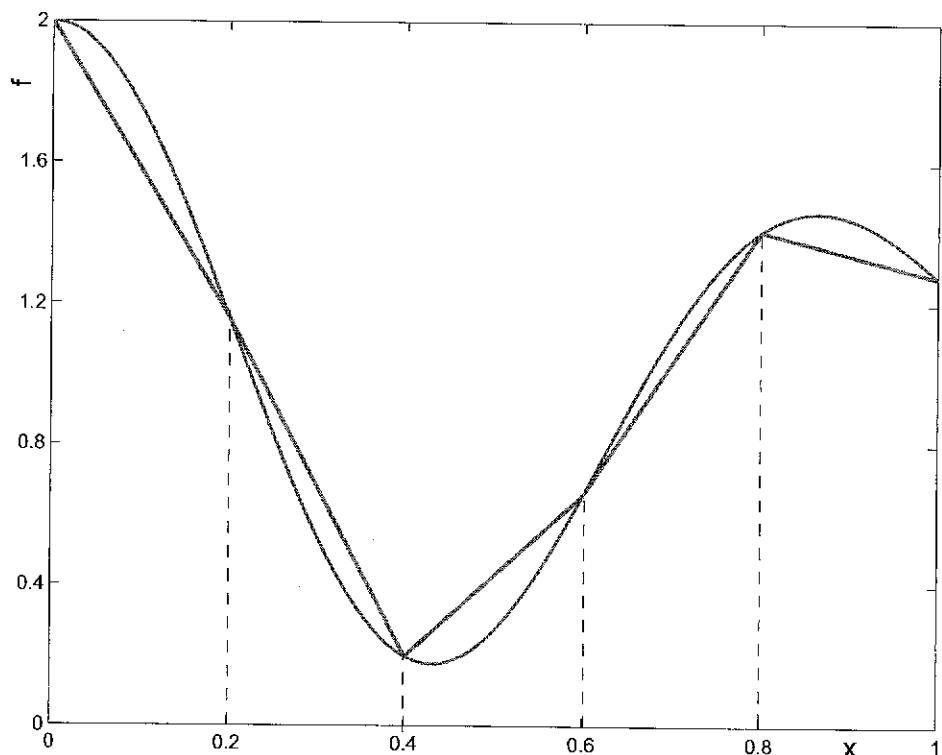


Figura 1.5: Integração pela regra do trapézio

$$I_{ab} = \Delta x \left(\frac{1}{2}f_1 + f_2 + f_3 + \dots + f_{n-1} + \frac{1}{2}f_n \right) = \Delta x \left(\sum_{i=1}^n f_i - \frac{1}{2}(f_1 + f_n) \right) . \quad (1.60)$$

Note que o resultado, neste caso em que escolhemos os Δx_i todos iguais, é muito semelhante ao da Eq.(1.59), diferindo daquele apenas pela subtração de metade das áreas dos retângulos nas extremidades.

Como exemplo vamos calcular a seguinte integral, que corresponde à função representada pelo gráfico da figura 1.5,

$$I_{01} = \int_0^1 \exp(-x^2) \cos(7x) dx . \quad (1.61)$$

Na figura 1.5 dividimos o intervalo $(0, 1)$ em apenas 5 segmentos,

para destacar a diferença entre o gráfico da função e as linhas retas dos trapézios. É claro que com uma divisão tão grosseira não esperamos encontrar um valor muito preciso para a integral. Repetimos o cálculo para diferentes números, n , de segmentos e mostramos os resultados na tabela abaixo.

n segmentos	integral
5	1.0131946
10	1.0191555
20	1.0205801
50	1.0209746
100	1.0210307
200	1.0210448
500	1.0210487
1000	1.0210493

Observamos da tabela que, à medida que o número de segmentos cresce, os resultados obtidos para a integral vão ficando cada vez mais próximos, sendo que a diferença entre os valores obtidos com 500 e 1000 segmentos aparece apenas no oitavo algarismo significativo.

1.3.2 Outras Aproximações para Integração Numérica

Se desejamos uma fórmula mais precisa, ou seja, com o mesmo número de divisões do intervalo de integração queremos um resultado mais preciso para a integral, podemos proceder de diversas maneiras. O primeiro passo pode ser interpolar cada três pontos consecutivos com um polinômio de segundo grau, usando, por exemplo, o polinômio interpolador de Lagrange, Eqs. (1.31) e (1.32). O número $n + 1$ de pontos escolhidos, incluindo os extremos, deve ser ímpar. No caso de escolhermos $\Delta x_i = \Delta x$ (i.e., todos iguais) a integração do polinômio entre os pontos x_{2k-1} e x_{2k+1} resulta em

$$I_k = \frac{\Delta x}{3} (f_{2k-1} + 4f_{2k} + f_{2k+1}) \quad (1.62)$$

e a integral global é, então,

$$I_{ab} = \sum_{k=1}^{n/2} I_k = \frac{\Delta x}{3} (f_1 + 4f_2 + 2f_3 + 4f_4 + \cdots + 2f_{n-1} + 4f_n + f_{n+1}) . \quad (1.63)$$

Esta fórmula é conhecida como “Regra de Simpson”.

São conhecidas ainda várias outras “receitas” para integração numérica de funções escalares de uma variável real, que podem ser encontradas, por exemplo, nos livros de cálculo numérico, listados nas Referências Bibliográficas. Como os computadores modernos são muito rápidos e possuem memória em abundância, geralmente não paga a pena preocupar-se com algoritmos mais precisos. O mais prático é dividir o intervalo de integração em um número suficientemente grande de segmentos de modo que a regra do trapézio já seja suficientemente precisa.

1.3.3 Integrais Impróprias

Uma integral definida é “imprópria” se um ou ambos os limites de integração são $\pm\infty$. Seja, por exemplo,

$$I = \int_0^\infty f(x)dx . \quad (1.64)$$

Em geral a integral acima pode existir ou não existir. É condição necessária (não é suficiente) para que ela exista que $f(x) \rightarrow 0$ quando $x \rightarrow \infty$.

A maneira mais simples, mas que nem sempre funciona bem, de se calcular, numericamente, uma integral desse tipo é substituí-la pela integral definida

$$I_X = \int_0^X f(x)dx \quad (1.65)$$

e escolher X “suficientemente grande”, i.e., tão grande que se ele for ainda maior não haverá alteração relevante no valor da integral.

Em muitas circunstâncias é bastante difícil decidir a partir de que valor X é “suficientemente grande”. Neste caso convém buscar-se

uma transformação de variável que transforme a integral imprópria numa integral própria. Uma maneira que funciona em muitos casos consiste em dividir a integral em duas,

$$I = \int_0^a f(x)dx + \int_a^\infty f(x)dx . \quad (1.66)$$

A primeira das integrais acima pode, em geral, ser resolvida pelos métodos usuais e na segunda efetuamos uma transformação de variáveis. Por exemplo,

$$y = 1/x \quad dx = -(1/y^2)dy \quad \text{e}$$

$$I_2 = \int_a^\infty f(x)dx = \int_0^{1/a} \frac{f(1/y)dy}{y^2} . \quad (1.67)$$

Se $f(x) \rightarrow 0$ quando $x \rightarrow \infty$ pelo menos tão rápido como $1/x^2$, então $f(1/y)/y^2$ não diverge em $y = 0$ e a integral I_2 é bem definida e pode ser calculada numericamente.

Exemplo 1.4:

$$I = \int_0^\infty \frac{dx}{1+x^2} = \frac{\pi}{2} . \quad (1.68)$$

Vamos desconsiderar que conhecemos o valor desta integral e ver como a resolveríamos numericamente. Inicialmente vamos dividi-la em duas: $I = I_1 + I_2$, onde

$$I_1 = \int_0^1 \frac{dx}{1+x^2} \quad (1.69)$$

e

$$I_2 = \int_1^\infty \frac{dx}{1+x^2} . \quad (1.70)$$

Na I_2 fazemos a transformação mencionada acima: $x = 1/y$,

$$I_2 = \int_0^1 \frac{dy}{y^2(1+\frac{1}{y^2})} = \int_0^1 \frac{dy}{1+y^2} = I_1 . \quad (1.71)$$

Portanto $I = 2I_1$, que pode ser resolvida numericamente com facilidade.

1.4 Transformada de Fourier

A transformação de Fourier tem sido usada não somente como uma poderosa ferramenta matemática, que facilita, por exemplo, a obtenção de soluções de equações diferenciais, mas também como uma importante técnica auxiliar na interpretação de resultados experimentais. Mais importante ainda, ela serve de caminho para a colaboração entre teoria e experiência, em diversas áreas da Física. Só para citar um exemplo, o físico teórico pode produzir uma função matemática, chamada “função resposta”, que prevê o deslocamento de uma variável dinâmica A de seu valor de equilíbrio, um tempo t após a aplicação de uma perturbação em forma de pulso. Para o físico experimental é mais simples aplicar a perturbação em forma de um campo periódico, de frequência ν e observar a suscetibilidade de A a este campo. Pode-se mostrar (veja no Capítulo VII a seção sobre Teoria da Resposta Linear) que a suscetibilidade é a transformada de Fourier da função resposta.

No que segue nós usaremos a letra t para indicar a variável independente da função $f(t)$, mas t não precisa indicar o tempo, sendo também muito usadas transformadas de Fourier no espaço. Para a variável independente da transformada usaremos a “frequência angular” $\omega = 2\pi\nu$. No caso de t ser uma variável espacial, ω significa o “número de onda” $k = 2\pi/\lambda$, onde λ é um “comprimento de onda”. Por simplicidade chamaremos ω de frequência.

Seja $f(t)$ uma função tal que a integral

$$g(\omega) = \int_{-\infty}^{+\infty} \exp(-i\omega t) f(t) dt \quad (1.72)$$

exista. Neste caso a função $g(\omega)$ se chama *transformada de Fourier* de $f(t)$.

Há uma correspondência biunívoca entre as funções $f(t)$ e suas transformadas $g(\omega)$. A relação

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \exp(+i\omega t) g(\omega) d\omega \quad (1.73)$$

se chama *transformada de Fourier inversa*. O fator $1/2\pi$ pode ser posto na Eq.(1.72) em lugar de incluí-lo na Eq.(1.73) ou ainda, como

é bastante usual, pode-se por o fator $1/\sqrt{2\pi}$ em ambas as equações. Devido à relação

$$\exp(\pm i\omega t) = \cos(\omega t) \pm i \sin(\omega t) , \quad (1.74)$$

vemos que a transformada de Fourier de uma função real $f(t)$ é uma função complexa $g(\omega)$ cuja parte real é uma função par de ω e cuja parte imaginária é uma função ímpar de ω .

1.4.1 Propriedades das Transformadas de Fourier

Vamos introduzir a notação $\mathcal{F}[f(t)]$ para indicar a transformada de Fourier de $f(t)$. Na notação acima, $\mathcal{F}[f(t)] = g(\omega)$. Semelhantemente $\mathcal{F}^{-1}[g(\omega)] = f(t)$. A seguir listamos, sem demonstração, uma série de propriedades das transformadas de Fourier. Para ver as demonstrações o leitor pode consultar, por exemplo, as referências [4] ou [5]. Algumas das propriedades listadas são consequências imediatas da própria definição, Eq.(1.72).

1) Paridade

- a) Se $f(t)$ é real, então

$$Re[g(-\omega)] = Re[g(\omega)] \quad \text{e} \quad Im[g(-\omega)] = -Im[g(\omega)] ,$$

ou seja,

$$g(-\omega) = g^*(\omega) ;$$

- b) Se $f(t)$ é real e par, então $g(\omega)$ é real;
c) Se $f(t)$ é real e ímpar, então $g(\omega)$ é imaginária;
d) Se $f(t)$ é imaginária, então as propriedades a, b, c, acima, serão válidas se trocarmos “real” por “imaginária” e vice versa.

2) Linearidade

$$\mathcal{F}[a f_1(t) + b f_2(t)] = a \mathcal{F}[f_1(t)] + b \mathcal{F}[f_2(t)] = a g_1(\omega) + b g_2(\omega) ,$$

onde a e b são constantes.

3) Inversão Temporal

$$\mathcal{F}[f(-t)] = g(-\omega) = g^*(\omega) ,$$

onde a última igualdade só é válida se $f(t)$ é real.

4) Relação de Escala

$$\mathcal{F}[f(at)] = \frac{1}{|a|} g\left(\frac{\omega}{a}\right)$$

onde $g(\omega) = \mathcal{F}[f(t)]$.

5) Deslocamento da Origem

$$\mathcal{F}[f(t - t_0)] = \exp(-i\omega t_0) g(\omega) .$$

Semelhantemente,

$$\mathcal{F}^{-1}[g(\omega - \omega_0)] = \exp(i\omega_0 t) f(t) .$$

6) Transformada da Derivada

$$\mathcal{F}\left[\frac{df}{dt}\right] = i\omega g(\omega) .$$

7) Transformada da Convolução

A *Convolução* da função $f_1(t)$ com a $f_2(t)$ é definida como a seguinte função de t ,

$$f_1 \otimes f_2 = \int_{-\infty}^{\infty} f_1(\tau) f_2(t - \tau) d\tau ,$$

e sua transformada de Fourier é

$$\mathcal{F}[f_1 \otimes f_2] = \mathcal{F}[f_1] \mathcal{F}[f_2] = g_1(\omega) g_2(\omega) .$$

8) Transformada da Correlação

A *Correlação* da função $f_1(t)$ com a $f_2(t)$ é definida como a seguinte função de t ,

$$\text{Corr}(f_1, f_2) = \int_{-\infty}^{\infty} f_1(\tau) f_2(t + \tau) d\tau .$$

Esta definição parece muito semelhante à definição de convolução, mas note que aqui a variável de integração, τ , comparece nos argumentos de f_1 e f_2 com o mesmo sinal, enquanto que na convolução os sinais de τ são trocados. Isto é uma diferença importante. A transformada de Fourier da correlação não é o produto das transformadas, mas, levando em conta a propriedade 2 se obtém

$$\mathcal{F}[\text{Corr}(f_1, f_2)] = g_1(-\omega) g_2(\omega) = g_1^*(\omega) g_2(\omega) .$$

Nas duas equações acima estamos supondo que as $f(t)$ são reais. Se forem complexas a correlação é definida com f_1^* em vez de f_1 e a última relação não é válida.

9) Densidade espectral

Se $f(t)$ for o campo elétrico de um pacote de onda eletromagnética, então a medida da intensidade que passa por um filtro de frequência, sintonizado na frequência ω , é proporcional à transformada de Fourier da função de autocorrelação de $f(t)$ (Teorema de Wiener-Khinchin)

$$S(\omega) = \mathcal{F}[\text{Corr}(f, f)] = |g(\omega)|^2 . \quad (1.75)$$

A $S(\omega)$ se chama *densidade espectral*. A seguinte relação é conhecida como *Teorema de Perseval*,

$$\int_{-\infty}^{\infty} dt |f(t)|^2 = \int_{-\infty}^{\infty} \frac{d\omega}{2\pi} |g(\omega)|^2 , \quad (1.76)$$

ou seja, a integral temporal da potência elétrica do pacote é igual à integral na frequência da densidade espectral, o que significa que ambos os membros desta relação são a energia total contida no pacote de onda.

1.4.2 A Delta de Dirac

A “função” $\delta(x - x_0)$, conhecida como a *delta de Dirac*, não é propriamente uma função. Ela é classificada como “distribuição”, ou “função generalizada”. Ela só tem um significado quando comparece no integrando de uma integral definida. Sua definição é: Seja $f(x)$ uma função qualquer da variável real x e $[a, b]$ um intervalo qualquer; então

$$\int_a^b f(x)\delta(x - x_0)dx = \begin{cases} f(x_0) & \text{se } a < x_0 < b \\ 0 & \text{se } x_0 \notin [a, b] \\ \frac{1}{2}f(x_0) & \text{se } x_0 = a \text{ ou } x_0 = b \end{cases} \quad (1.77)$$

Uma forma alternativa de definir a δ , embora menos rigorosa, permite uma visão mais pictórica de seu significado. Consiste em enunciar suas propriedades a, b, c, como segue:

- a) $\delta(x - x_0) = 0$ para todo $x \neq x_0$;
- b) $\delta(x - x_0) = \infty$ para $x = x_0$ de tal forma que

$$\int_{x_0-\epsilon}^{x_0+\epsilon} \delta(x - x_0)dx = 1 ;$$

- c) $\delta(-x) = \delta(x) ;$

Frente à transformada de Fourier é imediato verificar as seguintes propriedades:

- d) $\mathcal{F}[\delta(t - t_0)] = \exp(-i\omega t_0) ;$ em particular, $\mathcal{F}[\delta(t)] = 1 ;$
- e) Segue de d que a transformada inversa de 1 é a delta, ou seja,

$$\int_{-\infty}^{\infty} \exp(i\omega t)d\omega = 2\pi\delta(t) ; \quad (1.78)$$

devido à paridade da delta, propriedade c, é claro que a Eq.(1.78) continua válida se escrevermos $\exp(-i\omega t)$ em lugar de $\exp(i\omega t)$; equivalentemente,

$$\int_{-\infty}^{\infty} \exp(\pm i\omega t) dt = 2\pi\delta(\omega); \quad (1.79)$$

f) Se $f(x)$ é uma função real que possui raízes simples nos pontos $x_1, x_2, \dots, x_j, \dots$, ou seja, $f(x_j) = 0$, então

$$\delta(f(x)) = \sum_j \frac{\delta(x - x_j)}{\left| \frac{df}{dx} \right|}; \quad (1.80)$$

g) Se \mathbf{x} é uma variável vetorial, tri-dimensional, de componentes cartesianas x_1, x_2, x_3 , então

$$\delta(\mathbf{x} - \mathbf{x}_0) = \delta(x_1 - x_{01})\delta(x_2 - x_{02})\delta(x_3 - x_{03}) \quad (1.81)$$

e

$$\int_V f(\mathbf{x})\delta(\mathbf{x} - \mathbf{x}_0)d^3x = \begin{cases} f(\mathbf{x}_0) & \text{se } \mathbf{x}_0 \in V \\ 0 & \text{se } \mathbf{x}_0 \notin V \end{cases} \quad (1.82)$$

Se \mathbf{x} for bi-dimensional ou n -dimensional, então valem relações equivalentes a estas, mutatis-mutandis.

1.4.3 Transformada de Fourier Discreta

Vamos tratar explicitamente de funções $f(t)$ de suporte finito $[a, b]$, ou seja $f(t) = 0$ se $t \notin [a, b]$. A integral na definição da transformada $g(\omega)$, Eq.(1.72), é facilmente discretizada pela “regra do trapézio”, cf. seção 1.3.1, Eq.(1.60). Supomos que o número n de termos da integral discretizada é suficientemente grande para que possamos desprezar a correção dos extremos, $-\frac{1}{2}(f_1 + f_n)$. Escrevemos, portanto,

$$g(\omega_k) = \Delta t \sum_{j=1}^n \exp(-i\omega_k t_j) f_j, \quad (1.83)$$

onde $f_j = f(t_j)$ e discretizamos também a frequência ω . Por ora vamos esquecer o fator constante Δt e no final do processo invocaremos a Eq.(1.76) para definir a constante multiplicativa da $g(\omega_k)$. Uma pergunta imediata é: quais devem ser os valores de ω_k ? Como é desejável manter a correspondência biunívoca $f \Leftrightarrow g$, devemos ter

o mesmo número, n , de ω_k que temos de t_j . Para se caracterizar uma variação na função $f(t)$ é preciso deslocar no mínimo um ponto f_j de seu valor original. Tal deslocamento afeta a $f(t)$ num intervalo de tempo $\tau = 2\Delta t$. Este é, portanto, o mínimo período de oscilação da $f(t)$ que pode ser “visto” na $g(\omega)$. A máxima frequência relevante é, portanto, $\nu = 1/(2\Delta t)$, ou seja, $\omega_n = 2\pi\nu = \pi/\Delta t$. Mas também as frequências negativas, com módulo até $\pi/\Delta t$ devem ser consideradas. Um pequeno detalhe é que, se escolhermos $\omega_1 = -\pi/\Delta t$ e $\omega_n = \pi/\Delta t$, então $g(\omega_1) = g(\omega_n)$, pois os t_j são números inteiros de Δt e, portanto, $\pm(\pi/\Delta t)t_j$ diferem por números inteiros de 2π e $\exp(-i(\pi/\Delta t)t_j) = \exp(+i(\pi/\Delta t)t_j)$. Para que $g(\omega_k)$ seja periódica em k , precisamos ter $\omega_{n+k} = \omega_k + 2\pi$. É usual escolher

$$\Delta\omega = \frac{2\pi}{n} \quad (1.84)$$

e

$$\omega_1 = \frac{-\pi}{\Delta t} \quad \omega_{k+1} = \omega_1 + k\Delta\omega \quad \omega_n = \frac{\pi}{\Delta t} - \Delta\omega. \quad (1.85)$$

Esta escolha das frequências é também usada no procedimento chamado "Transformada de Fourier Rápida", que veremos mais adiante. Entretanto, em muitos casos esta escolha se torna inconveniente, concentrando a região de frequências mais relevantes em um intervalo muito pequeno, tornando os gráficos de $\text{Re}(g(\omega))$ e $\text{Im}(g(\omega))$ difíceis de serem analisados.

Alternativamente, podemos programar a transformada de Fourier discreta de maneira a permitir ao usuário que escolha o intervalo de frequências e o número delas arbitrariamente, de maneira a produzir gráficos convenientes. É este o procedimento que usamos no programa abaixo, uma "function", `tfd(t,f)`.

Vamos re-escrever a Eq.(1.83) em forma matricial. Para isto definimos a matriz retangular W , de elementos

$$W_{kj} = \exp(-i\omega_k t_j) \quad (1.86)$$

e os vetores coluna f , de elementos f_j e g , de elementos g_k . Com isto a Eq.(1.83) adquire a forma

$$g = W * f \quad (1.87)$$

onde o sinal * indica produto matricial.

A operacionalização da transformada de Fourier discreta é feita no programa tipo “function”, tfd.sci, que apresentamos a seguir:

Programa: tfd.sci

```

function G=tfd(t,f)
// Transformada de Fourier Discreta;
// Permite escolha das frequências discretas a serem
// usadas na transformada de Fourier. No workspace deve
// haver o par de vetores (t,f),
// t=[t1, t2=t1+Dt, ..., tN] e // f=[f1, f2, ..., fN];
// O usuário é convidado a dar valores "tentativa"\ para
// os limites da frequência w e número de frequências,
// nw, que poderão ser modificados a seguir

[j,k]=size(f); if(j==1); f=f';end // f será coluna
[j,k]=size(t); if(k==1); t=t';end // t será fila
N=length(t); i=%i;pi=%pi;
Dt=t(2)-t(1);
xset('window',1); clf;
plot(t,f); legend('f(t)')
par=[0,0,0]
while(length(par)==3)
    par=input('deh [wmin, wmax, nw], enter encerra \n');
    if(length(par)~=3); break; end;
    wmin=par(1); wmax=par(2); nw=par(3);
    w=linspace(wmin, wmax, nw)';
    E=w*t; // matriz nw x N
    W=exp(i*E);
    g=Dt*(W*f); // transformada de f, vetor coluna
    rg=real(g); ig=imag(g);
    xset('window',2); clf; plot(w,rg);legend('Re(g(w))')
    xset('window',3); clf; plot(w,ig);legend('Im(g(w))')
// se o conjunto de fréquencias escolhidas não for bom,
```

```
// dê novos valores
end
G=[w';g']; // w' e g' são setores fila.
endfunction
```

Exemplo 1.5: Gaussiana Centrada em $t = 0$

Uma Gaussiana centrada na origem é uma função real e par. Portanto sua transformada de Fourier é também real e par. Na verdade ela é também uma Gaussiana, que pode ser obtida analiticamente. Mas é instrutivo ver o resultado da aplicação da função tfd.sci a este caso.

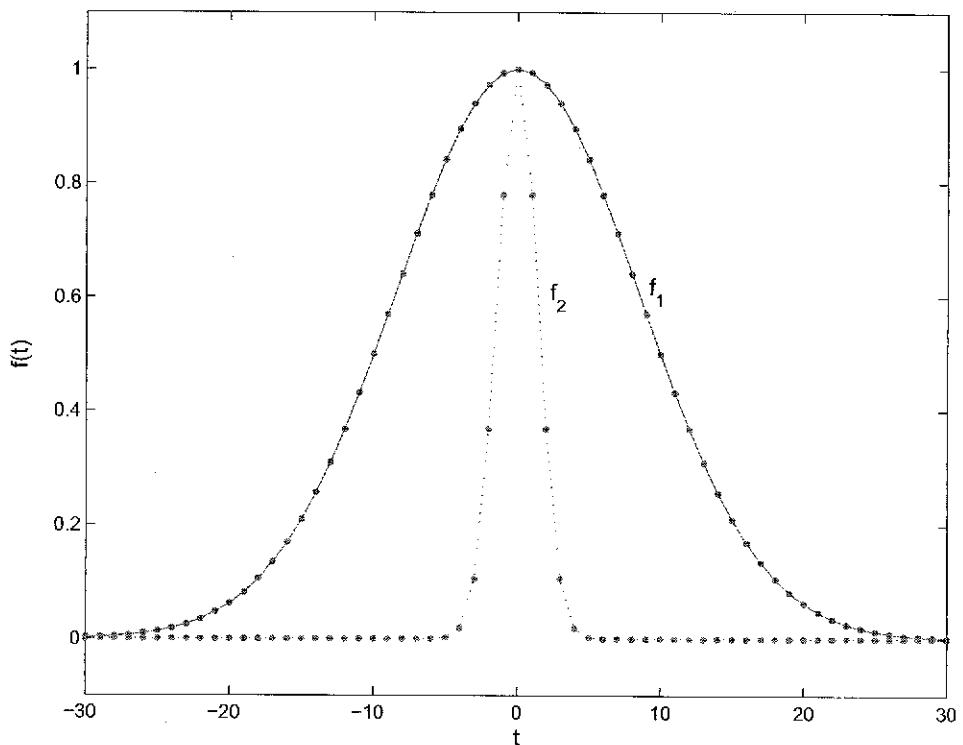


Figura 1.6: Gaussianas $f(t)$ de diferentes larguras

Na figura 1.6 temos duas Gaussianas, uma larga, $f_1(t)$ e uma estreita, $f_2(t)$. Os pontos destacados são os que foram usados para

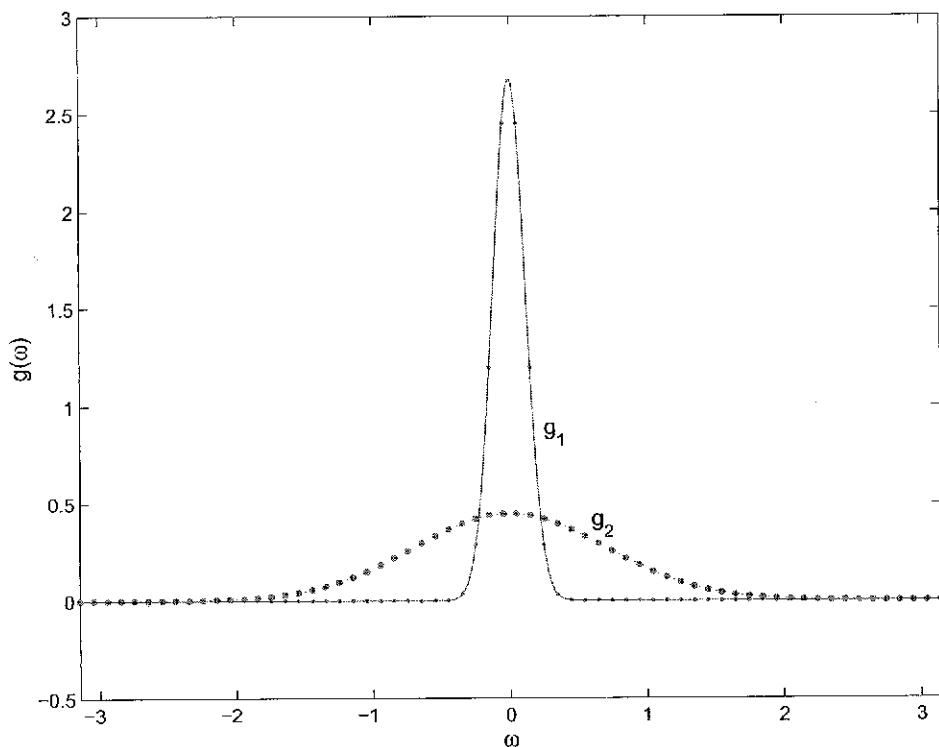


Figura 1.7: Transformadas das $f(t)$ da figura 1.6

aplicação de `tfid.m`, que resultou nas transformadas de Fourier, figura 1.7. Vemos que a função larga na variável t tem uma transformada de Fourier estreita em ω e vice-versa. Este fato não é uma particularidade da Gaussiana, mas uma propriedade geral das transformadas de Fourier: quanto mais larga for uma função, tanto mais estreita será sua transformada de Fourier. O caso extremo é a transformada da delta de Dirac (largura zero), que é uma constante (largura infinita). Esta propriedade tem sua correspondência em Mecânica Quântica: O momentum de uma partícula é $p = \hbar k$ onde $k = 2\pi/\lambda$, chamado “número de onda”, é a variável independente de uma transformada de Fourier no espaço, x . A incerteza na posição de uma partícula é a largura da função de onda, Δx , e a incerteza no momentum é \hbar vezes a largura de sua transformada de Fourier, Δk . O princípio de incerteza de Heisenberg diz que $\Delta x \Delta k \geq 1/2$, o que é perfeitamente

coerente com a propriedade mencionada.

Exercício 1.8:

Produza Gaussianas

$$f(t) = \exp\left(-\frac{(t - t_0)^2}{2\sigma^2}\right) \quad (1.88)$$

com diferentes vetores $t = (t_1, t_2, \dots, t_N)$, diferentes centros t_0 e diferentes larguras σ . Observe como varia a transformada de Fourier frente às referidas variações da $f(t)$. Em particular, verifique que quando a Gaussiana não é par em torno de $t = 0$ a parte imaginária da g não é nula mas o valor absoluto da $g(\omega)$ continua sendo uma Gaussiana.

Oscilador Harmônico Amortecido

A figura 1.8 mostra a amplitude de um oscilador harmônico amortecido, mas também pode representar a função resposta (polarização) de um material polarizável a um pulso elétrico aplicado em $t = 0$. Ela foi obtida fazendo $t = \text{linspace}(0, 10, 256)$ e $f = \sin(10*t).*\exp(-t)$, ou seja, é uma oscilação de frequência $\omega = 10$ e tempo de relaxação $\tau = 1$. Sua transformada de Fourier que, neste caso, pode ser chamada de “suscetibilidade dinâmica”, é mostrada nas figuras 1.9 (parte real) e 1.10 (parte imaginária). Os valores de frequência onde a parte imaginária tem picos (positivo ou negativo) são as frequências de ressonância. Neste exemplo temos, de fato, apenas uma frequência de ressonância, pois, em oscilações lineares, frequência positiva e negativa são, fisicamente, a mesma coisa. Nestes valores de frequência a parte real da transformada corta o eixo da abscissa ($g = 0$).

As figuras 1.9 e 1.10 tem seus eixos assim como foram produzidos por “`plot(ω, real(g(ω)))`” e “`plot(ω, imag(g(ω)))`”. Poderíamos reduzir a extensão do eixo ω para obter uma figura mais bem distribuída. Alternativamente, este resultado pode ser obtido alterando os limites de frequência, ω_{min} e ω_{max} ao executar a "function" `tfd.sci`. A figura 1.11 foi obtida por este procedimento.

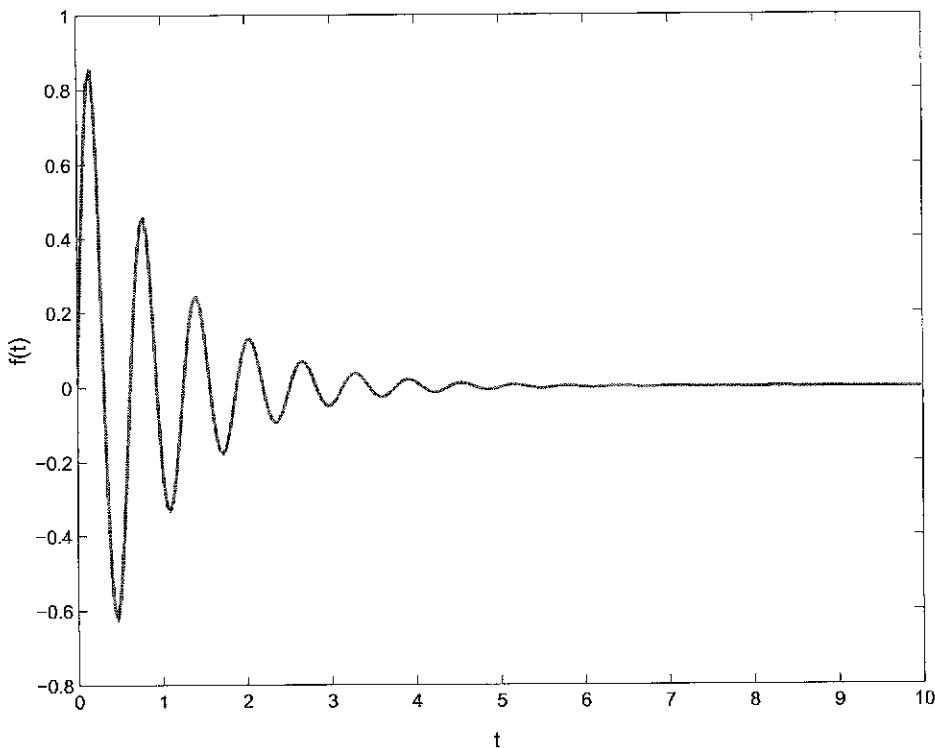


Figura 1.8: Oscilador harmônico amortecido

1.4.4 Transformada de Fourier Rápida

No caso de funções $f(t)$ muito rica em detalhes podemos necessitar de um número N muito grande de pontos na discretização de t para obtermos um gráfico fiel. A matriz W introduzida na seção anterior tem N^2 elementos complexos. Se, por exemplo, $N \sim 10^5$, então $N^2 \sim 10^{10}$ e não apenas o espaço de memória necessário para armazenar W se torna demasiadamente grande para muitos computadores, mas o tempo de CPU para executar o produto $W * f$, Eq.(1.87), se torna proibitivo. Este problema é particularmente grave quando a variável independente tem duas ou três dimensões, como, por exemplo, nas transformadas de Fourier no espaço ($t = \mathbf{r}$) que são usadas em Física do Estado Sólido.

Um algoritmo rápido e eficiente para implementar a transformada de Fourier discreta, hoje conhecido por “fft” (fast Fourier trans-

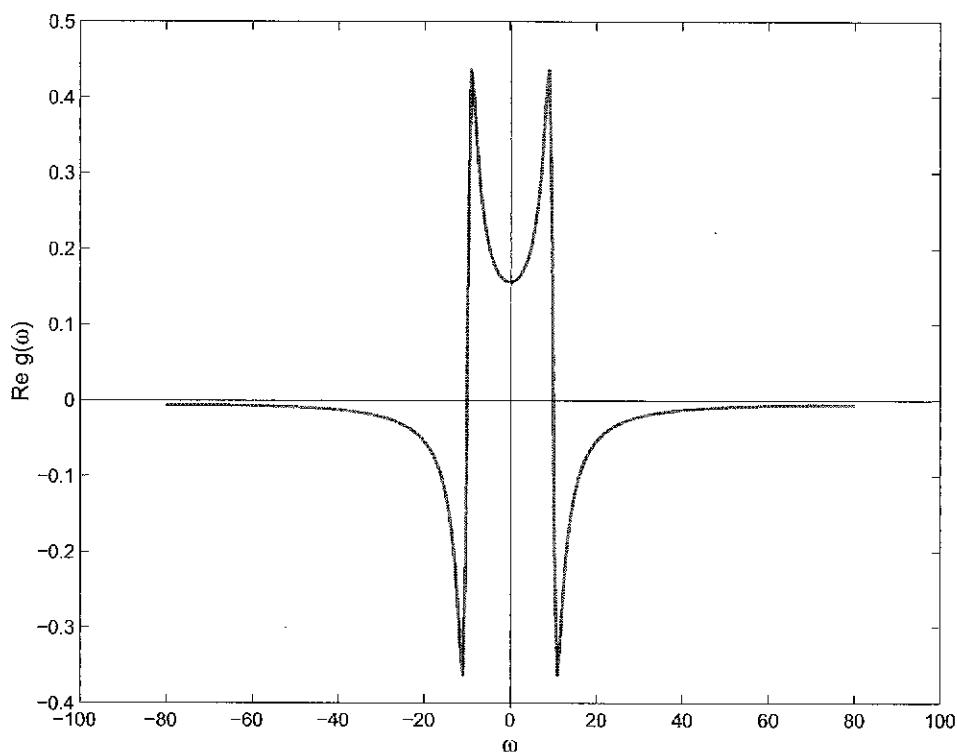


Figura 1.9: Parte real da transformada do oscilador da figura 1.8

form), tem sido desenvolvido com a colaboração de muitos autores, tornando-se amplamente usado a partir de meados da década de 1960. Entretanto, muito antes disso já se pode encontrar referências a um tal procedimento, iniciando por Gauss, em 1805. Mais recentemente, merecem menção o trabalho de Danielson e Lanczos (1942), que mostra que uma tfd de comprimento N pode ser re-escrita como duas de comprimento $N/2$ e o trabalho de Cooley e Tukey[7] pelo qual o algoritmo passou a ser mais amplamente conhecido.

Vamos apresentar brevemente a ideia básica do algoritmo. Vamos supor que N seja uma potência de 2, isto é, $N = 2^n$, sendo n um número inteiro. Como estamos tratando de funções $f(t)$ de suporte limitado, se N não for uma potência de dois nós completamos $f(t)$ com zeros até que N seja uma potência de dois. Por simplicidade vamos escolher unidades tais que $\Delta t = 1$ e as frequências ω no inter-

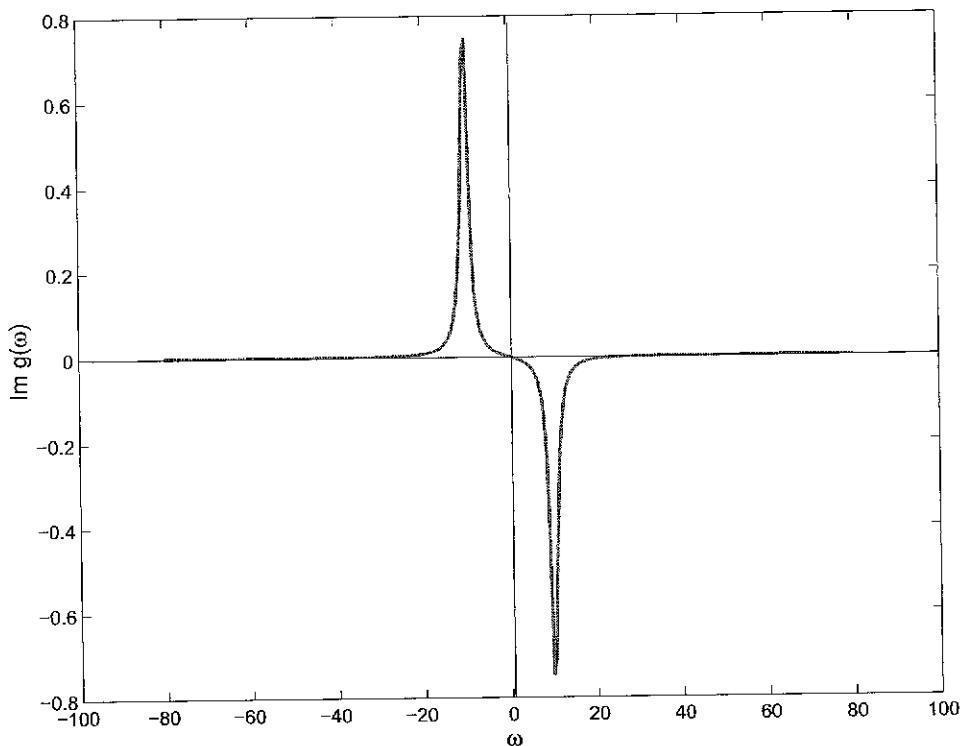


Figura 1.10: Parte imaginária da transformada do oscilador da figura 1.8

valo $[0, 2\pi)$ (aberto à direita por que excluímos o ponto $\omega = 2\pi$ que é equivalente a $\omega = 0$. Introduzimos a notação $f_j = f(t_j) = f(j)$ e $g_k = g(\omega_k)$, com $\omega_k = 0, 2\pi/N, \dots, k * 2\pi/N \dots$. Assim a Eq.(1.87) pode ser escrita na forma

$$g_k = \sum_{j=0}^{N-1} [\exp(-2\pi i/N)]^{kj} f_j . \quad (1.89)$$

Como N é par podemos separar o somatório acima em dois, cada um com $N/2$ termos:

$$g_k = \sum_{j=par}^{N-2} [\exp(-2\pi i/N)]^{kj} f_j + \sum_{j=impar}^{N-1} [\exp(-2\pi i/N)]^{kj} f_j$$

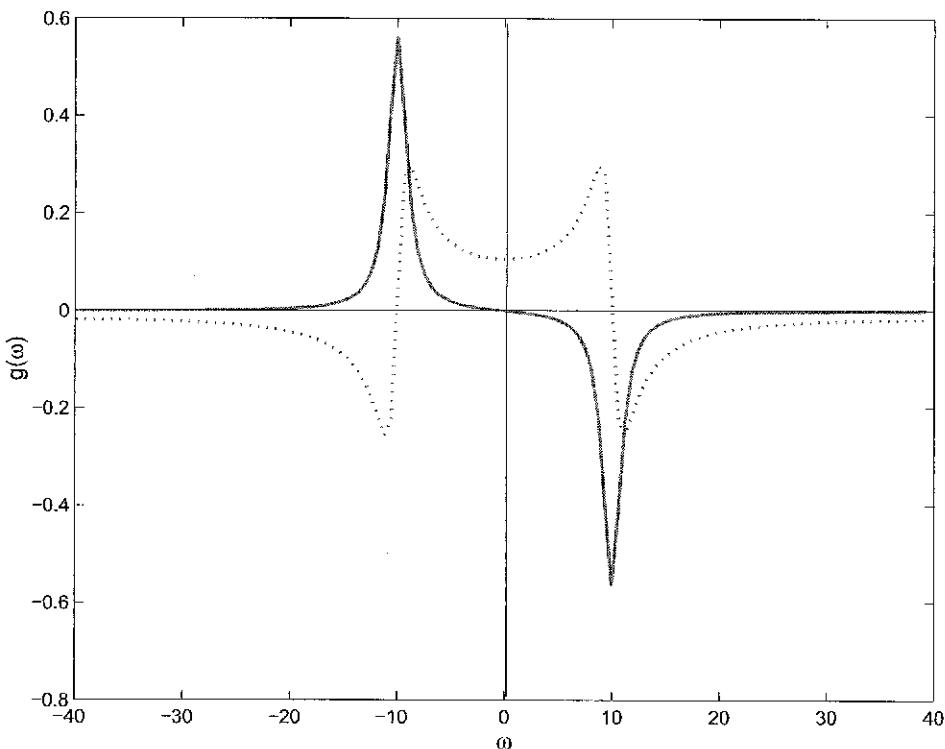


Figura 1.11: Parte real (linha pontilhada) e parte imaginária (linha cheia) da transformada do oscilador da figura 1.8

$$\begin{aligned}
 &= \sum_{j=0}^{(N/2)-1} [\exp(-2\pi i/N)]^{k(2j)} f_{2j} + \\
 &+ \sum_{j=0}^{(N/2)-1} [\exp(-2\pi i/N)]^{k(2j+1)} f_{2j+1}. \tag{1.90}
 \end{aligned}$$

Introduzimos agora a notação $W = \exp(-2\pi i/N)$ e $N' = N/2$ e re-escrevemos a Eq.(1.90) na forma

$$\begin{aligned}
 g_k &= \sum_{j=0}^{N'-1} [\exp(-2\pi i/N')]^{kj} f_{2j} + W^k \sum_{j=0}^{N'-1} [\exp(-2\pi i/N')]^{kj} f_{2j+1} \\
 &= g_k^{(par)} + W^k g_k^{(impar)} \tag{1.91}
 \end{aligned}$$

Como vemos, separamos a transformada de Fourier em duas, uma dos pontos pares de t e a outra dos pontos ímpares. Com isso a Eq.(1.87) passa a conter $2N'^2 = N^2/2$ multiplicações em vez dos N^2 da forma original. Como $N = 2^n$ tem-se $N' = 2^{n-1}$ e portanto também par. Assim podemos separar cada transformada da Eq.(1.91) em duas, cada uma para $N/4$ pontos. Continuamos o procedimento até que cada transformada se reduz a um único termo. Pode-se verificar que o procedimento completo exige $N \log_2(N)$ multiplicações de números complexos, em vez dos N^2 da forma original. Para N grande esta redução é enorme, podendo reduzir dias de computação para alguns segundos.

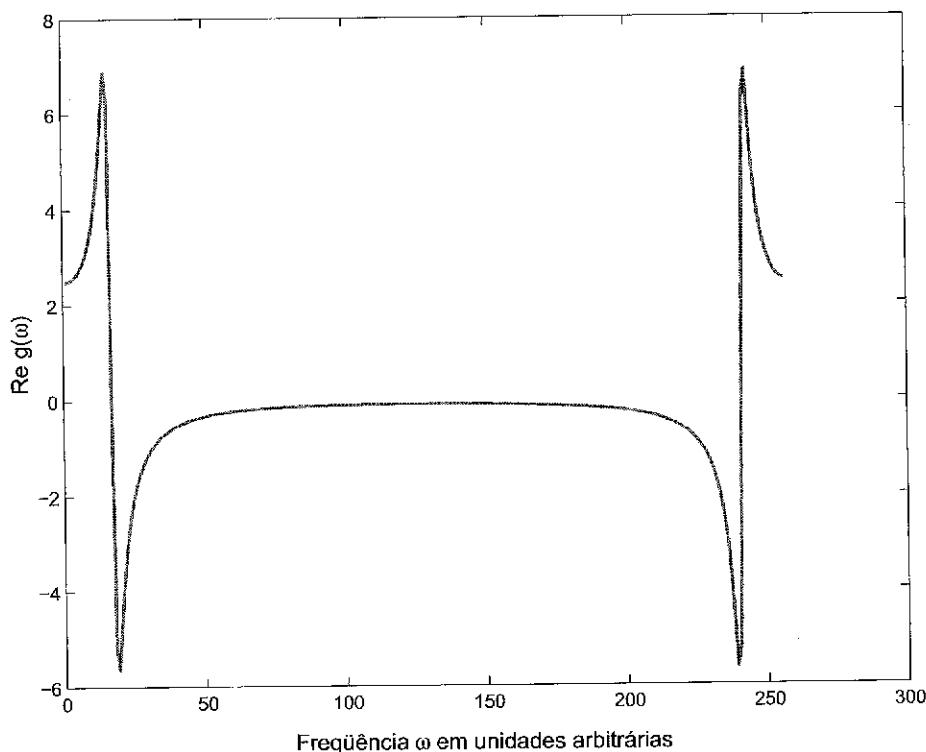


Figura 1.12: Parte real da transformada fft do oscilador da figura 1.8

Quase todas as linguagens de programação possuem fft como subrotina de biblioteca, geralmente de forma otimizada, de maneira que não paga a pena escrever uma nova rotina para ela. Em SCILAB

temos a “function” $fft(f)$, onde $f = [f_1, f_2, \dots, f_N]$. Para a $f(t)$ da figura 1.8 o gráfico de $real(fft(f))$ é mostrado na figura 1.12. Como vemos a $fft(f)$ apresenta alguns inconvenientes, que, no entanto, podem ser sanados. Como as frequências de fft estão definidas no intervalo $[0, 2\pi]$, as componentes de baixa frequência da transformada encontram-se nas extremidades do gráfico e não no centro, como na figura 1.9. Além disso, por usar $\Delta t = 1$ a transformada não traz informação sobre os valores das frequências, registrando a abscissa do gráfico apenas os índices das componentes ω_k do vetor de frequências. Se a $f(t)$ contém valores negativos de t em seu domínio, este fato não é levado em conta, considerando sempre $t_1 = 0$. Todos estes problemas podem ser corrigidos usando-se as propriedades das transformadas de Fourier. Nosso programa $tfr(t, f)$ (Transformada de Fourier Rápida) usa a $fft(f)$ mas apresenta o resultado efetuando as correções dos problemas mencionados acima.

Programa: tfr.sci

```

function G=tfr(t,f)
// Transformada de Fourier Rápida; dados os "vetores"
// t e f(t), t=t1, t2, ... , tN e f=f1, f2, ... , fN;
// Usa fft do SCILAB.
Dt=t(2)-t(1);
wN=%pi/Dt;
N=length(t);
N2=N/2;
n0=find(t==0); l0=length(n0);
lista=find(t>=0);
ntp=length(lista); //numero de tempos >0;
t1=t;
if(ntp < N); // significa que ha t<0;

////////////////// Transformada no caso de haver t<0:
////////// Simetrizar o tempo
ntn=N-ntp; // número de tempos <0;
N=2*max(ntp,ntn); // novo N;
N2=N/2;
if(ntp > ntn);

```

```

dpn=ntp-ntn;
t1=zeros(1,N);
if(10==1);
    t1(1)=-Dt*ntp;
    t1=t1(1):Dt:Dt*(ntp-1);
else
    t1(1)=-Dt*(ntp-.5);
    t1=t1(1):Dt:-t1(1);
end
elseif(ntn > ntp);
    dnp=ntn-ntp;
    t1=zeros(1,N);
    if(10==1); // existe t=0;
        t1(1)=-Dt*ntn;
        t1=t1(1):Dt:Dt*(ntn-1);
    else
        t1(1)=-Dt*(ntn-.5);
        t1=t1(1):Dt:-t1(1);
    end
end

//////// Verificar se N é potência de 2:
n=log(N)/log(2);
    if(ceil(n)^=n); // N não é 2^n
        n=ceil(n);
        N=2^n;           // aumenta N para 2^n
        N2=N/2;
        if(10==1); // se existe t=0;
            t1(1)=-Dt*N2;
            t1=t1(1):Dt:Dt*(N2-1);
        else
            t1(1)=-Dt*(N2-.5);
            t1=t1(1):Dt:-t1(1);
        end
    end
////////
///////////

```

```
////// Adicionar zeros na f:  
f1=zeros(1,N);  
f1(N2-ntn+1:N2)=f(1:ntn);  
f1(N2+1:N2+ntp)=f(ntn+1:ntn+ntp);  
/////  
  
////// dividir a f em parte par p e parte ímpar q  
fi=zeros(1,N);  
fi(N:-1:1)=f1;  
  
lp=N2+1:N;  
p=.5*(f1(lp)+fi(lp));  
q=.5*(f1(lp)-fi(lp));  
/////  
  
////// transformada simetrizada em w  
N4=N/4;  
gp=real(fft(p));  
gq=imag(fft(q));  
Gp=zeros(1,N2);  
Gp(1:N4)=2*gp(N4+1:N2);  
Gp(N4+1:N2)=2*gp(1:N4);  
Gq=zeros(1,N2);  
Gq(1:N4)=2*gq(N4+1:N2);  
Gq(N4+1:N2)=2*gq(1:N4);  
g=Gp+i*Gq;  
Dw=4*pi/(N*Dt);  
w=-wN:Dw:wN-Dw;  
//////////  
  
else  
    //// Verificar se N é potência de 2:  
    n=log(N)/log(2);  
        if(ceil(n) ~= n); // N não é 2^n  
        n=ceil(n);  
        L=N;  
        N=2^n;           // aumenta N para 2^n
```

```

N2=N/2;
t1=t(1):Dt:(N-1)*Dt;
f1=f; f1(L+1:N)=0; // acrescenta zeros
else
    t1=t;
    f1=f;
end
//////

Dw=2*pi/(N*Dt);
w=-wN:Dw:wN-Dw;
g1=fft(f1);
g(1:N2)=g1(N2+1:N);
g(N2+1:N)=g1(1:N2);
end
gn=g/sqrt(N-1); // normalização da g;
G=[w;gn];
xset('window',11); clf;
plot(t1,f1); legend('f(t)');
xset('window',12); clf;
plot(w,real(gn)); legend('Re(g(w))');
xset('window',13); clf;
plot(w,imag(gn)); legend('Im(g(w))')
endfunction

```

Exemplo 1.6:

Faça:

 $t = linspace(0, 10, 256);$
 $f = \sin(10 * t) .* \exp(-t);$
 $G = tfir(t, f);$

O programa apresentará os gráficos de $f(t)$, de $\operatorname{Re} g(\omega)$ e de $\operatorname{Im} g(\omega)$, que devem ser iguais aos das figuras 1.8, 1.9, e 1.10.

Um teste interessante que o leitor pode realizar é produzir uma $f(t)$ com $N = 2^{11} = 2048$ e a seguir::

1) execute:

 $tic; G = tfid(t, f); toc;$

com isto contando o tempo de CPU;

2) execute:

`tic; G = tfr(t, f); toc;`

e compare o tempo de CPU com o anterior.

Uma comparação ainda melhor entre os tempos de execução das duas rotinas se obtém desativando-se em ambas os comandos “plot”.

Capítulo 2

EQUAÇÕES DIFERENCIAIS ORDINÁRIAS

Métodos de solução numérica de equações diferenciais, tanto ordinárias como parciais, existem muitos. Não pretendemos apresentar todos eles neste capítulo, mas apenas uma seleção daqueles que elegemos como os mais representativos e importantes. Na bibliografia indicada no final do livro muitos outros métodos podem ser encontrados [4, 5, 6].

2.1 Equações de Primeira Ordem

Seja t uma variável real definida no intervalo $[0, t_{max}]$. Em Física geralmente t é o tempo. Inicialmente vamos ver como se resolve numericamente uma equação diferencial ordinária de primeira ordem,

$$\frac{dx}{dt} \equiv \dot{x} = f(x, t), \quad (2.1)$$

com condição inicial $x(0) = x_1$ (não usamos a notação mais usual x_0 por que em SCILAB os índices de vetores começam em 1).

2.1.1 Derivada Numérica

No capítulo I, seção 1.1.2, introduzimos os conceitos de diferenças finitas de primeira e segunda ordem e de derivadas numéricas. O

primeiro passo no procedimento numérico para se obter uma solução de uma equação do tipo da Eq.(2.1) consiste em *discretizar* a variável contínua t , i.e., substituí-la por um conjunto discreto crescente

$$t_1 = 0, \quad t_2 = t_1 + \Delta t, \quad t_3 = t_2 + \Delta t, \quad \dots, \quad t_n = t_{max}.$$

Em certos casos os intervalos ou *diferenças finitas* Δt podem ser variáveis, i.e., $t_{j+1} = t_j + \Delta t_j$, com $\Delta t_j \neq \Delta t_k$. A solução da Eq.(2.1) será um conjunto discreto

$$x_1, \quad x_2, \quad \dots, \quad x_n.$$

Em SCILAB tratam-se os conjuntos t_1, \dots, t_n e x_1, \dots, x_n como as componentes dos vetores t e x , respectivamente.

Usando a notação acima e o símbolo \dot{x} para indicar derivada em relação a t , a derivada numérica à direita, como definida pela Eq.(1.9), fica

$$\left. \frac{dx}{dt} \right|_{t_j} \equiv \dot{x}_j = \frac{x_{j+1} - x_j}{\Delta t}, \quad (2.2)$$

e a forma centrada, Eq.(1.11), é

$$\dot{x}_j = \frac{x_{j+1} - x_{j-1}}{2\Delta t}, \quad (2.3)$$

Usando-se a forma da Eq.(2.2) escreve-se a Eq.(2.1) como

$$x_{j+1} = x_j + \Delta t f(x_j, t_j) \quad (2.4)$$

e, por recorrência se obtém facilmente todo conjunto x_j a partir de x_1 (note que $f(x, t)$ é, em princípio, uma função conhecida). Este método de solução numérica de equações diferenciais de primeira ordem se chama **Método de Euler**.

Alternativamente, usando-se a forma da Eq.(2.3), escreve-se a Eq.(2.1) como

$$x_{j+1} = x_{j-1} + 2\Delta t f(x_j, t_j) \quad (2.5)$$

Essa forma é um pouco mais complicada de usar por que para calcular x_2 se precisaria de x_0 , que não está definido e para calcular x_3 se precisa de x_1 (condição inicial dada) e x_2 , que não foi calculado.

Pode-se contornar este problema usando a Eq.(2.4) apenas para calcular x_2 , e a partir de x_3 usar a Eq.(2.5).

Exemplo 2.1: Implementar a solução numérica de

$$\frac{dx}{dt} = x, \quad x(0) = 1,$$

com ambas as formas de derivada e comparar com a solução exata.

Programa: derivadas.sce

```
// Programa: derivadas.sce
// Neste exemplo se resolve a equação dx/dt=x;
// a solução é obtida com duas formas distintas
// de derivada numérica, "à direita"\e "centrada"
// e comparadas com a sol. exata, x=exp(t);
// Exemplo de dados: [tmax=5, dt=.1];
// use também dt=.001, .01, .1, .2, .5 para comparar ;

par=input('[tmax,dt],\n');
tmax=par(1); dt=par(2);
nt=floor(tmax/dt)+1;
t=linspace(0,tmax,nt);
x=zeros(1,nt);
x(1)=1; umdt=1+dt; h=2*dt;
// forma "forward"
for n=2:nt;
    x(n)=umdt*x(n-1);
end
// forma "centrada"
y(1)=1; y(2)=umdt*x(1);
for n=3:nt;
    y(n)=y(n-2)+h*y(n-1);
end
// solução exata
z=exp(t);
xset('window',1); clf;
plot(t,x,t,y,t,z,'--')
```

```
legend('a direita','centrada','exata', 2)
```

Exercício 2.1:

- Implemente o programa acima (derivadas.sce) e observe a convergência para a solução exata à medida que for diminuindo dt ;
- Modifique o programa acima para resolver a equação $dx/dt = ax^2$, com $x(0) = 1$, e execute-o com diferentes valores da constante a , positivos e negativos, comparando com a solução exata $x = 1/(1 - at)$.

Observações sobre a parte b:

- Para $a > 0$, em $t = 1/a$ a solução exata diverge; por isso, iniciando-se o procedimento numérico em $t = 0$, só se pode obter solução para tempos tais que $at < 1$;
- Para $a < 0$ a solução por “derivada centrada” apresenta um comportamento estranho, principalmente se $|a dt|$ não for muito pequeno; por exemplo, use $a = -1$, $dt = 0.1$, $tmax = 10$ e faça o gráfico da solução resultante marcando apenas os pontos calculados, “plot(t,y,’.’)”; observam-se duas linhas de pontos y_j , aparentemente independentes, uma para $j=\text{ímpar}$ e outra para $j=\text{par}$, ambas afastadas da solução exata; isto ocorre por que o incremento de y entre dois j ímpares consecutivos é calculado a partir do j par intermediário, e vice-versa, ou seja, os incrementos dos y_j de uma paridade são calculado pelos y_j da outra paridade, o que é diferente do prescrito pela equação diferencial que se pretende resolver; isto mostra que nem sempre a forma “centrada” da derivada primeira é a mais indicada.

2.1.2 Erro na Derivação Numérica

Vamos re-escrever a Eq.(2.2) na forma

$$x(t_j + \Delta t) = x(t_j) + \dot{x}(t_j)\Delta t. \quad (2.6)$$

Para que a Eq.(2.6) fosse mais exata precisaríamos acrescentar mais termos da série de Taylor, i.e., escrever

$$x(t_j + \Delta t) = x(t_j) + \dot{x}(t_j)\Delta t + \frac{1}{2}\ddot{x}(t_j)(\Delta t)^2 + O(\Delta t^3). \quad (2.7)$$

O símbolo $+O(\Delta t^3)$ indica que o erro dessa equação, por não incluirmos os termos subsequentes da série, é da ordem de $(\Delta t)^3$. Assim, o erro da Eq.(2.6) é $O(\Delta t^2)$. Como esse erro ocorre no ponto t_j , onde se está fazendo a derivação, ele se chama "erro local". Ao integrar-se uma equação com n pontos t_j diz-se que o "erro global" é $n = t_{max}/\Delta t$ vezes a ordem do erro local, ou seja, o erro global é uma ordem Δt mais baixa que o erro local. No caso da Eq.(2.6) o erro global é $O(\Delta t)$.

Trocando Δt por $-\Delta t$ na Eq.(2.7) segue

$$x(t_j - \Delta t) = x(t_j) - \dot{x}(t_j)\Delta t + \frac{1}{2}\ddot{x}(t_j)(\Delta t)^2 - O(\Delta t^3). \quad (2.8)$$

Subtraindo a Eq.(2.8) da Eq.(2.7) obtém-se

$$x(t_{i+1}) - x(t_{i-1}) = 2\Delta t \dot{x}(t_i) + O(\Delta t^3). \quad (2.9)$$

Logo, a derivada centrada, Eq.(2.3), tem erro $O(\Delta t^3)$.

O método de Euler, Eq.(2.2), é dito um método de *primeira ordem* por que o erro local é de segunda ordem em Δt e, portanto, o método da Eq.(2.5) é de *segunda ordem*. Este *método da derivada centrada* pode, entretanto, em alguns casos, apresentar grandes erros, como vimos na parte b do exercício 1.

2.1.3 Sistema de Equações

A Eq.(2.1) pode representar um sistema de n equações, se x e f forem vetores de n componentes, $x = (x_a, x_b, \dots, x_n)$ e $f = (f_a, f_b, \dots, f_n)$ (não confundir com a discretização x_1, x_2, \dots):

O procedimento numérico é idêntico ao exposto para o caso de uma variável, exceto que no caso de n variáveis, a cada passo se calculam todas as componentes de x e f .

Exemplo 2.2: Seja $\mathbf{x} = (x, y)$ e $\mathbf{f} = (-ay, bx)$. A equação equivalente à Eq.(2.4) é o sistema

$$\begin{aligned} x_{j+1} &= x_j - a \Delta t y_j \\ y_{j+1} &= y_j + b \Delta t x_j . \end{aligned} \quad (2.11)$$

No programa “sistema.sce”, abaixo, implementamos o exemplo 2.2, deixando como “input” os valores de $a, b, x(0), y(0)$. Definindo \mathbf{x} como vetor coluna, os passos de integração são realizados com auxílio da matriz

$$A = \begin{pmatrix} 0 & -a \\ b & 0 \end{pmatrix},$$

isto é,

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \Delta t A \mathbf{x}_j .$$

Salientamos que este truque só pode ser usado porque \mathbf{f} é função linear das componentes de x ; caso contrário teríamos que escrever as equações componente a componente, como na Eq.(2.11)

Programa: sistema.sce

```
// Programa: sistema.sce
// Resolve o sistema dX/dt =f, sendo X=(x,y) e
// f=(-a y, b x). Exemplo:
// [a=1, b=1, x(0)=1, y(0)=0, tmax=12, dt=.01]

par=input('deh [a, b, x(0), y(0), tmax, dt] \n');
a=par(1); b=par(2); x0=par(3);
y0=par(4); tmax=par(5); dt=par(6);
X0=[x0;y0];
nt=floor(tmax/dt)+1;
```

```
t=linspace(0,tmax,nt);
X=zeros(2,nt); X(:,1)=X0; // X(:,j) é o vetor [x;y]
A=[0,-a;b,0];           // no instante tj
// solução por Euler:
for j=1:nt-1;
    X(:,j+1)=X(:,j)+dt*A*X(:,j);
end
x=X(1,:); y=X(2,:);
clf; plot(t,x,t,y); legend('x(t)', 'y(t)')
```

Exercício 2.2:

- a) Execute o programa sistema.sce, com diferentes valores dos parâmetros;
 b) Modifique o programa sistema.sce, usando para \mathbf{f} uma função não linear de x e y (por ex., $f_x = -ay^3$, $f_y = bx^3$).

2.2 Equações de Segunda Ordem

As equações diferenciais de segunda ordem no tempo tem importância especial em Mecânica por ser o caso da Lei de Newton:

$$\frac{d^2\mathbf{x}}{dt^2} = \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, t) \quad (2.12)$$

Há duas maneiras de tratar a Eq.(2.12):

- 1) Escrever a Eq.(2.12) na forma de um sistema de equações de primeira ordem:

$$\begin{aligned} \frac{d\mathbf{x}}{dt} &= \mathbf{v} \\ \frac{d\mathbf{v}}{dt} &= \mathbf{f}(\mathbf{x}, \mathbf{v}, t) \end{aligned} \quad (2.13)$$

Os métodos de solução são os mesmos que acabamos de ver para sistemas de equações de primeira ordem. Em particular, interpretando y , no Exemplo 2, como a velocidade, temos um oscilador harmônico unidimensional.

2) Alternativamente, usamos o “Algoritmo de Verlet”, que passamos a descrever.

2.2.1 Algoritmo de Verlet

Usamos a seguinte forma discreta para a derivada segunda, conforme visto na seção 1.1.2:

$$\left(\frac{d^2x}{dt^2} \right)_{t=t_j} = \frac{x_{j+1} + x_{j-1} - 2x_j}{(\Delta t)^2} \quad (2.14)$$

Uma forma de escrever a solução numérica da Eq.(2.12), usando a Eq.(2.14), é:

$$x_{j+1} = f(x_j, v_j)(\Delta t)^2 + 2x_j - x_{j-1} + O(\Delta t^4) \quad (2.15)$$

Nesta forma não existe erro $O(\Delta t^3)$ devido à simetria da equação frente à troca $\Delta t \rightarrow -\Delta t$.

2.2.1.1 Forças independentes da velocidade

Note que na forma de solução da Eq.(2.15) não se calculam as velocidades, mas elas são usadas no cálculo da força. Por isso este algoritmo é mais apropriado para os casos de forças independentes da velocidade. Este é o "Algoritmo de Verlet", cujo erro é $O(\Delta t^4)$.

Mesmo no caso de força independente da velocidade, esta poderá ser necessária para a obtenção de variáveis, como a energia cinética, que dependem dela. Neste caso é usual definir-se

$$v_j = (x_{j+1} - x_{j-1})/2\Delta t \quad (2.16)$$

Inicialização: Para se calcular x_{j+1} pelo algoritmo de Verlet se necesita conhecer x nos dois instantes anteriores, x_j e x_{j-1} . Usualmente as condições iniciais fornecem a posição e a velocidade no instante inicial, x_1 e v_1 , de modo que não é possível aplicar diretamente o algoritmo de Verlet. Pode-se resolver este impasse calculando x_2 como se a aceleração f fosse constante no intervalo $[t_1, t_2]$, ou seja

$$x_2 = x_1 + v_1 \Delta t + \frac{1}{2} f(x_1)(\Delta t)^2. \quad (2.17)$$

O erro assim cometido é geralmente muito pequeno pois, além de ser cometido apenas no primeiro passo da integração, é $O(\Delta t^3)$.

Exemplo 2.3: Pêndulo simples rígido

Consideremos um pêndulo de comprimento $l = 1$. A equação de movimento para o ângulo θ , entre a barra do pêndulo e a vertical é

$$\ddot{\theta} = -g \sin(\theta).$$

O programa pendulo.sce, a seguir, implementa a solução do Exemplo 2.3 por Verlet e por Euler, para comparar. Usa $g = 10$. Calcula também a energia em função do tempo.

Programa: pendulo.sce

```
// Programa: pendulo.sce
// Movimento do pêndulo rígido, por 2 algoritmos:
// Verlet e Euler.
// Exemplo de dados: [v0=0, teta0=1.5, tmax=10, dt=.01]

par=input('deh [v0, teta0, tmax, dt], \n');
v0=par(1); teta0=par(2); tmax=par(3); dt=par(4);
dt2=dt^2; ddt=2*dt;
nt=floor(tmax/dt)+1;
t=linspace(0,tmax,nt);
teta=zeros(nt,1); // teta por Verlet; vetor coluna
v=zeros(nt,1); // velocidade por Verlet;
te=zeros(nt,1); // te=teta por Euler;
ve=zeros(nt,1); // ve=velocidade por Euler;
teta(1)=teta0; te(1)=teta0;
v(1)=v0; ve(1)=v0;
teta(2)=teta0+v0*dt-5*sin(teta0)*dt2;
te(2)=te(1)+ve(1)*dt;
ve(2)=ve(1)-10*sin(te(1))*dt;
```

```

for j=2:nt-1;
    // por Verlet:
    teta(j+1)=2*teta(j)-teta(j-1)-10*sin(teta(j))*dt2;
    v(j)=(teta(j+1)-teta(j-1))/dtt;
    // por Euler:
    te(j+1)=te(j)+ve(j)*dt;
    ve(j+1)=ve(j)-10*sin(te(j))*dt;
end
v(nt)=v(nt-1)-10*sin((teta(nt)+teta(nt-1))/2)*dt;
    // Energia por Verlet:
EV=.5*v.*v-10*cos(teta);
    // Energia por Euler:
EE=.5*ve.*ve-10*cos(te);
xset('window',1); clf; plot(t,teta,t,te)
legend('teta Verlet','teta Euler',2)
xset('window',2);clf; plot(t,Ev,t,EE)
legend('energia Verlet','energia Euler', 2)

```

Exercício 2.3:

Entenda o programa “pendulo.sce”, execute-o com diferentes valores dos parâmetros e analise os resultados.

2.2.1.2 Forças dependentes de velocidade:

Quando $f = f(x, v)$, o algoritmo de Verlet não pode ser empregado diretamente porque as velocidades não são calculadas. Não dá também para usar a Eq.(2.16) por que precisamos de v_j para calcular x_{j+1} . Uma boa aproximação é calcular v_j , para uso em $f(x_j, v_j)$, pela aproximação de Euler,

$$v_j = v_{j-1} + f_{j-1}\Delta t , \quad (2.18)$$

pois, nestas alturas já se tem f_{j-1} . O erro neste cálculo de v_j é $O(\Delta t^2)$, mas como, na expressão para x_{j+1} a f_j está multiplicada por Δt^2 , o erro em x_{j+1} é apenas $O(\Delta t^4)$. É necessário, para não acumular erro no cálculo dos próximos v_j , pelo uso sucessivo da Eq.(2.18), que após calcular x_{j+1} pelo método descrito acima, se recalcule v_j pela Eq.(2.16).

Exemplo 2.4: Trajetória de uma bola.

Consideremos que a bola com dada velocidade inicial, move-se sob ação de duas forças: seu peso e atrito com o ar:

$$\mathbf{f}(\mathbf{r}, \mathbf{v}) = [f_x, f_y] = [-\gamma v v_x, -g - \gamma v v_y],$$

onde v é o módulo de \mathbf{v} e tomamos massa $m = 1$. A equação de movimento é

$$\frac{d^2\mathbf{r}}{dt^2} = \mathbf{f}(\mathbf{r}, \mathbf{v}).$$

O programa “bola.sce”, a seguir, resolve o Exemplo 4 por Verlet e por Euler, apresentando também a solução exata para o caso sem atrito e a solução com atrito por Verlet com dt 10 vezes menor (solução “quase-exata”).

Programa: bola.sce

```
// Programa: bola.sce
// trajetória de uma bola, em presença de atrito do ar;
// solução por Euler e por Verlet, e para atrito nulo
// solução exata. Obtém também uma solução "quase exata",
// com dt 10 vezes menor, para comparar; Exemplo:
// [v0=10, theta=%pi/4, gama=.01, dt=.1]

par=input('deh [v0, teta0, gama, dt], \n');
v0=par(1); teta0=par(2); gama=par(3); dt=par(4);
dt2=dt^2; ddt=2*dt;
vx0=v0*cos(teta0); vy0=v0*sin(teta0);
tmax=vy0/5; // escolhemos tmax de modo que, sem atrito,
// y(tmax)=vy0*tmax-.5*g*tmax^2=0;
nt=floor(tmax/dt)+1;
t=linspace(0,tmax,nt);
// solução por Euler:
xe=zeros(nt,1); ye=zeros(nt,1); // posições
vxe=zeros(nt,1); vye=zeros(nt,1); // velocidades
vxe(1)=vx0; vye(1)=vy0;
```

```

for j=1:nt-1;
    ve=norm([vxe(j),vye(j)]);
    xe(j+1)=xe(j)+vxe(j)*dt;
    ye(j+1)=ye(j)+vye(j)*dt;
    vxe(j+1)=vxe(j)-gama*ve*vxe(j)*dt;
    vye(j+1)=vye(j)-(10+gama*ve*vye(j))*dt;
end
// solução por Verlet
x=zeros(nt,1); y=zeros(nt,1); // posições
x(2)=vx0*dt-.5*gama*v0*vx0*dt2;
y(2)=vy0*dt-.5*(10+gama*v0*vy0)*dt2;
vx=vx0-gama*v0*vx0*dt; // para j=2
vy=vy0-(10+gama*v0*vy0)*dt;
vxtemp=vx; vytemp=vy; // estimativa temporária
for j=2:nt-1;
    v=norm([vxtemp,vytemp]); // norma de v em j
    fx=-gama*v*vxtemp; // fx em j
    fy=-(10+gama*v*vytemp); // fy em j
    x(j+1)=2*x(j)-x(j-1)+fx*dt2;
    y(j+1)=2*y(j)-y(j-1)+fy*dt2;
    vx=(x(j+1)-x(j-1))/ddt;
    vy=(y(j+1)-y(j-1))/ddt;
    vxtemp=vx+fx*dt; // vx em j+1 (aprox.)
    vytemp=vy+fy*dt; // vy em j+1 (aprox.)
end
// solução exata sem atrito:
xs=vx0*t; ys=vy0*t-5*t.^2;
// solução quase exata (Verlet com dt => dt/10)
dt=dt/10; dt2=dt^2; ddt=2*dt; T=floor(tmax/dt)+1;
X=zeros(T,1); Y=zeros(T,1);
X(2)=vx0*dt-.5*gama*v0*vx0*dt2;
Y(2)=vy0*dt-.5*(10+gama*v0*vy0)*dt2;
VX=vx0-gama*v0*vx0*dt;
VY=vy0-(10+gama*v0*vy0)*dt;
VXtemp=VX; VYtemp=VY; // estimativa temporária
for j=2:T-1;
    V=norm([VX,VY]); // norma de V em j

```

```

fx=-gama*V*VXtemp;
fy=-10-gama*V*VYtemp;
X(j+1)=2*X(j)-X(j-1)+fx*dt2;
Y(j+1)=2*Y(j)-Y(j-1)+fy*dt2;
VX=(X(j+1)-X(j-1))/ddt;
VY=(Y(j+1)-Y(j-1))/ddt;
VXtemp=VX+fx*dt; // vx em j+1 (aprox)
VYtemp=VY+fy*dt; // vy em j+1 (aprox)
end
clf; plot(x,y,xe,ye,X,Y, xs,ys)
legend('Verlet','Euler','Quase-Exata', 'Sem Atrito')

```

Exercício 2.4:

Entenda o programa “bola.sce”, execute-o com diferentes valores dos parâmetros e analise os resultados;

2.2.2 O Algoritmo “Leap-Frog” (Pulo de Rã)

Consiste em atualizar as velocidades em tempos intermediários àqueles em que são atualizadas as posições. Esclarecendo melhor, sejam x_1, x_2, \dots, x_n os valores de x calculados nos instantes t_1, t_2, \dots, t_n , sendo $t_{j+1} = t_j + \Delta t$. Os valores v_j , de v , serão calculados nos instantes $t_j + \Delta t/2$, ou seja, $v_j = v(t_{j+1/2})$. Teremos a seguinte sequência de cálculo:

$$\begin{aligned} x_{j+1} &= x_j + v_j \Delta t \\ v_{j+1} &= v_j + f(x_{j+1}) \Delta t \end{aligned} \tag{2.19}$$

Cuidado deve ser tomado com a inicialização. Suponhamos que a condição inicial é $x(t_1) = x(1) = x_0$ e $v(t_1) = v_0$. Então se calcula $v(1) = v_0 + f(x_0) * dt/2$. O cálculo de x_2, v_2 etc., segue normalmente pela Eq.(2.19). Notemos que este método também tem problema quando a força depende da velocidade, pois, no momento de calcular v_{j+1} não temos $v(t_{j+1})$. É claro que podemos contornar este problema usando um artifício semelhante ao da Eq.(2.18), mas adaptado ao presente caso. Como uma característica do método leap-frog

é ser simples, não exploraremos aqui o caso de força dependente de velocidade, para o qual existem métodos melhores (veja Runge-Kutta 4^a ordem, a seguir).

2.2.3 O Algoritmo “Velocity-Verlet”

Um algoritmo equivalente a Verlet na determinação da órbita, mas que determina a velocidade $\mathbf{v}(t)$, nos mesmos instantes em que determina $\mathbf{x}(t)$, como parte integrante do algoritmo, conhecido por “Velocity-Verlet”, usa, nesta ordem, as seguintes operações:

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{f}(t)\Delta t^2, \quad (2.20)$$

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \frac{1}{2}\{\mathbf{f}(t + \Delta t) + \mathbf{f}(t)\}\Delta t. \quad (2.21)$$

Notemos que a atualização da velocidade deve ser feita após a atualização da posição, pois a força usada para atualizar a velocidade é a média dos valores no início e no fim do intervalo, necessitando-se, portanto, usar $\mathbf{x}(t + \Delta t)$ para calcular $\mathbf{f}(t + \Delta t)$ e, a seguir, $\mathbf{v}(t + \Delta t)$. Notemos ainda que se a força depende da velocidade não podemos usar o esquema acima.

Vamos mostrar que “Velocity-Verlet” produz a mesma trajetória que “Verlet”. Para isso re-escrevemos a Eq.(2.20) com $t \rightarrow t + \Delta t$,

$$\mathbf{x}(t + 2\Delta t) = \mathbf{x}(t + \Delta t) + \mathbf{v}(t + \Delta t)\Delta t + \frac{1}{2}\mathbf{f}(t + \Delta t)\Delta t^2. \quad (2.22)$$

Subtraindo a Eq.(2.20) da Eq.(2.22) e rearranjando os termos obtém-se

$$\begin{aligned} \mathbf{x}(t + 2\Delta t) &= 2\mathbf{x}(t + \Delta t) - \mathbf{x}(t) + \{\mathbf{v}(t + \Delta t) - \mathbf{v}(t)\}\Delta t + \\ &\quad + \frac{1}{2}\{\mathbf{f}(t + \Delta t) - \mathbf{f}(t)\}\Delta t^2 \end{aligned} \quad (2.23)$$

e substituindo $\{\mathbf{v}(t + \Delta t) - \mathbf{v}(t)\}\Delta t$ por $\frac{1}{2}\{\mathbf{f}(t + \Delta t) + \mathbf{f}(t)\}\Delta t$, conforme Eq.(2.21), segue

$$\mathbf{x}(t + 2\Delta t) = 2\mathbf{x}(t + \Delta t) - \mathbf{x}(t) + \mathbf{f}(t + \Delta t)\Delta t^2 , \quad (2.24)$$

que é o algoritmo de Verlet (veja Eq.(2.15)).

Exercício 2.5:

Modifique o programa “pendulo.sce”, do Exemplo 3, retirando a solução por Euler e acrescentando soluções por “Velocity-Verlet” e por “Leap-Frog”; faça um “plot” para as 3 soluções, execute o programa com diversos valores dos parâmetros e analise os resultados.

2.2.4 Métodos Runge-Kutta

Consideremos novamente um sistema de equações diferenciais de primeira ordem, Eq.(2.10), que escrevemos na forma vetorial

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t). \quad (2.25)$$

Sob o nome *métodos Runge-Kutta de ordem n* incluímos todos os métodos de solução numérica de sistemas representados pela Eq.(2.25) tais que, para calcular \mathbf{x}_{j+1} , usam apenas o conhecimento de \mathbf{x}_j . Isto exclui, por exemplo, o algoritmo de Verlet, Eq.(2.15), que necessita de \mathbf{x}_j e \mathbf{x}_{j-1} para calcular \mathbf{x}_{j+1} . O método de Euler, por exemplo, é um Runge-Kutta de 1^a ordem.

2.2.4.1 Runge-Kutta 2^a ordem

Consideremos novamente o método de Euler,

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \Delta t \mathbf{f}(\mathbf{x}_j, t_j) \quad (2.26)$$

Por simetria, é claro que igualmente válido seria escolher para argumento da \mathbf{f} o extremo à direita do intervalo $[t_j, t_{j+1}]$, isto é, escrever

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \Delta t \mathbf{f}(\mathbf{x}_{j+1}, t_{j+1}) \quad (2.27)$$

Naturalmente, pode se esperar que soluções melhores se obtém com uma forma intermediária entre as das equações. Eq.(2.26) e Eq.(2.27), ou seja,

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \Delta t \mathbf{f} \left(\frac{\mathbf{x}_j + \mathbf{x}_{j+1}}{2}, \frac{t_j + t_{j+1}}{2} \right). \quad (2.28)$$

A dificuldade que se apresenta é que não conhecemos \mathbf{x}_{j+1} para pôr no argumento da \mathbf{f} . Um procedimento possível é usar a Eq.(2.26) para obter uma primeira aproximação para \mathbf{x}_{j+1} a ser usado no argumento da \mathbf{f} . Para simplificar a notação vamos supor que a \mathbf{f} não depende explicitamente de t e representar o valor aproximado, obtido pela Eq.(2.26), para $(\mathbf{x}_{j+1} + \mathbf{x}_j)/2$, por \mathbf{x}_{med} , ou seja

$$\begin{aligned} \mathbf{x}_{med} &= \mathbf{x}_j + 0.5 \Delta t \mathbf{f}(\mathbf{x}_j) \\ \mathbf{x}_{j+1} &= \mathbf{x}_j + \Delta t \mathbf{f}(\mathbf{x}_{med}). \end{aligned} \quad (2.29)$$

A Eq.(2.29) é uma **Runge-Kutta 2^a ordem (RK2)**, também chamada de "Euler modificada".

2.2.4.2 Runge-Kutta 4^a ordem (RK4)

O método Runge-Kutta em ordens superiores usa médias ponderadas da função \mathbf{f} calculada nos extremos e em pontos intermediários do intervalo $[t_j, t_{j+1}]$. De longe o mais utilizado é o esquema com precisão de quarta ordem. Considerando novamente a Eq.(2.25) e definindo $h = \Delta t/2$, o esquema é:

$$\begin{aligned} \mathbf{F}_1 &= \mathbf{f}(\mathbf{x}_j, t_j) \\ \mathbf{F}_2 &= \mathbf{f}(\mathbf{x}_j + h\mathbf{F}_1, t_j + h) \\ \mathbf{F}_3 &= \mathbf{f}(\mathbf{x}_j + h\mathbf{F}_2, t_j + h) \\ \mathbf{F}_4 &= \mathbf{f}(\mathbf{x}_j + \Delta t\mathbf{F}_3, t_j + \Delta t) \\ \mathbf{x}_{j+1} &= \mathbf{x}_j + \frac{\Delta t}{6}(\mathbf{F}_1 + 2\mathbf{F}_2 + 2\mathbf{F}_3 + \mathbf{F}_4) \end{aligned} \quad (2.30)$$

A demonstração de que o erro local deste esquema é $O(\Delta t^5)$ é bastante trabalhosa e deixaremos de apresentá-la. Em princípio podem ser construídos esquemas em qualquer ordem. Quanto mais alta a ordem, maior o número de vezes que se necessita calcular a \mathbf{f} por passo de integração. Por isso na maioria dos problemas não compensa, em tempo, ir além da 4^a ordem.

2.2.4.3 Forças dependentes de velocidade:

Uma vantagem importante dos métodos de Runge-Kutta é que não apresentam nenhuma dificuldade para tratar forças dependentes de velocidade. Efetivamente, como cada passo de integração é calculado a partir do estado final do passo anterior, qualquer força dependente das variáveis de estado (posição e velocidade) são calculadas naturalmente. Como exemplo tratamos o problema da bola com atrito do ar por Runge-Kutta 2^a e 4^a ordens no programa a seguir. Incluimos, também, para comparação, o cálculo por Verlet, usando o esquema do exemplo 2.4. Neste exemplo a força é dependente apenas da velocidade, não dependendo da posição. Por isso, no cálculo de F_1 , F_2 , F_3 e F_4 não se precisa atualizar x e y .

Exemplo 2.5

Vamos refazer o exemplo 2.4 (bola com atrito) resolvendo sua trajetória por Verlet, por RK2, por RK4 e por RK4 com dt 10 vezes menor (“quase exata”). Para executar RK4 nós criamos uma “function”, *fun(r,gama)*, onde $r = [x, y, vx, vy]$ que é usada para obter F_1, F_2, F_3 e F_4 . Note que as velocidades v_{4x} e v_{4y} não são guardadas como vetores, pois não precisamos das mesmas para graficar a trajetória, e que os vetores x_4 e y_4 não são usados em “*fun(r,gama)*”, pois as forças não dependem de posição.

Programa: *bola_RK.sce*

```
// Programa: bola_RK.sce
// trajetória de uma bola, em presença de atrito do ar;
// solução por RK2, por RK4 e por Verlet.
// Obtém também uma solução "quase exata",
// com dt 10 vezes menor, por RK4, para comparar;
// Exemplo:
// [v0=400, teta=0.8, gama=0.01, dt=.1, tmax=11]

par=input('deh [v0, teta, gama, dt, tmax], \n');
v0=par(1); teta=par(2); gama=par(3);
dt=par(4); tmax=par(5);
dt2=dt^2; ddt=2*dt; h=dt/2;
vx0=v0*cos(teta); vy0=v0*sin(teta);
```

```

nt=floor(tmax/dt)+1;
t=linspace(0,tmax,nt);
// solução por Verlet
x=zeros(nt,1); y=zeros(nt,1); // posições
x(2)=vx0*dt-.5*gama*v0*vx0*dt2;
y(2)=vy0*dt-.5*(10+gama*v0*vy0)*dt2;
vx=vx0-gama*v0*vx0*dt; // para j=2
vy=vy0-(10+gama*v0*vy0)*dt;
vxtmp=vx; vtmp=vy; // estimativa temporária
for j=2:nt-1;
    v=norm([vxtmp,vtmp]); // norma de v em j
    fx=-gama*v*vxtmp; // fx em j
    fy=-(10+gama*v*vtmp); // fy em j
    x(j+1)=2*x(j)-x(j-1)+fx*dt2;
    y(j+1)=2*y(j)-y(j-1)+fy*dt2;
    vx=(x(j+1)-x(j-1))/ddt;
    vy=(y(j+1)-y(j-1))/ddt;
    vxtmp=vx+fx*dt; // vx em j+1 (aprox.)
    vtmp=vy+fy*dt; // vy em j+1 (aprox.)
end
// solução por RK2
x2=zeros(nt,1); y2=zeros(nt,1); // posições por RK2
vx2=zeros(nt,1); vy2=zeros(nt,1); // velocidades
vx2(1)=vx0; vy2(1)=vy0;
v2=v0;
for j=1:nt-1
    vxmed=vx2(j)-gama*v2*vx2(j)*h;
    vymed=vy2(j)-(10+gama*v2*vy2(j))*h;
    vmed=sqrt(vxmed^2+vymed^2);
    x2(j+1)=x2(j)+vxmed*dt;
    y2(j+1)=y2(j)+vymed*dt;
    vx2(j+1)=vx2(j)-gama*vmed*vxmed*dt;
    vy2(j+1)=vy2(j)-(10+gama*vmed*vymed)*dt;
    v2=sqrt(vx2(j+1)^2+vy2(j+1)^2);
end
// solução por RK4
x4=zeros(nt,1); y4=zeros(nt,1); // posições por RK2

```

2.2.4.3 Forças dependentes de velocidade:

Uma vantagem importante dos métodos de Runge-Kutta é que não apresentam nenhuma dificuldade para tratar forças dependentes de velocidade. Efetivamente, como cada passo de integração é calculado a partir do estado final do passo anterior, qualquer força dependente das variáveis de estado (posição e velocidade) são calculadas naturalmente. Como exemplo tratamos o problema da bola com atrito do ar por Runge-Kutta 2^a e 4^a ordens no programa a seguir. Incluimos, também, para comparação, o cálculo por Verlet, usando o esquema do exemplo 2.4. Neste exemplo a força é dependente apenas da velocidade, não dependendo da posição. Por isso, no cálculo de F_1 , F_2 , F_3 e F_4 não se precisa atualizar x e y .

Exemplo 2.5

Vamos refazer o exemplo 2.4 (bola com atrito) resolvendo sua trajetória por Verlet, por RK2, por RK4 e por RK4 com dt 10 vezes menor (“quase exata”). Para executar RK4 nós criamos uma “function”, $\text{fun}(r, gama)$, onde $r = [x, y, vx, vy]$ que é usada para obter F_1, F_2, F_3 e F_4 . Note que as velocidades v_{4x} e v_{4y} não são guardadas como vetores, pois não precisamos das mesmas para graficar a trajetória, e que os vetores x_4 e y_4 não são usados em “ $\text{fun}(r, gama)$ ”, pois as forças não dependem de posição.

Programa: bola_RK.sce

```
// Programa: bola_RK.sce
// trajetória de uma bola, em presença de atrito do ar;
// solução por RK2, por RK4 e por Verlet.
// Obtém também uma solução "quase exata",
// com dt 10 vezes menor, por RK4, para comparar;
// Exemplo:
// [v0=400, teta=0.8, gama=0.01, dt=.1, tmax=11]

par=input('deh. [v0, teta, gama, dt, tmax], \n');
v0=par(1); teta=par(2); gama=par(3);
dt=par(4); tmax=par(5);
dt2=dt^2; ddt=2*dt; h=dt/2;
vx0=v0*cos(teta); vy0=v0*sin(teta);
```

```

nt=floor(tmax/dt)+1;
t=linspace(0,tmax,nt);
// solução por Verlet
x=zeros(nt,1); y=zeros(nt,1); // posições
x(2)=vx0*dt-.5*gama*v0*vx0*dt2;
y(2)=vy0*dt-.5*(10+gama*v0*vy0)*dt2;
vx=vx0-gama*v0*vx0*dt; // para j=2
vy=vy0-(10+gama*v0*vy0)*dt;
vxtmp=vx; vtmp=vy; // estimativa temporária
for j=2:nt-1;
    v=norm([vxtmp,vtmp]); // norma de v em j
    fx=-gama*v*vxtmp; // fx em j
    fy=-(10+gama*v*vtmp); // fy em j
    x(j+1)=2*x(j)-x(j-1)+fx*dt2;
    y(j+1)=2*y(j)-y(j-1)+fy*dt2;
    vx=(x(j+1)-x(j-1))/ddt;
    vy=(y(j+1)-y(j-1))/ddt;
    vxtmp=vx+fx*dt; // vx em j+1 (aprox.)
    vtmp=vy+fy*dt; // vy em j+1 (aprox.)
end
// solução por RK2
x2=zeros(nt,1); y2=zeros(nt,1); // posições por RK2
vx2=zeros(nt,1); vy2=zeros(nt,1); // velocidades
vx2(1)=vx0; vy2(1)=vy0;
v2=v0;
for j=1:nt-1
    vxmed=vx2(j)-gama*v2*vx2(j)*h;
    vymed=vy2(j)-(10+gama*v2*vy2(j))*h;
    vmed=sqrt(vxmed^2+vymed^2);
    x2(j+1)=x2(j)+vxmed*dt;
    y2(j+1)=y2(j)+vymed*dt;
    vx2(j+1)=vx2(j)-gama*vmed*vxmed*dt;
    vy2(j+1)=vy2(j)-(10+gama*vmed*vymed)*dt;
    v2=sqrt(vx2(j+1)^2+vy2(j+1)^2);
end
// solução por RK4
x4=zeros(nt,1); y4=zeros(nt,1); // posições por RK2

```

```

function F=fun(r,gama);
    v=sqrt(r(3)^2 + r(4)^2);
    F=[r(3); r(4); -gama*v*r(3); -(10+gama*v*r(4))];
endfunction
r=[0;0;vx0;vy0];
for j=2:nt
    F1=fun(r);
    F2=fun(r+h*F1);
    F3=fun(r+h*F2);
    F4=fun(r+dt*F3);
    r=r+(dt/6)*(F1+2*F2+2*F3+F4);
    x4(j)=r(1);
    y4(j)=r(2);
end
// solução quase exata (RK4 com dt => Dt=dt/10)
Dt=dt/10; Dt2=Dt^2; DDt=2*Dt; H=Dt/2;
T=floor(tmax/Dt)+1;
X=zeros(T,1); Y=zeros(T,1);
R=[0;0;vx0;vy0];
for j=2:T
    F1=fun(R);
    F2=fun(R+H*F1);
    F3=fun(R+H*F2);
    F4=fun(R+Dt*F3);
    R=R+(Dt/6)*(F1+2*F2+2*F3+F4);
    X(j)=R(1);
    Y(j)=R(2);
end
clf; plot(x,y,x2,y2,x4,y4,X,Y,'k--')
legend('Verlet','RK2','RK4','Quase-Exata',2)

```

Exercício 2.6

Execute o programa “bola_RK.sce” com os parâmetros sugeridos no próprio programa e variando dt; observe como os resultados dos diversos algoritmos se aproximam e se afastam da solução “quase exata” para diferentes valores de dt.

2.2.5 Monitoração da Precisão pela Energia

Quando as forças não dependem da velocidade a energia total se conserva. Uma maneira de monitorar a precisão dos resultados é calcular a energia inicial $E_1 = E(\mathbf{x}_1, \mathbf{v}_1)$ e acompanhar a precisão do resultado nos instantes t_j calculando o erro relativo:

$$e_j = \frac{E(\mathbf{x}_j, \mathbf{v}_j) - E_1}{E_1} \quad (2.31)$$

Exercício 2.7:

Refaça o Exercício 2.5 (pêndulo), substituindo Euler e leap-frog por RK2 e RK4 e fazendo a monitoração da energia para todos os métodos, conforme Eq.(2.31). Sugestão: Ao programar RK4 crie uma “function” análoga à *fun(r, gama)* do programa bola_RK.sce, para calcular $F1, F2, F3$ e $F4$.

2.2.6 Incrementos Adaptativos

Às vezes, na integração de uma equação diferencial, há regiões onde os incrementos temporais, Δt , podem ser muito maiores que em outras regiões para gerar erros comparáveis. Por exemplo, na órbita de um cometa, enquanto ele estiver muito distante do Sol, propaga-se quase em linha reta e com velocidade lentamente variável, mas quando ele está próximo ao Sol sua trajetória e velocidade são rapidamente variáveis e se necessita Δt muito pequeno para não cometer erros muito grandes. Digamos que estabelecemos como o erro máximo admissível por passo um valor ϵ . Vamos chamar de *erro corrente*, ϵ_c , o erro relativo estimado no passo em execução. Não podemos conhecer com precisão o erro corrente. Há várias maneiras de se obter uma estimativa razoável para o mesmo. O método apresentado em “Numerical Recipes” [5] envolve o emprego de um algoritmo de ordem de precisão superior ao que está sendo correntemente empregado no cálculo e definir o erro como a diferença entre os resultados dos dois algoritmos. Alternativamente, A. Garcia (Numerical Methods for Physics)[6] sugere o seguinte procedimento:

Seja Δt o passo de tempo corrente e Δx o incremento calculado; seja δx o incremento calculado na soma de dois incrementos consecutivos com passos $\Delta t/2$ cada; como δx é muito mais exato que Δx , consideramos δx como exato e definimos o erro corrente como

$$\epsilon_c = \sqrt{\frac{(\Delta x - \delta x)^2}{(\delta x)^2}} \quad (2.32)$$

Se o algoritmo usado é de ordem $n - 1$ o erro local é de ordem n . Assim podemos esperar que para restabelecer um erro aproximadamente igual a ϵ precisamos renormalizar Δt para

$$\Delta t_{novo} \simeq \Delta t \left(\frac{\epsilon}{\epsilon_c} \right)^{1/n}$$

Para que não tenhamos que trocar Δt com muita frequência é conveniente introduzir um fator de segurança $S_1 < 1$ (por exemplo, $S_1 = 0.9$) e definir

$$\Delta t_{novo} = S_1 \Delta t \left(\frac{\epsilon}{\epsilon_c} \right)^{1/n}. \quad (2.33)$$

É ainda conveniente introduzir um segundo fator de segurança, $S_2 > 1$ (por exemplo, $S_2 = 2$) para evitar aumentos demasiadamente rápidos em Δt . Coloca-se a restrição, no programa, que se Δt_{novo} calculado pela Eq.(2.33) for maior do que $S_2 \Delta t$ então se faz $\Delta t_{novo} = S_2 \Delta t$.

Exemplo 2.6: Órbita de um cometa.

Embora se conheça a solução exata para a órbita de um corpo celeste em torno de um sol de massa infinitamente maior que a do corpo, este é um bom exemplo para aplicar algoritmos com passo adaptativo. Vamos considerar um cometa com massa desprezível frente à do Sol. Seja G a constante gravitacional universal, M a massa do Sol e \mathbf{r} a posição do corpo, estando o Sol na origem do sistema de referência. Por conservação de momentum angular (temos força central) o movimento ocorre num plano, ou seja, $\mathbf{r} = [x, y]$. A equação de movimento é

$$\frac{d^2\mathbf{r}}{dt^2} = \frac{-GM}{r^3}\mathbf{r}. \quad (2.34)$$

No programa “orbita.sce”, a seguir, calcula-se a órbita de um cometa. No mesmo gráfico apresenta-se a solução exata e a solução por Runge-Kutta 4^a ordem “adaptive steps”. Apresentam-se também os gráficos da energia, tamanho “tau” dos passos de integração, erro estimado por passo e número de ciclos usados no “loop” de renovação de tau. O programa "orbita.sce" chama a “function” “rka” para recalcular tau e esta chama a “function” “rk4o”, que atualiza o estado (\mathbf{r}, \mathbf{v}) . Usamos $GM = 1$.

Programa: orbita.sce

```
// Programa: orbita.sce; Calcula a órbita de um cometa
// usando "adaptive Runge-Kutta", redimensionando tau
// após cada passo; dá opção de continuação do cálculo.
// Traça também a órbita exata;
// Ex: [x0=1000, v0=.002, dt=5, erro=1.e-6, N=200]

par=input('deh [distancia, vel.tang., dt, erro, N],\n');
x0=par(1); v0=par(2); dt=par(3);
erro=par(4); N=par(5);
// Órbita exata:
L=x0*v0; E0=L^2/(2*x0^2)-1/x0; //mom. angular e energia
a=1/(2*E0); // semi-eixo maior
ec2=1+2*E0*L^2; ec=sqrt(ec2); // ecentricidade
pi=%pi;
teta=-pi:pi/101:pi; I=ones(1,length(teta));
re=a*(1-ec2)*I./(1+ec*cos(teta)); // raio exato
xe=re.*cos(teta); ye=re.*sin(teta);
// cálculo numérico por RK4:
estado=[x0, 0, 0, v0]; // a posição e a vel. inicial
x=[];y=[];vx=[];vy=[];t=[];tau=[]; errc=[];
x(1)=x0;y(1)=0; vx(1)=0; vy(1)=v0; n=1;
tau(1)=dt; t(1)=0;
function F=fun(R);
    r3=(sqrt(R(1)^2 + R(2)^2))^3;
```

```
F=[R(3), R(4), -R(1)/r3, -R(2)/r3];
endfunction
// introduz rka:
getf rka.sci
while(N >= 1)
    x=[x,zeros(1,N)];
    y=[y,zeros(1,N)];
    vx=[vx,zeros(1,N)];
    vy=[vy,zeros(1,N)];
    tau=[tau,zeros(1,N)];
    t=[t,zeros(1,N)];
    errocc=[errocc,zeros(1,N)];
for it=n+1:n+N;
    R=rka(estado, dt, erro);
    estado=R(1:4);
    x(it)=R(1); y(it)=R(2);
    vx(it)=R(3); vy(it)=R(4);
    tau(it)=R(5);
    if(it>1); t(it)=t(it-1)+tau(it); end
    errocc(it)=R(6); // erro calculado
    dt=R(5);
end;
xmin=min(x); xmax=max(x); ymin=min(y); ymax=max(y);
xset('window',0); clf; isoview(xmin,xmax,ymin,ymax);
plot(x,y,'k',0,0,'ro');
legend('órbita calculada')
n=n+N;
N=input('quantos passos mais? 0:encerra, \n');
end
// Energia
Ec=(vx.^2 + vy.^2)/2; raio=sqrt(x.^2+y.^2);
V=-ones(raio)./raio; E=Ec+V;
xmin=min(x); xmax=max(x); ymin=min(y); ymax=max(y);
xset('window',0); clf; isoview(xmin,xmax,ymin,ymax);
plot(xe,ye,x,y,0,0,'r.');
legend('exata','rka')
xset('window',1); clf;
```

```

plot(t,Ec,t,V,t,E);
legend('En. cinetica', 'En. potencial', 'En. total')
xset('window',2); clf;
plot(t,tau);
legend('tau(t)')
xset('window',3); clf;
plot(t,erroc);
legend('erro computado',4)

```

Função: rka.sci

```

function RK=rka(state, tau, erro);
// calcula um passo da integração da órbita por
// Runge-Kutta-4 "adaptive step", redimensionando,
// o intervalo de tempo, tau;
S1=.9; S2=2;
    //// atualização de state e tau /////
h=tau/2;hh=h/2;
    //// os dois passos pequenos /////
F1=fun(state);
F2=fun(state+hh*F1);
F3=fun(state+hh*F2);
F4=fun(state+h*F3);
F=(F1+2*F2+2*F3+F4)/6;
stmed=state+h*F;
F1=fun(stmed);
F2=fun(stmed+hh*F1);
F3=fun(stmed+hh*F2);
F4=fun(stmed+h*F3);
F=(F1+2*F2+2*F3+F4)/6;
stfin=stmed+h*F;
    //// passo único grande /////
F1=fun(state);
F2=fun(state+h*F1);
F3=fun(state+h*F2);
F4=fun(state+tau*F3);
F=(F1+2*F2+2*F3+F4)/6;

```

```

stg=state+tau*F;
    //// computa o erro corrente////
dif2=(stg(1)-stfin(1))^2+(stg(2)-stfin(2))^2;
ds2=(stfin(1)-state(1))^2+(stfin(2)-state(2))^2;
errc=sqrt(dif2/(ds2));
    //// decide novo tau ////
if(errc > erro);
    tau=S1*tau*(erro/(errc+%eps))^(.2);
elseif(errc < erro/S2)
    tau=S1*tau*(erro/(errc+%eps))^(.2);
    tau=min(tau, S2*tau);
end
RK=[stfin, tau, errc];

```

Exercício 2.8:

Considere uma bola que se choca frontalmente e elasticamente contra uma parede, num movimento unidimensional de ida e volta. Seja $x(t)$ a distância da bola à parede no instante t e seja $V = 1/x^{12}$ o potencial de interação entre a bola e a parede, e não há outras forças envolvidas. Modifique o programa “orbita.sce”, adaptando-o a este problema. Você deve também modificar a “function” rka.sci ou criar uma nova para este problema. Em vez do gráfico da órbita, neste caso devem ser produzidos gráficos de $x(t)$ e de $v(t)$ (velocidade). No cálculo do erro, convém somar os erros relativos na posição e na velocidade. O comportamento da posição e da velocidade devem ser observados com detalhe durante o choque. Exemplo de dados iniciais: $x_0 = 1$, $v_0 = -100$, $dt = 0.001$, $erro = 1.e-8$, $N = 500$.

Capítulo 3

EQUAÇÕES DIFERENCIAIS A DERIVADAS PARCIAIS

Métodos de solução numérica de equações diferenciais a derivadas parciais poderiam constituir um livro por si só. Nós optamos por apresentar, neste capítulo, uma seleção dos métodos que nos parecem os mais significativos para os físicos. Somente equações lineares, isto é, aquelas cujos termos são lineares na função incógnita ou independentes dela, serão tratadas. Além disso, na maior parte de nossa apresentação trabalharemos com apenas duas variáveis independentes, que denotaremos por t (tempo) e x (espaço). A extensão dos métodos para mais variáveis é, na maioria dos casos, bastante óbvia. Na seção 3.2 abordaremos métodos de encontrar soluções estacionárias para equações em duas ou mais dimensões espaciais.

3.1 Equações Uni-Dimensionais no Espaço

Vamos iniciar por uma equação diferencial linear na função incógnita $f(x, t)$, que escrevemos de forma bastante geral para incluir várias equações usuais em Física como casos particulares:

$$\frac{\partial f}{\partial t} = D(x, t) \frac{\partial^2 f}{\partial x^2} + d(x, t) \frac{\partial f}{\partial x} - S(x, t)f + s(x, t). \quad (3.1)$$

Na linguagem de “teoria de difusão” f representa uma “densidade” ou “concentração”, $D(x, t)$ um “coeficiente de difusão”, $d(x, t)$ um “coefi-

ciente de deriva (drift)", $S(x, t)$ um "sumidouro" ou "taxa de aniquilação" e $s(x, t)$ uma "fonte (source)". Mas a Eq.(3.1) também inclui, dependendo da escolha dos coeficientes, a equação de Schrödinger e outras.

Condições de Contorno

Sendo a Eq.(3.1) uma equação de primeira ordem em t e segunda ordem em x , necessita, para admitir solução única, uma condição inicial e duas condições de contorno, por exemplo $f(x, 0)$, $f(x_0, t)$ e $f(L, t)$. Há circunstâncias, entretanto em que a condição inicial é suficiente para determinar a solução num intervalo de tempo limitado e em uma região limitada do espaço.

Vamos tratar vários casos particulares, que usaremos para introduzir diferentes métodos de solução numérica.

3.1.1 Equação da Difusão

$$\frac{\partial f}{\partial t} = D \frac{\partial^2 f}{\partial x^2}. \quad (3.2)$$

Discretização:

de t : $t_1 = 0, \quad t_2 = dt, \quad \dots \quad t_{n+1} = n dt, \quad \dots$

de x : $x_1 = 0, \quad x_2 = dx, \quad \dots \quad x_{j+1} = j dx, \quad \dots$

de f : $f_j^n = f(x_j, t_n)$

Método FTCS (Forward Time Central Space)

É o método mais óbvio e um dos mais usados em equações deste tipo, e também em formas mais gerais da Eq.(3.1), como veremos. Consiste em definir a derivada em t na forma não simetrizada (forward) e as derivadas em x na forma simetrizada:

$$\frac{\partial f}{\partial t} = \frac{f(x, t + dt) - f(x, t)}{dt}$$

$$\frac{\partial^2 f}{\partial x^2} = \frac{f(x + dx, t) + f(x - dx, t) - 2f(x, t)}{(dx)^2}$$

Em notação discreta, a Eq.(3.2) fica:

$$\begin{aligned} f_j^{n+1} &= f_j^n + \frac{D dt}{(dx)^2} (f_{j+1}^n + f_{j-1}^n - 2f_j^n) \\ &= f_j^n + k (f_{j+1}^n + f_{j-1}^n - 2f_j^n) \end{aligned} \quad (3.3)$$

onde definimos

$$k = \frac{D dt}{(dx)^2}. \quad (3.4)$$

Convém notar que as diferenças finitas dt e dx e o coeficiente de difusão D comparecem na Eq.(3.3) apenas na forma $k = D dt / dx^2$. Tem-se, portanto, nessa equação, apenas um parâmetro, k . A forma óbvia de se resolver a Eq.(3.3) é por recorrência: Dada f_j^1 para todo j (condição inicial) calcula-se f_j^2 para todo j , e assim sucessivamente. Os valores de f^n nos extremos do intervalo, i.e., f_1^n e f_L^n , são dados como condições de contorno. Este procedimento “funciona” bem se $k \leq 1/2$, mas se $k > 1/2$ o resultado que se obtém é uma sequência cada vez maior de flutuações, divergente da solução correta da Eq.(3.2). Diz-se que a **condição de estabilidade numérica** da Eq.(3.3) é $k \leq 1/2$, ou seja $D dt / dx^2 \leq 1/2$. No procedimento “Crank-Nicholson”, que veremos adiante, pode-se usar um valor grande para k .

Exemplo 3.1:

Suponhamos que em $x = 0$ há uma fonte inesgotável de material que se difunde para $x > 0$ a partir do instante $t = 0$, isto é, $f(x = 0, t) = 1$. Suponhamos ainda que em $x = L$ há um sumidouro que absorve todo material que lá chega, ou seja, $f(L, t) = 0$. Como condição inicial supomos que $f(x > 0, t = 0) = 0$.

O programa “difusão”, a seguir, resolve este exemplo. O operador deve informar o comprimento L , o tempo $tmax$ de integração (duração do processo, em unidades de k) e o parâmetro k . O gráfico da concentração $f(x, tmax)$ é apresentado e o operador pode optar por continuar a integração por mais tempo, se desejar.

Programa: difusao.sce

```
// Solução da equação de difusão; condições de contorno:
```

```

// f(0,t)=1 e f(L,t)=0 (fonte inesgotável em x=0 e
// sumidouro em x=L; condição inicial: f(x>0,t=0)=0;
// dados: comprimento L, tmax e k=D*dt/dx^2);
// exemplo de dados: [L=100, tmax=400 k=.4]

dados=input('deh [L, tmax, k], \n');
L=dados(1); tmax=dados(2); k=dados(3);
x=0:L; // note que x(1)=0 e x(L+1)=L;
f=zeros(1,L+1);
f(1)=1;
while(tmax>0)
    for t=1:tmax
        f(2:L)=f(2:L)+k*(f(1:L-1)+f(3:L+1)-2*f(2:L));
    end
    plot(x,f)
    tmax=input('deh novo tmax; se enter, encerra \n');
end

```

Coeficiente de Difusão Dependente de Posição

Sempre que D depende de x devemos interpretar o termo em derivada segunda na posição como segue:

$$D \frac{\partial^2 f}{\partial x^2} \longrightarrow \frac{D_{j,j+1}(f_{j+1} - f_j) - D_{j-1,j}(f_j - f_{j-1})}{(dx)^2}, \quad (3.5)$$

onde $D_{j,j+1}$ é o coeficiente de difusão entre as posições x_j e x_{j+1} .

Com efeito, a equação da difusão vem da lei de Fick, que diz que a corrente de difusão, J_D , é proporcional ao negativo do gradiente de concentração, i.e.,

$$J_D = -D \frac{\partial f}{\partial x},$$

que, com a equação da continuidade, implica em:

$$\frac{\partial f}{\partial t} = -\frac{\partial J_D}{\partial x} = \frac{\partial}{\partial x} \left(D \frac{\partial f}{\partial x} \right). \quad (3.6)$$

O segundo membro da Eq.(3.5) é a forma discreta do último membro da Eq.(3.6).

Exemplo 3.2:

Suponhamos agora que a amostra onde ocorre a difusão tem comprimento L e que desde uma extremidade, $x = 0$, até $x = M$ há, no instante inicial, $t = 0$, o material difusivo em concentração 1. A condição inicial é, então,

$$f(0 \leq x \leq M, t = 0) = 1, \quad f(M < x \leq L, t = 0) = 0.$$

Neste caso o coeficiente de difusão para $x < 0$ e para $x > L$ é zero e a massa, $\int_0^L f(x, t) dx$, deve ser constante.

Exercício 3.1:

Modifique o programa “difusao.sce” para resolver o exemplo 2, usando, para a derivada segunda discretizada a forma da Eq.(3.5) e lembrando que D se anula nas extremidades da amostra. Executando o programa, verifique a conservação de matéria. Teste a validade da condição de estabilidade $D dt/dx^2 \leq 1/2$.

3.1.2 Equação de Deriva

É o caso particular da Eq.(3.1) em que o membro à direita só tem o termo em derivada primeira. Ela pode representar uma situação física em que não há corrente de difusão, só de deriva,

$$J = v f, \quad v = \text{constante.}$$

A equação será:

$$\frac{\partial f}{\partial t} = -\nabla \cdot J = -v \frac{\partial f}{\partial x}. \quad (3.7)$$

Qualquer função do argumento $x - vt$ é solução da Eq.(3.7), i.e., $f(x, t) = f(x - vt, 0)$, ou seja, uma distribuição inicial se propaga, sem deformação, com velocidade v .

A solução numérica da Eq.(3.7) serve para mostrar que mesmo para equações tão simples como esta pode haver instabilidade numérica. Em notação discreta a Eq.(3.7), usando FTCS, é

$$f_j^{n+1} = f_j^n - \frac{v dt}{2 dx} (f_{j+1}^n - f_{j-1}^n). \quad (3.8)$$

Exercício 3.2:

Programe a solução da Eq.(3.8) e verifique que ela é instável para qualquer valor do parâmetro $v dt/2 dx$.

Método de Lax

Uma pequena modificação na Eq.(3.8), introduzida por Lax, torna o procedimento numericamente estável, produzindo a solução correta. Consiste em substituir f_j^n por $\frac{1}{2}(f_{j+1}^n + f_{j-1}^n)$:

$$f_j^{n+1} = \frac{1}{2}(f_{j+1}^n + f_{j-1}^n) - \frac{v dt}{2 dx}(f_{j+1}^n - f_{j-1}^n). \quad (3.9)$$

Observação: a Eq.(3.9) pode ser pensada como a Eq.(3.8) mais o termo de difusão

$$\frac{1}{2}(f_{j+1}^n + f_{j-1}^n - 2f_j^n).$$

O critério de estabilidade, neste caso, conhecido como CFL (Courant-Friedrichs-Lowy), é

$$\frac{v dt}{2 dx} \leq \frac{1}{2}.$$

Na verdade, a melhor escolha é a relação acima na igualdade, pois para $v dt/2 dx < 1/2$ há aniquilação de massa.

Exercício 3.3: Modifique o programa feito para o exercício 3.2, introduzindo o método de Lax e teste o critério de estabilidade numérica com

$$v dt/2 dx > 1/2, = 1/2 \text{ e } < 1/2.$$

3.1.3 Defeitos por Radiação

Consideremos a equação

$$\frac{\partial f}{\partial t} = D \frac{\partial^2 f}{\partial x^2} - S f + s(x), \quad (3.10)$$

onde f pode, por exemplo, representar a concentração de vacâncias num sólido irradiado. D (coeficiente de difusão) e S (concentração de

sumidouros) são constantes e $s(x)$ (intensidade de fonte) representa a concentração de vacâncias criadas por segundo pela radiação. Um bom modelo é representar $s(x)$ por uma Gaussiana,

$$s(x) = s_0 \exp(-(x - x_0)^2 / 2\sigma^2) \quad (3.11)$$

Esta é uma versão simplificada do problema, em que o efeito dos defeitos “intersticiais” não é levado em conta.

Uma condição de contorno óbvia se obtém considerando que nos extremos da amostra as vacâncias se aniquilam imediatamente, ou seja, sua concentração é nula nesses pontos,

$$f(0, t) = f(L, t) = 0. \quad (3.12)$$

Como condição inicial podemos tomar $f(x, 0) = 0$.

Discretização: Se escolhemos escrever o passo de tempo dt em unidades de dx^2/D , no esquema FTCS a Eq.(3.10) fica:

$$f_j^{n+1} = f_j^n + dt * \{(f_{j+1}^n + f_{j-1}^n - 2f_j^n) - Sf_j^n + s_j\}. \quad (3.13)$$

Observação: Pela nossa escolha de unidade de tempo, o símbolo dt na Eq.(3.13) significa, efetivamente, $D dt/dx^2$; S e s também têm suas unidades modificadas em relação à Eq.(3.10).

O programa “defeitos”, a seguir implementa a solução da Eq.(3.13), plotando $f(x, t)$ para vários valores de t , no mesmo gráfico. Também se inclui nesse gráfico a solução estacionária. A explicação de como se obtém a solução estacionária é apresentada na seção seguinte.

Programa: defeitos.sce

```
// Eq. de difusão de defeitos com fonte e sumidouros;
// intensidade s0, posição central x0 e largura sigma
// da fonte, concentração S de sumidouros, largura L da
// amostra e k=D*dt/dx^2 são "input". Inclui solução
// estacionária.
// Ex. de dados: [10,100,10,.0002,400,2000,.4];
```

```
par=input('deh [s0,x0,sigma, S, L, tmax, dt] -\n');
s0=par(1); x0=par(2); sigma=par(3); S=par(4);
L=par(5); tmax=par(6); dt=par(7);
s2=1/(2*sigma^2);
nt=floor(tmax/dt);
x=1:L;
f=zeros(1,L); // f=concentração inicial de defeitos
s=s0*exp(-((x-x0).^2)*s2);

// solução estacionaria:
b=(2+S);
M=zeros(L);
for j=1:L;
    M(j,j)=b;
end
for j=1:L-1
    M(j,j+1)=-1;
    M(j+1,j)=-1;
end
m=inv(M);
fes=m*s';
r=0.7*max(fes); rs=r*s/s0;

// relaxação

while(tmax >= 0)
for it=2:nt;
    f(2:L-1)=f(2:L-1)+k*(f(3:L)+f(1:L-2)- ...
                           b*f(2:L-1)+s(2:L-1));
end

// gráfico dos resultados:
plot(x,rs,x,f,x,fes)
tmax=input('deh novo tmax; se enter encerra \n');
nt=floor(tmax/dt);
end
```

3.1.4 Solução Estacionária

Dependendo da forma dos coeficientes e condições de contorno de equações do tipo da Eq.(3.1), existe solução estacionária, isto é,

$$\frac{\partial f}{\partial t} = 0.$$

Uma maneira de encontrar a solução estacionária consiste em resolver a equação completa em função do tempo até um tempo tal que $f(x, t+dt) \simeq f(x, t)$, dentro de uma tolerância pré-estabelecida. Este método se chama “método de relaxação”. Você pode testá-lo com o programa “defeitos” do parágrafo anterior. Em muitos casos o método de relaxação é de execução muito demorada, o que nos motiva a procurar uma solução mais direta, igualando a zero o membro da direita da Eq.(3.1). Podemos sempre escrever a equação resultante na forma discreta (definindo, apropriadamente, a, b, c)

$$a_j f_{j-1} + b_j f_j + c_j f_{j+1} = s_j. \quad (3.14)$$

Sendo f_1 e f_L dados pelas condições de contorno, precisamos do sistema de equações para f_j , $j = 2, 3, \dots, L-1$:

$$\begin{aligned} b_2 f_2 + c_2 f_3 &= s_2 - a_2 f_1 \\ a_3 f_2 + b_3 f_3 + c_3 f_4 &= s_3 \\ a_4 f_3 + b_4 f_4 + c_4 f_5 &= s_4 \\ a_5 f_4 + b_5 f_5 + c_5 f_6 &= s_5 \\ &\dots \\ &\dots \\ a_{L-1} f_{L-2} + b_{L-1} f_{L-1} &= s_{L-1} - c_{L-1} f_L \end{aligned} \quad (3.15)$$

Temos assim um sistema linear de $L-2$ equações nas $L-2$ incógnitas f_2, \dots, f_{L-1} . Definimos a matriz M de dimensão $L-2 \times L-2$,

$$M = \left(\begin{array}{cccccc} b_2 & c_2 & 0 & 0 & \cdots & 0 \\ a_3 & b_3 & c_3 & 0 & \cdots & 0 \\ 0 & a_4 & b_4 & c_4 & \cdots & 0 \\ \cdots & & & & & \\ 0 & 0 & 0 & \cdots & a_{L-1} & b_{L-1} \end{array} \right) \quad (3.16)$$

e os vetores

$$\mathbf{g} = \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_{L-1} \end{pmatrix} \quad (3.17)$$

e

$$\mathbf{r} = \begin{pmatrix} s_2 - a_2 f_1 \\ s_3 \\ s_4 \\ \vdots \\ s_{L-1} - c_{L-1} f_L \end{pmatrix} \quad (3.18)$$

O sistema de equações (3.15) pode, então, ser escrito de forma matricial como

$$\mathbf{M} * \mathbf{g} = \mathbf{r}, \quad (3.19)$$

cuja solução é

$$\mathbf{g} = \mathbf{M}^{-1} * \mathbf{r}, \quad (3.20)$$

e a solução estacionária que procurávamos é então

$$\mathbf{f} = (f_1, g_1, g_2, \dots, g_{L-2}, f_L). \quad (3.21)$$

Esta solução, para o caso dos defeitos de radiação, está implementada na parte final do programa “defeitos.sce”, acima.

Nota: Embora seja muito mais rápido que o método de “relaxação”, este método matricial pode ser complicado de executar se L é muito grande, pois precisamos inverter a matriz \mathbf{M} . Como \mathbf{M} é tri-diagonal, existe uma maneira prática de resolver a Eq.(3.19) sem inverter a \mathbf{M} , chamada “algoritmo de Thomas”, que passamos a apresentar, como “receita”.

Algoritmo de Thomas

Vamos redefinir os índices de a, b, c , fazendo $j \rightarrow j - 1$, $N = L - 2$, de maneira que a matriz M passa a ser

$$M = \begin{pmatrix} b_1 & c_1 & 0 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & 0 & \cdots & 0 \\ 0 & a_3 & b_3 & c_3 & \cdots & 0 \\ \cdots & & & & & \\ 0 & 0 & 0 & \cdots & a_N & b_N \end{pmatrix} \quad (3.22)$$

Definimos então

$$\beta_1 = b_1, \quad \beta_j = b_j - a_j \frac{c_{j-1}}{\beta_{j-1}}, \quad j = 2, \dots, N \quad (3.23)$$

$$\gamma_1 = r_1, \quad \gamma_j = r_j - a_j \frac{\gamma_{j-1}}{\beta_{j-1}}, \quad j = 2, \dots, N. \quad (3.24)$$

Finalmente, em recorrência “para traz”,

$$g_j = \frac{\gamma_j - c_j g_{j+1}}{\beta_j}, \quad (3.25)$$

com $g_{N+1} = f_L$ (dado).

Exercício 3.4:

Faça um programa que implementa as duas formas acima, inversão da matriz M e algoritmo de Thomas, para obter a solução estacionária do problema “defeitos de radiação”, medindo o tempo de CPU de ambas. Você deve verificar que no caso de L grande (digamos $L=800$) o algoritmo de Thomas é consideravelmente mais rápido que a inversão de M .

3.1.5 Procedimentos “Explícito”, “Implícito” e “Crank-Nicholson”

Consideremos uma equação diferencial de primeira ordem em t e ordem qualquer em \mathbf{r} (d -dimensões espaciais), que escrevemos na forma

$$\frac{\partial f}{\partial t} = L\mathbf{r}f(\mathbf{r}, t) + \mathbf{s}(\mathbf{r}) \quad (3.26)$$

onde $L_{\mathbf{r}}$ é um operador diferencial linear em \mathbf{r} e $s(\mathbf{r})$ é um termo de “fonte”, independente de f . Em forma discretizada no tempo podemos escrever

$$f^{n+1}(\mathbf{r}) - f^n(\mathbf{r}) = dt [L_{\mathbf{r}} f^n(\mathbf{r}) + s(\mathbf{r})]. \quad (3.27)$$

É claro, por simetria, que uma forma igualmente válida se obtém usando para a f , à direita, seu valor no fim do intervalo, ou seja,

$$f^{n+1}(\mathbf{r}) - f^n(\mathbf{r}) = dt [L_{\mathbf{r}} f^{n+1}(\mathbf{r}) + s(\mathbf{r})]. \quad (3.28)$$

A forma da Eq.(3.27) é nossa conhecida FTCS, que também se chama “explícita”, por que no cálculo de f^{n+1} só se usa o valor já explicitamente calculado f^n , enquanto que a forma Eq.(3.28) é conhecida como forma “implícita” por que o valor de f^{n+1} está implícito na Eq.(3.28), faltando explicitá-lo. É claro que no limite $dt \rightarrow 0$ e $f(\mathbf{r}, t)$ derivável em t , as formas Eq.(3.27) e Eq.(3.28) são matematicamente equivalentes. Em cálculo numérico, entretanto, se usa dt finito e essas formas dão, geralmente, resultados distintos (erros distintos). Não há, em princípio motivo para considerar uma melhor que a outra, exceto do ponto de vista da facilidade de implementar o programa, sendo a forma da Eq.(3.27) muito mais simples. Do ponto de vista da precisão, onde uma erra por excesso a outra erra por falta, de maneira que uma forma mais precisa que ambas se obtém tomando a média das duas:

$$f^{n+1}(\mathbf{r}) - f^n(\mathbf{r}) = \frac{dt}{2} \left(L_{\mathbf{r}} f^{n+1}(\mathbf{r}) + L_{\mathbf{r}} f^n(\mathbf{r}) \right) + dt s(\mathbf{r}). \quad (3.29)$$

Note que assim temos uma derivada temporal simétrica em relação aos pontos inicial e final do intervalo. Re-escrevemos a Eq.(3.29) na forma

$$(1 - \frac{dt}{2} L_{\mathbf{r}}) f^{n+1}(\mathbf{r}) = (1 + \frac{dt}{2} L_{\mathbf{r}}) f^n(\mathbf{r}) + dt s(\mathbf{r}) \quad (3.30)$$

ou ainda

$$f^{n+1}(\mathbf{r}) = (1 - \frac{dt}{2} L_{\mathbf{r}})^{-1} [(1 + \frac{dt}{2} L_{\mathbf{r}}) f^n(\mathbf{r}) + dt s(\mathbf{r})] \quad (3.31)$$

Portanto, os elementos de matriz de L_r podem ser escritos na forma

$$[L_r]_{k,j} = \frac{-v}{2dx}(\delta_{k,j+1} - \delta_{k,j-1}) \quad (3.37)$$

Definindo

$$c = \frac{-vdt}{4dx} \quad (3.38)$$

os elementos da matriz que representa o operador

$$M = 1 + \frac{dt}{2} L_r \quad (3.39)$$

são

$$M_{k,j} = \delta_{k,j} + c(\delta_{k,j+1} - \delta_{k,j-1}). \quad (3.40)$$

Usando condições periódicas de contorno, conforme descrito acima, a Matriz M será escrita na forma

$$M = \begin{pmatrix} 1 & -c & 0 & 0 & \cdots & c \\ c & 1 & -c & 0 & \cdots & 0 \\ 0 & c & 1 & -c & \cdots & 0 \\ \dots & & & & & \\ & & & & & \\ -c & 0 & 0 & \cdots & c & 1 \end{pmatrix} \quad (3.41)$$

Um programa que resolve o exemplo 3.3 pode ser como segue.

Programa: deriva_CN.sce

```
// deriva_CN.sce: equação de deriva por Crank-Nicholson
// sendo c=bdt/4dx Ex: [1,500, 10]
```

```
par=input('deh [c,L,tmax] \n');
c=par(1); L=par(2); tmax=par(3);
L0=L/3; sig=L/10; sig2=2*sig*sig;
M=zeros(L,L); E=zeros(L,L);
x=(1:L)';
f=exp(-((x-L0).^2)/sig2);
for j=2:L
```

```

M(j,j)=1;
M(j-1,j)=-c;
M(j,j-1)=c;
E(j,j)=1;
E(j-1,j)=c;
E(j,j-1)=-c;
end
M(1,1)=1;
E(1,1)=1;
M(1,L)=c;
M(L,1)=-c;
E(1,L)=-c;
E(L,1)=c;
Einv=inv(E);
while(tmax > 0)
    for j=1:tmax
        f=Einv*(M*f);
    end
    clf
    plot(x,f)
    tmax=input('deh novo tmax; se enter encerra \n');
end

```

Exercício 3.5: Execute o programa acima com $L=500$ e dando valores $t_{max} = 10$, sucessivamente, observando a distribuição se deslocar, reentrando pelo lado oposto. Com diferentes valores de c você pode observar a estabilidade da solução.

Note que mesmo com $c = 2$ o programa funciona bem, i.e., a distribuição se desloca sem distorção. Este valor de c corresponde a $v dt/2dx = 4$ e nós vimos que o método de Lax só é estável se $v dt/2dx \leq 1/2$. Veremos, em outros exemplos, que o esquema de Crank-Nicholson apresenta grande estabilidade e precisão.

Exemplo 3.4: Defeitos por Crank-Nicholson

Consideremos novamente a Eq.(3.10). Comparando-a com a Eq.(3.26) identificamos

$$L\mathbf{r} = D \frac{\partial^2}{\partial x^2} - S. \quad (3.42)$$

Como precisamos, conforme Eq.(3.34), inverter o operador

$$E \equiv 1 - \frac{dt}{2} L\mathbf{r} = 1 - \frac{D dt}{2} \frac{\partial^2}{\partial x^2} + \frac{S dt}{2} \quad (3.43)$$

vamos escrevê-lo em forma matricial. O espaço vetorial correspondente é o que contém o vetor “função incógnita”

$$\mathbf{f}(x) = \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_N) \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_N \end{pmatrix} \quad (3.44)$$

assim como o vetor “fonte”

$$\mathbf{s}(x) = \begin{pmatrix} s(x_1) \\ s(x_2) \\ s(x_3) \\ \vdots \\ s(x_N) \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_N \end{pmatrix} \quad (3.45)$$

O operador “derivada segunda”, quando aplicado à $f(x)$ no ponto x_j resulta em

$$\frac{f_{j+1} + f_{j-1} - 2f_j}{dx^2}.$$

Assim os elementos da matriz que representa $\frac{\partial^2}{\partial x^2}$ são

$$\left[\frac{\partial^2}{\partial x^2} \right]_{k,j} = \frac{1}{dx^2} (\delta_{k,j+1} + \delta_{k,j-1} - 2\delta_{k,j})$$

e os elementos de matriz de E são

$$[E]_{k,j} = \delta_{k,j} \left(1 + \frac{D dt}{dx^2} + \frac{S dt}{2} \right) - (\delta_{k,j+1} + \delta_{k,j-1}) \frac{D dt}{2 dx^2}$$

Definindo-se ainda as seguintes constantes,

$$a = \frac{D dt}{2 dx^2}, \quad b = 1 + \frac{D dt}{dx^2} + \frac{S dt}{2}, \quad (3.46)$$

a matriz \mathbf{E} pode ser escrita, explicitamente, como

$$\mathbf{E} = \begin{pmatrix} b & -a & 0 & 0 & \cdots & 0 \\ -a & b & -a & 0 & \cdots & 0 \\ 0 & -a & b & -a & \cdots & 0 \\ \cdots & & & & & \\ 0 & 0 & 0 & \cdots & -a & b \end{pmatrix} \quad (3.47)$$

Necessitamos ainda, conforme Eq.(3.31), do operador

$$\mathbf{M} = 1 + \frac{dt}{2} L \mathbf{r} = 1 + \frac{D dt}{2} \frac{\partial^2}{\partial x^2} - \frac{S dt}{2} \quad (3.48)$$

cuja forma matricial, explicitamente, é

$$\mathbf{M} = \begin{pmatrix} c & a & 0 & 0 & \cdots & 0 \\ a & c & a & 0 & \cdots & 0 \\ 0 & a & c & a & \cdots & 0 \\ \cdots & & & & & \\ 0 & 0 & 0 & \cdots & a & c \end{pmatrix} \quad (3.49)$$

onde

$$c = 1 - \frac{D dt}{2 dx^2} - \frac{S dt}{2}.$$

Usando como condições de contorno $f_0^n = f_{N+1}^n = 0$, a solução da Eq.(3.10), discretizada pelo algoritmo de Crank-Nicholson, adquire a forma, cf. Eq.(3.34),

$$\mathbf{f}^{n+1} = \mathbf{E}^{-1} \cdot (\mathbf{M} \cdot \mathbf{f}^n + dt \mathbf{s}). \quad (3.50)$$

O programa “defeitos_CN.sce”, a seguir, implementa o procedimento acima para a dinâmica dos “defeitos de radiação”, por Crank-Nicholson, usando como condição inicial $f(x, t = 0) = 0$;

Programa: defeitos_CN.sce

```

// Eq. de difusão de defeitos com fonte e sumidouros
// pelo esquema Crank-Nicholson; intensidade s0,
// posição central x0 e largura sigma da fonte,
// concentração S de sumidouros e largura L da amostra
// são "input"\e tomamos D=1. Inclui solução
// estacionária. Ex: [10,70,10,.0003,300,10000,50];

par=input('deh [s0,x0,sigma, S, L, tmax, dt] -\n');
s0=par(1); x0=par(2); sigma=par(3); S=par(4);
L=par(5); tmax=par(6); dt=par(7);
s2=1/(2*sigma^2);
nt=floor(tmax/dt); // T é o num. de pontos temporais
x=(1:L)'; // grade espacial, vetor coluna
f=zeros(L,1); // concentração inicial de defeitos
s=s0*exp(-((x-x0).^2)*s2); // distribuição da fonte
// solução estacionaria
bb=2+S;
for j=1:L;
    M(j,j)=bb;
end
for j=1:L-1
    M(j,j+1)=-1;
    M(j+1,j)=-1;
end
m=inv(M);
fes=m*s;
r=0.7*max(fes); rs=r*s/s0; //renormaliza s para plotar
// matrizes do método CN
h=dt/2; b=1+h*(2+S); B=1-h*(2+S); a=-h;
E=zeros(L); M=zeros(L);
for j=1:L;
    E(j,j)=b;
    M(j,j)=B;
end
for j=2:L;
    E(j,j-1)=a;
    M(j,j)=B;
end

```

```

E(j-1,j)=a;
M(j,j-1)=h;
M(j-1,j)=h;
end
IE=inv(E);
// evolução temporal
while(tmax >= 0)
for it=1:nt;
f=IE*(M*f+dt*s);
end
// gráfico dos resultados:
plot(x,rs,x,f,x,fes,'--')
tmax=input('deh novo tempo; se enter encerra \n');
nt=floor(tmax/dt);
end

```

Exercício 3.5:

Entenda o programa “defeitos_CN”, execute-o com diversos valores dos parâmetros, especialmente diversos dt , introduza uma medida de tempo de CPU na parte dependente do tempo e compare com o tempo de CPU necessário para o programa “defeitos” (seção 3.1.3) produzir resultados comparáveis.

3.1.6 Equação de Schrödinger

Na notação usual da Mecânica Quântica, a equação de Schrödinger se escreve na forma

$$H\Psi = i\hbar \frac{\partial \Psi}{\partial t}. \quad (3.51)$$

Na representação de posição, o Hamiltoniano H é um operador em \mathbf{r} , onde \mathbf{r} pode ser o conjunto de coordenadas das partículas do sistema. Escolhendo-se um conjunto completo de funções $\{\phi_j(\mathbf{r})\}$, H pode ser escrito como uma matriz quadrada de elementos

$$H_{ij} = \langle \phi_i | H | \phi_j \rangle = \int \phi_i^*(\mathbf{r}) H \phi_j(\mathbf{r}) d\mathbf{r} \quad (3.52)$$

e a função de onda

$$\Psi(\mathbf{r}, t) = \sum_j C_j(t) \phi_j(\mathbf{r}), \quad (3.53)$$

como vetor

$$\Psi = \begin{pmatrix} C_1(t) \\ C_2(t) \\ C_3(t) \\ \vdots \\ \vdots \\ C_N(t) \end{pmatrix} \quad (3.54)$$

Uma vez conhecida a matriz H e a Ψ inicial, isto é, os $C_j(0)$, a aplicação de Crank-Nicholson é trivial. Note que a Eq.(3.51) é igual à Eq.(3.26) com

$$L_r = \frac{-i}{\hbar} H \quad (3.55)$$

e $s = 0$. A Eq.(3.31) fica

$$\Psi^{n+1} = \left(1 + \frac{idt}{2\hbar} H \right)^{-1} \left(1 - \frac{idt}{2\hbar} H \right) \Psi^n \quad (3.56)$$

Exemplo 3.5: Partícula livre no espaço unidimensional, com pacote de onda inicial Gaussiano

O Hamiltoniano de uma partícula livre, em uma dimensão espacial, é

$$H = -\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2}. \quad (3.57)$$

No espaço discretizado

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_N \end{pmatrix} \quad (3.58)$$

uma base $\{\phi_j\}$ pode ser

$$\phi_j(x) = \begin{cases} 1 & \text{para } x = x_j \\ 0 & \text{para } x \neq x_j \end{cases} \quad (3.59)$$

Além disso, vamos usar a condição de contorno periódica, com período L , isto é,

$$\phi_j(x + L) = \phi_j(x),$$

de modo que a Eq.(3.59) deve ser substituída por

$$\begin{aligned}\phi_j(x_i) &= \delta_{i,j} \\ \phi_L(x_{i-1}) &= \delta_{i,1} \\ \phi_1(x_{i+1}) &= \delta_{i,L}\end{aligned}\quad (3.60)$$

onde $1 \leq i \leq L$ e $1 \leq j \leq L$. As duas últimas linhas da Eq.(3.60) dão conta da condição de contorno periódica.

Os elementos de matriz do Hamiltoniano, nesta base, são

$$\begin{aligned}H_{ij} &= -\frac{\hbar^2}{2m} \int \phi_i^*(x) \frac{\partial^2}{\partial x^2} \phi_j(x) dx \\ &= -\frac{\hbar^2}{2m} \sum_k \phi_i^*(x_k) \{ \phi_j(x_{k+1}) + \phi_j(x_{k-1}) - 2\phi_j(x_k) \\ &= -\frac{\hbar^2}{2m} \{ \phi_j(x_{i+1}) + \phi_j(x_{i-1}) - 2\phi_j(x_i)\}\end{aligned}\quad (3.61)$$

onde, na última linha, usamos a primeira linha da Eq.(3.60) para $\phi_i^*(x_k)$.

Usando novamente a Eq.(3.60) a matriz de H adquire a forma

$$H = -\frac{\hbar^2}{2m} \begin{pmatrix} -2 & 1 & 0 & 0 & \cdots & 1 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ \cdots & & & & & \\ & & & & & \\ 1 & 0 & 0 & \cdots & 1 & -2 \end{pmatrix} \quad (3.62)$$

Seja

$$\Psi(x, 0) = \frac{1}{\sqrt{\sigma} \sqrt{\pi}} e^{ik_0 x} e^{-(x-x_0)^2/2\sigma^2} \quad (3.63)$$

o pacote de onda inicial da partícula, localizado em torno de x_0 e com momentum em torno de $p_0 = \hbar k_0$.

Notas:

1) A normalização é tal que $\int_{-\infty}^{+\infty} |\Psi|^2 dx = 1$;

2) O pacote, Eq.(3.63), tem $\Delta x \Delta p = \hbar/2$;

3) A solução exata é

$$\Psi(x, t) = \frac{1}{\sqrt{\sigma \sqrt{\pi}} \alpha} e^{ik_0(x-p_0 t/2m)} e^{-(x-x_0-p_0 t/m)^2/2\alpha^2} \quad (3.64)$$

onde $\alpha^2 = \sigma^2 + i\hbar t/m$, $p_0 = \hbar k_0$.

O valor esperado de $x(t)$ é

$$\langle x(t) \rangle = x_0 + \frac{p_0 t}{m} \quad (3.65)$$

isto é, o centro do pacote move-se com velocidade p_0/m .

Vamos usar unidades tais que $\hbar = m = 1$ e definir a matriz

$$E = 1 + \frac{idt}{2} H = \begin{pmatrix} b & -a & 0 & 0 & \cdots & -a \\ -a & b & -a & 0 & \cdots & 0 \\ 0 & -a & b & -a & \cdots & 0 \\ \cdots & & & & \cdots & \\ -a & 0 & 0 & \cdots & -a & b \end{pmatrix} \quad (3.66)$$

onde $a = idt/4$ e $b = 1 + 2a$.

A Eq.(3.56) fica, neste caso,

$$\Psi^{n+1} = E^{-1} E^* \Psi^n \quad (3.67)$$

onde E^* é a complexa conjugada à E . Notemos a diferença entre a Eq.(3.46) e a Eq.(3.66). Na última $E_{1,L}$ e $E_{L,1}$ são não nulos, iguais a $-a$. Isto é por causa da condição de contorno periódica.

O programa “schroedinger.sce”, a seguir, implementa o exemplo acima.

Programa: schroedinger.sce

```
// Equação de Schrödinger de partícula livre pelo
// esquema Crank-Nicolson; m=1, hbar=1; condição
```

```
// de contorno periódica; Exemplo:  
// [k=.5, x0=50, sigma=10, N=300, tmax=400, dt=2]  
  
par=input('deh [k,x0,sigma, N, tmax, dt] -\n');  
k=par(1); x0=par(2); sigma=par(3); N=par(4);  
tmax=par(5); dt=par(6);  
pi=%pi; i=%i;  
K=1/(sqrt(sigma*sqrt(pi)));  
s2=1/(2*sigma^2);  
nt=floor(tmax/dt)+1; // nt é o num. de pontos temporais  
x=1:N; x=x';  
c=K*exp(i*k*(x-x0)).*exp(-((x-x0).^2)*s2);  
C=zeros(N,3);  
C(:,1)=c;  
h=dt/2; ih=i*h; mih=-0.5*ih; b=1+ih;  
E=zeros(N);  
for j=1:N;  
    E(j,j)=b;  
end  
for j=2:N;  
    E(j,j-1)=mih;  
    E(j-1,j)=mih;  
end  
E(1,N)=mih; E(N,1)=mih;  
IEM=inv(E)*conj(E);  
pt=ceil(nt/2);  
for tp=2:3;  
    for it=1:pt;  
        c=IEM*c;  
    end  
    C(:,tp)=c;  
end  
xset('window',1); clf  
plot(x,real(C(:,1)),x,real(C(:,2)),x,real(C(:,3)))  
legend('Re Psi(0)', 'Re Psi(0.5 tmax)', 'Re Psi(tmax)')  
xset('window',2);clf  
plot(x,imag(C(:,1)),x,imag(C(:,2)),x,imag(C(:,3)))
```

```

legend('Im Psi(0)', 'Im Psi(0.5 tmax)', 'Im Psi(tmax)')
xset('window',3); clf
plot(x,abs(C(:,1)),x,abs(C(:,2)),x,abs(C(:,3)))
legend('Abs(Psi(0))', 'Abs(Psi(0.5 tmax))', ...
'Abs(Psi(tmax))')

```

Exercício 3.6:

Entenda o programa “schroediger.sce” e execute-o com diferentes parâmetros.

3.2 Métodos de Relaxação em Duas ou Mais Dimensões Espaciais

Vamos considerar equações do tipo da Eq.(3.1), mas sem o termo em derivada temporal e com a variável espacial x substituída pelo vetor posição \mathbf{r} . Por simplicidade usaremos $\mathbf{r} = (x, y)$, i.e., apenas duas dimensões, mas a extensão das equações abaixo para três dimensões é, em geral, trivial. Portanto, trataremos de encontrar soluções numéricas para equações do seguinte tipo:

$$a_x \frac{\partial^2 f}{\partial x^2} + a_y \frac{\partial^2 f}{\partial y^2} + b_x \frac{\partial f}{\partial x} + b_y \frac{\partial f}{\partial y} - c f + s = 0. \quad (3.68)$$

Quando $a_x = a_y = a$ =constante e $b = c = 0$ temos a equação de Laplace,

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0. \quad (3.69)$$

Este é o conhecido problema de eletrostática de encontrar o potencial no interior de uma região quando ele é conhecido no contorno da mesma, ou o problema termodinâmico de encontrar a temperatura, nas mesmas condições. Soluções exatas só são conhecidas em casos de contornos com simetrias muito simples, como círculos ou retângulos, em duas dimensões, ou esferas, cilindros ou paralelepípedos, em três dimensões, com valores de f também dados por funções muito simples sobre o contorno. Soluções numéricas podem ser encontradas para condições de contorno bem mais gerais, para o que existem

diversos métodos. Neste texto apresentamos apenas os chamados “métodos de relaxação”.

Como a solução $f(x, y)$ da Eq.(3.69), não depende do tempo, podemos escrevê-la na forma

$$\frac{\partial f}{\partial t} = a \left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \right), \quad (3.70)$$

pois o membro à esquerda é nulo. Reconhecemos a Eq.(3.70) como a equação da difusão e sua solução estacionária $f(x, y)$ é a solução da equação de Laplace. A ideia do método consiste em encontrar a solução $f(x, y, t)$ da Eq.(3.70) a partir de uma f inicial escolhida arbitrariamente, $f(x, y, 0)$, que satisfaça as condições de contorno dadas, até um tempo t_f , tal que $f(x, y, t_f) = f(x, y, t_f - dt)$, dentro de uma tolerância pré-estabelecida. Como não estamos interessados na evolução temporal da solução, mas apenas no limite estacionário, não precisamos de um método preciso em relação à dependência temporal. Por isso usamos FTCS:

$$f_{i,j}^{n+1} = f_{i,j}^n + \frac{a \, dt}{dx^2} \left(f_{i+1,j}^n + f_{i-1,j}^n + f_{i,j+1}^n + f_{i,j-1}^n - 4f_{i,j}^n \right) \quad (3.71)$$

onde os índices i e j referem-se às coordenadas x e y , respectivamente, e usamos $dy = dx$.

3.2.1 Algoritmo de Jacobi em Duas Dimensões Espaciais

Vimos que a condição de estabilidade numérica do método FTCS em uma dimensão é: $a \, dt / dx^2 \leq 1/2$. Em duas dimensões espaciais a condição de estabilidade é

$$\frac{a \, dt}{dx^2} + \frac{a \, dt}{dy^2} \leq \frac{1}{2}.$$

Como escolhemos $dx = dy$, o máximo valor de dt que satisfaz a condição de estabilidade é tal que

$$\frac{a \, dt}{dx^2} = \frac{1}{4}.$$

Com esta escolha, escrevemos a Eq.(3.71) como

$$f_{i,j}^{n+1} = \frac{1}{4} \left(f_{i+1,j}^n + f_{i-1,j}^n + f_{i,j+1}^n + f_{i,j-1}^n \right). \quad (3.72)$$

Portanto, entendendo os valores de $f_{i,j}^n$, para os sucessivos n , como os sucessivos passos de aproximação ao valor estacionário, vemos que o valor de $f_{i,j}$ no passo $n+1$ é a média aritmética dos valores dos quatro vizinhos mais próximos no passo anterior. A Eq.(3.72) é aplicável somente aos pontos interiores, i.e., não vale para pontos sobre o contorno, onde os valores de f são dados pelas condições de contorno. Este procedimento que acabamos de descrever se chama *Algoritmo de Jacobi*.

Exemplo 3.6:

No programa “Jacobi2d.sce”, a seguir, resolvemos a equação de Laplace pelo algoritmo de Jacobi. A geometria do espaço é um quadrado de lado $L=50$. As condições de contorno são: $f(x = 0, y) = 1$; $f(x = 50, y) = 0$; $f(x, y = 0) = \exp(-\lambda_1 x)$; $f(x, y = 50) = \exp(-\lambda_2 x)$, onde λ_1 e λ_2 são dados pelo operador. A cada dez passos de integração verifica-se de quanto variou f em relação à última verificação. A máxima diferença ($\max(f_{t+10}(x, y) - f_t(x, y))$) é chamada de erro e o processo se encerra quando o erro for menor que a tolerância (tol) dada no “input”. O gráfico tri-dimensional (mesh) de $f(x, y)$ é apresentado e o número de vezes que o erro foi observado (número de passos de integração dividido por 10) é informado.

Programa: jacobi2d.sce

```
// Solução da Eq. de Laplace em 2d, pelo método de
// relaxação, algoritmo de Jacobi. Geometria:
// quadrado de lado L e condições de contorno:
// f(0,y)=1, f(L,y)=0, f(x,0)=exp(-lam1 x),
// f(x,L)=exp(-lam2 x); escolhemos dx=dy=1, L=50;
// exemplo de dados: [lam1=.1, lam2=.3, tol=0.0001]

clear
par=input('deh [lam1, lam2, tol], \n');
lam1=par(1); lam2=par(2); tol=par(3);
```

```

f=zeros(51,51);
x=(0:50)'; y=0:50;

// condições de contorno:
f(1,:)=1;
f(51,:)=0;
f(:,1)=exp(-lam1*x);
f(:,51)=exp(-lam2*x);

// algoritmo de Jacobi
m=0; erro=2*tol;
while(erro>tol)
    m=m+1;
    g=f;
    for n=1:10;
        f(2:50, 2:50)=.25*(f(1:49,2:50)+f(3:51,2:50)+ ...
        f(2:50,1:49)+f(2:50,3:51));
    end
    ....
    erro=max(abs(f-g));
end
m
xset('window',1); clf
surf(x,y,f')

```

Exercício 3.7:

Após entender e executar o programa “Jacobi2d.sce”, modifique as suas condições de contorno; por exemplo, use $\cos(3\pi x/5)$ de $\exp(-\lambda_1 x)$.

modifique as
0) em lugar

ensões

espaciais é
média entre
dimensional,
mo obtemos
a apresentar

3.2.2 Algoritmo de Jacobi em Três Dimensões Espaciais

A extensão do algoritmo de jacobi para três dimensões é bastante simples. No processo de relaxação, em vez da média entre os valores de f nos 4 sítios vizinhos, usada no caso bidimensional, a média agora é realizada entre os 6 sítios vizinhos. Como uma função definida num espaço de três dimensões, para

os resultados em forma gráfica precisamos definir cortes em forma de planos no espaço da amostra e graficar o valor da f nesses planos. Isto é executado no programa a seguir. Os gráficos são apresentados em três figuras separadas.

Programa: Jacobi3d.sce

```
// Solução da Eq. de Laplace em 3d, pelo método de
// relaxação, algoritmo de Jacobi. Geometria:
// cubo de lado L e condições de contorno:
// f(0,y,z)=1, f(L,y,z)=0, f(x,0,z)=exp(-lam1 x),
// f(x,L,z)=exp(-lam2 x); escolhemos dx=dy=dz=1, L=30;
// exemplo de dados: [lam1=.2, lam2=.3, tol=0.0001]

par=input('deh [lam1, lam2, tol], \n');
lam1=par(1); lam2=par(2); tol=par(3);
f=zeros(31,31,31);
x=(0:30)'; y=0:30; z=(0:30)';
// condições de contorno:
f(1,:,:)=1;
f(31,:,:)=0;
for j=2:30
f(:,1,j)=exp(-lam1*x);
f(:,31,j)=exp(-lam2*x);
end
// algoritmo de Jacobi
m=0; erro=2*tol;
while(erro>tol)
    m=m+1;
    g=f;
    c=1/6;
    for n=1:10;
        for j=2:30
            f(2:30, 2:30,j)=c*(f(1:29,2:30,j)+f(3:31,2:30,j)+...
            f(2:30,1:29,j)+ f(2:30,3:31,j)+f(2:30,2:30,j-1)+ ...
            f(2:30,2:30,j+1));
        end
    end
end
```

```

    erro=max(abs(f-g));
end
m
F1=zeros(31,31); F2=F1; F3=F1;
F3=f(:,:,2);
for j=1:31
    for k=1:31
        F1(j,k)=f(10,j,k);
        F2(j,k)=f(j,2,k);
    end
end
xset('window',1);clf; surf(y',z,F1')
legend('um corte perpendicular ao eixo x',2)
xset('window',2);clf; surf(x,z,F2)
legend('um corte perpendicular ao eixo y',2)
xset('window',3);clf; surf(x,y,F3')
legend('um corte perpendicular ao eixo z',2)

```

3.2.3 Algoritmo de Gauss-Seidel

Consiste em usar, na determinação de $f_{i,j}^{n+1}$ os valores de f^{n+1} que já foram calculados, ou seja, a Eq.(3.72) fica substituída por

$$f_{i,j}^{n+1} = \frac{1}{4} \left(f_{i+1,j}^n + f_{i-1,j}^{n+1} + f_{i,j+1}^n + f_{i,j-1}^{n+1} \right). \quad (3.73)$$

Com isto se consegue convergência com um número menor de passos, em torno da metade, que com Jacobi. No caso de programação com SCILAB ou MATLAB, entretanto, perde-se em Gauss-Seidel a possibilidade de vetorizar o cálculo de f^{n+1} a partir de f^n ; isto pode prolongar bastante o tempo de CPU. Por outro lado, dentro de um “for” (ou “do”, em FORTRAN), como não se tem necessidade de associar a variável n à função $f_{i,j}$, o algoritmo de Gauss-Seidel será executado naturalmente, pois cada vez que $f_{i,j}$ for usado, à direita do sinal $=$, é exatamente o último valor calculado que estará disponível.

3.2.4 Algoritmo “Super-relaxação” (Overrelaxation)

Trata-se de introduzir uma modificação no algoritmo de Gauss-Seidel que pode acelerar em muito a convergência para a solução estacionária. A ideia é alterar a média à direita da igualdade na Eq.(3.73), introduzindo um peso maior que 1 para a média dos vizinhos e incluindo o próprio $f_{i,j}$ com peso negativo (a soma dos pesos continua 1), ou seja:

$$f_{i,j}^{n+1} = -\alpha f_{i,j}^n + \frac{1+\alpha}{4} (f_{i+1,j}^n + f_{i-1,j}^n + f_{i,j+1}^n + f_{i,j-1}^n). \quad (3.74)$$

Para $\alpha = 0$ temos o algoritmo de Gauss-Seidel, para $\alpha < 0$ temos “sub-relaxação” (demora mais a convergir) e para $\alpha > 0$ temos “super-relaxação”. Para $\alpha \geq 1$ o algoritmo é instável. Geralmente um valor de α próximo de um resulta estável e com boa convergência, mas em alguns casos pode ser mais seguro um valor menor. Alguns testes em cada caso indicarão o valor ótimo de α .

Exercício 3.8:

Modifique o programa "Jacobi2d.sce", acrescentando solução pelos algoritmos de Gauss-Seidel e de Super-relaxação. Acrescente a possibilidade de medir o tempo de CPU usado por cada um dos três algoritmos. Compare os tempos de CPU no caso de os parâmetros serem os mesmos (use diferentes α , como 0, 0.5, 0.8, 0.9)

Observações:

- 1) É claro que os algoritmos de Jacobi, Gauss-Seidel e Super-relaxação convergem para a mesma solução estacionária, pois quando $f^{n+1} = f^n$ as equações Eq.(3.72), Eq.(3.73) e Eq.(3.74) são iguais;
- 2) No exercício 3.9, usando $\alpha = 0.9$, o algoritmo de Super-relaxação necessita um número de passos em torno de 70 vezes menor que Jacobi para resultar em convergência comparável. No caso de se usar uma linguagem de programação para a qual a vetorização não implica em aceleração muito grande, o algoritmo de Super-relaxação torna-se altamente vantajoso;
- 3) Pode parecer tentador usar o tipo de média do algoritmo de Super-relaxação com o algoritmo de Jacobi (em vez de usá-lo com Gauss-

Seidel), obtendo assim as duas vantagens, vetorização e rapidez de convergência. Entretanto, como em Jacobi os valores de f^n para n ímpar são calculados usando exclusivamente os valores correspondentes a n par, e vice-versa, o processo de aceleração da relaxação resulta em duas linhas separadas e divergentes de soluções, uma para n pares e outra para n ímpares.

4) Usar os algoritmos acima para a equação de Poisson ou outra do tipo da Eq.(3.68) não apresenta nenhuma dificuldade extra: basta acrescentar ao membro à direita das respectivas equações discretizadas os termos que lhes faltam para representarem as equações de interesse.

Exercício 3.9:

Introduza nos programas do exercício 3.9, ou em “Jacobi2d.sce”, um termo de fonte, isto é, substitua = 0 na Eq.(3.69) por = $\rho(x, y)$, sendo $\rho(x, y)$ uma função dada; com isto você estará resolvendo a equação de Poisson em vez da de Laplace.

Capítulo 4

PROBABILIDADE, VARIÁVEL ALEATÓRIA E PROCESSO ESTOCÁSTICO

Com vistas à preparação para os próximos capítulos, apresentamos neste, muito resumidamente, aspectos fundamentais da Teoria de Probabilidade, Variáveis Aleatórias e Processos Estocásticos

4.1 Introdução à Teoria de Probabilidade

4.1.1 O Conceito de Probabilidade

Consideremos um experimento com as seguintes características:

- 1) Há um conjunto discreto de resultados possíveis para o mesmo, que denotaremos por $R_1, R_2, \dots, R_j, \dots, R_n$;
- 2) É impossível prever qual dos resultados ocorrerá em uma realização do experimento;
- 3) A “chance” de ocorrer um determinado resultado permanece constante em realizações sucessivas do experimento.

Nestas circunstâncias a experiência nos diz que realizando-se o experimento um número N muito grande de vezes a frequência média, n_j/N , de ocorrência de um dado resultado, R_j , onde n_j é o número de vezes que ocorre o resultado R_j , tende a um valor constante, independente de N . Definimos a probabilidade do resultado R_j por

$$P(R_j) = \lim_{N \rightarrow \infty} \frac{n_j}{N}. \quad (4.1)$$

No contexto da teoria de probabilidade os possíveis resultados de um experimento são chamados *eventos*. A especificação dos eventos não é única. Por exemplo, no lançamento de um dado, resultar o número 5 para cima é um evento, mas resultar um número par ou um número ímpar também são eventos. O evento “maior que 3” e o evento “ímpar” ocorrem simultaneamente se o resultado for o número 5. Quando dois eventos não podem ocorrer simultaneamente, como os eventos “par” e “ímpar”, eles são ditos *disjuntos* ou *mutuamente excludentes*. Pode-se usar para os eventos a terminologia da teoria de conjuntos. Assim, a união dos eventos “ímpar” e “maior que 3” é o evento $[1,3,4,5,6]$ e a intersecção dos mesmos é o evento 5. Um conjunto de eventos é *completo* ou *exaustivo* quando qualquer evento está contido em um evento do conjunto. Eventos que não são a união de outros eventos são ditos *elementares*. Assim, no caso do dado, os eventos 1, 2, 3, etc., são elementares mas os eventos “par” e “ímpar” não. Da definição de probabilidade, Eq.(4.1), seguem, trivialmente, algumas propriedades, como veremos. Para isso vamos introduzir a seguinte notação: Probabilidade de ocorrer simultaneamente os eventos A e B = $P(A, B)$; probabilidade de ocorrer A ou B ou ambos = $P(A \text{ ou } B)$; para qualquer evento A , se tem $0 \leq P(A) \leq 1$. Se A e B são eventos disjuntos, então $P(A \text{ ou } B) = P(A) + P(B)$. Se existe intersecção entre A e B , então a soma $P(A) + P(B)$ conta duas vezes os resultados em que ocorrem A e B . Por isso, em geral,

$$P(A \text{ ou } B) = P(A) + P(B) - P(A, B). \quad (4.2)$$

Se R_1, R_2, \dots, R_n são todos os eventos elementares de um experimento, então

$$\sum_{j=1}^n P(R_j) = 1. \quad (4.3)$$

Esta relação é fundamental em teoria de probabilidade e é válida também para um conjunto de eventos não elementares, desde que seja completo e seus elementos sejam disjuntos.

Exercício 4.1: Provar que a Eq.(4.3) é válida para qualquer conjunto completo de eventos disjuntos.

Seja Ω a união de todos os eventos (o conjunto de todos os resultados possíveis) e Φ o conjunto vazio de eventos. Então

$$P(\Omega) = 1, \text{ e } P(\Phi) = 0.$$

Se \bar{A} é o complemento de A , isto é, o conjunto de todos eventos não contidos em A , então

$$P(\bar{A}) = 1 - P(A).$$

4.1.2 Probabilidade Condicional

Fala-se em “Probabilidade de A dado que B ” e simboliza-se por $P(A | B)$ para indicar a probabilidade de que o resultado seja A sabendo-se que é B (obviamente $P(A | B)$ só será não nula se existe intersecção de A com B). A probabilidade condicional $P(A | B)$ está relacionada com a probabilidade conjunta $P(A, B)$ por

$$P(A, B) = P(A | B)P(B) = P(B | A)P(A). \quad (4.4)$$

Exemplo 4.1: Consideremos novamente o dado de seis faces equivalentes;

$$P(\geq 4 | par) = 2/3,$$

pois existem 3 resultados pares equivalentes e dois deles são ≥ 4 . Além disso,

$$P(\geq 4, par) = P(\geq 4 | par)P(par).$$

Com efeito,

$$\frac{2}{6} = \frac{2}{3} \times \frac{1}{2}.$$

Consideremos novamente o conjunto completo de eventos disjuntos, R_1, R_2, \dots, R_n . Então, sendo A um evento qualquer,

$$\sum_{j=1}^n P(A | R_j) P(R_j) = \sum_{j=1}^n P(A, R_j) = P(A). \quad (4.5)$$

4.1.3 Eventos Independentes

Dois eventos A e B são “independentes” se

$$P(A, B) = P(A) P(B), \quad (4.6)$$

ou, equivalentemente,

$$P(A | B) = P(A) \text{ e } P(B | A) = P(B). \quad (4.7)$$

Assim, no caso do dado, os eventos ≥ 3 e *par* são independentes, pois $P(\geq 3, \text{par}) = 2/6$, $P(\geq 3) = 4/6$ e $P(\text{par}) = 1/2$, o que satisfaz a Eq.(4.6). O leitor pode verificar a Eq.(4.7) para este exemplo.

4.2 Variável Aleatória

Uma variável, V , cujo valor não pode ser previsto com segurança antes de sua observação, ou seja, em repetições do mesmo experimento aleatório em idênticas condições ela assume valores diferentes, se chama “variável aleatória”. Por exemplo, o número que fica para cima no lançamento de um dado é uma variável aleatória, assim como uma função qualquer desse número. Uma variável aleatória pode ser “discreta”, como no exemplo do dado, quando seus possíveis valores formam conjunto discreto, ou “contínua”, como a posição de uma partícula em movimento Browniano, em que os possíveis valores pertencem a um conjunto contínuo. Inicialmente trataremos de variáveis aleatórias discretas.

4.2.1 Média

Se R_1, R_2, \dots, R_n formam um conjunto completo de eventos disjuntos de um experimento, tais que a variável aleatória $V(R)$ possui um valor determinado para cada evento R_j , definimos o “valor médio de V ”, ou simplesmente “média de V ”, por

$$\langle V \rangle = \sum_{j=1}^n P(R_j) V(R_j). \quad (4.8)$$

Para simplificar a notação vamos representar $V(R_j)$ e $P(R_j)$, respectivamente por V_j e P_j . Assim a Eq.(4.8) será escrita como

$$\langle V \rangle = \sum_j P_j V_j. \quad (4.9)$$

Uma maneira alternativa de calcular a média, importante em muitas ocasiões, como veremos, é reunir os possíveis valores distintos de V , digamos v_1, v_2, \dots (note que os $V_j \equiv V(R_j)$ podem ser iguais) e definir a probabilidade de ocorrência de cada um desses valores:

$$P_V(v) = \sum_{j, \text{tais que } V_j=v} P_j. \quad (4.10)$$

Dessa forma a média pode ser escrita como

$$\langle V \rangle = \sum_v P_V(v)v. \quad (4.11)$$

Com esta notação a “probabilidade conjunta” de que a variável X tenha o valor x e a variável Y tenha o valor y será representada por $P_{XY}(x, y)$. Se X e Y forem variáveis aleatórias independentes, então

$$P_{XY}(x, y) = P_X(x)P_Y(y). \quad (4.12)$$

A média de um produto de variáveis, X e Y , pode ser escrita de duas maneiras distintas, mas equivalentes, como segue:

$$\langle XY \rangle = \sum_j P_j X_j Y_j = \sum_{x,y} P_{XY}(x, y) x y. \quad (4.13)$$

Para que a primeira das formas acima seja correta é necessário que os eventos sejam definidos de tal maneira que a cada evento R_j corresponda um e só um par de valores X_j, Y_j .

4.2.2 Momentos, Variância e Covariância

O momento de ordem n (ou n^{mo} momento) de uma variável aleatória V é definido por

$$M_n(V) = \langle V^n \rangle = \sum_j P_j V_j^n \quad (4.14)$$

Quando o primeiro momento, ou média, $\langle V \rangle$, é diferente de zero, é usual definir os momentos "em torno da média", por

$$\mathcal{M}_n(V) = \langle (V - \langle V \rangle)^n \rangle = \sum_j P_j (V_j - \langle V \rangle)^n. \quad (4.15)$$

O segundo momento em torno da média tem importância especial, como veremos, e se chama "variância":

$$\text{var}(V) \equiv \sigma_V^2 \equiv \langle (V - \langle V \rangle)^2 \rangle = \langle V^2 \rangle - \langle V \rangle^2. \quad (4.16)$$

Deixamos ao leitor a demonstração da última igualdade da Eq.(4.16). Chama-se "desvio padrão" ou "largura da distribuição" à raiz quadrada da variância:

$$\sigma_V = \sqrt{\sigma_V^2} = \sqrt{\langle [V - \langle V \rangle]^2 \rangle}. \quad (4.17)$$

Semelhantemente, define-se a "covariância" de duas variáveis aleatórias, X e Y , por

$$\text{cov}(X, Y) = \langle X, Y \rangle = \langle (X - \langle X \rangle)(Y - \langle Y \rangle) \rangle = \langle XY \rangle - \langle X \rangle \langle Y \rangle. \quad (4.18)$$

A covariância também costuma ser chamada de "correlação" entre as variáveis envolvidas, mas alguns autores preferem reservar a palavra correlação para quando se usa a seguinte normalização:

$$\text{corr}(X, Y) = \frac{\langle X, Y \rangle}{\sigma_X \sigma_Y} = \frac{\langle (X - \langle X \rangle)(Y - \langle Y \rangle) \rangle}{\sigma_X \sigma_Y} \quad (4.19)$$

Exercício 4.2: Mostrar que se X e Y são proporcionais, i.e., $Y = \alpha X$, com $\alpha > 0$, $\text{corr}(X, Y) = 1$ e se $\alpha < 0$, $\text{corr}(X, Y) = -1$ e se X e Y forem independentes então $\text{corr}(X, Y) = 0$.

4.2.3 Variável Aleatória Contínua

Vamos considerar uma variável aleatória contínua X , real, que pode assumir qualquer valor de $-\infty$ a $+\infty$. Neste caso não faz sentido perguntar "qual é a probabilidade de X assumir um dado valor x ". O que faz sentido é a probabilidade de X assumir um valor num dado

intervalo, $a \leq X \leq b$. Em particular, tem importância especial a probabilidade de X assumir um valor menor que um dado x ,

$$F_X(x) \equiv P(-\infty < X \leq x) , \quad (4.20)$$

que é uma função monótona crescente de x e se chama “função distribuição de X ”. Define-se a “densidade de probabilidade” de X como a derivada em relação a x da função distribuição:

$$f_X(x) = \frac{dF_X(x)}{dx} . \quad (4.21)$$

Se dx é um intervalo infinitesimal em torno de x , a quantidade $f_X(x) dx$ é a probabilidade de X assumir um valor em dx .

Exercício 4.3: Usando as definições acima, Eq.(4.20) e Eq.(4.21), e supondo $a < b$, mostre que

$$\int_a^b f_X(x) dx = P(a \leq X \leq b) . \quad (4.22)$$

Todas as relações e propriedades enunciadas acima, para a probabilidade de variáveis aleatórias discretas, têm correspondentes na densidade de probabilidade das variáveis contínuas. Assim a “condição de normalização”, Eq.(4.3), fica, no caso contínuo,

$$\int_{-\infty}^{+\infty} f_X(x) dx = 1 . \quad (4.23)$$

A relação entre probabilidade conjunta e probabilidade condicional, Eq.(4.4), corresponde, no caso contínuo, à relação entre a densidade de probabilidade conjunta de duas variáveis, X e Y , e a densidade condicional de X , dado que $Y = y$, ou vice-versa:

$$f_{X,Y}(x, y) = f_{X|Y}(x | y) f_Y(y) = f_{Y|X}(y | x) f_X(x) . \quad (4.24)$$

O “traço” sobre uma das variáveis (i.e., soma sobre todos os valores possíveis da mesma) na probabilidade conjunta dá a probabilidade da outra, cf. Eq.(4.5); no caso de variáveis contínuas, o traço é substituído por integral:

$$\int_{-\infty}^{+\infty} f_{X,Y}(x,y) dy = f_X(x) . \quad (4.25)$$

Quando duas variáveis são independentes, a relação equivalente à Eq.(4.12) é, no caso contínuo,

$$f_{X,Y}(x,y) = f_X(x)f_Y(y) . \quad (4.26)$$

A definição de média, Eq.(4.11), adquire a forma

$$\langle X \rangle = \int_{-\infty}^{+\infty} f_X(x) x dx . \quad (4.27)$$

A média de um produto de variáveis se escreve na forma

$$\langle XY \rangle = \int \int dx dy f_{X,Y}(x,y) x y \quad (4.28)$$

e o momento de ordem n da densidade de probabilidade f na forma

$$M_n = \int_{-\infty}^{+\infty} dx f_X(x) x^n . \quad (4.29)$$

As demais relações, variância (Eq.(4.16)), desvio padrão (Eq.(4.17)), covariância (Eq.(4.18)) e função de correlação (Eq.(4.19)), escritas simbolicamente em termos de médias, continuam válidas na forma em que se encontram e, por isso, não precisamos repeti-las.

4.2.4 Distribuições Especiais

Algumas distribuições de probabilidade, devido à suas importâncias, levam nomes especiais, indicadores da forma das correspondentes “densidades de probabilidade”.

Distribuição Uniforme

Neste caso a densidade de probabilidade é constante num intervalo $[a, b]$ e nula fora deste intervalo. Em particular, a mais usada das distribuições uniformes é a que vale 1 no intervalo $[0, 1]$ e 0 fora dele, i.e.

$$f_X(x) = \begin{cases} 1 & \text{for } 0 \leq x \leq 1 \\ 0 & \text{for } x < 0 \text{ or } x > 1 \end{cases} \quad (4.30)$$

A função “rand()” de SCILAB gera números aleatórios com esta distribuição uniforme. Para geradores de números aleatórios com outras linguagens de programação, recomendamos usar uma das “receitas” apresentadas nas diferentes versões do livro “Numerical Recipies”[5].

Distribuição Gaussiana (ou Normal)

Neste caso a densidade de probabilidade é uma Gaussiana, centrada em \bar{x} e com largura σ ,

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \frac{(x - \bar{x})^2}{2\sigma^2} \quad (4.31)$$

A função “rand(m,n,’normal’)” de SCILAB gera uma matriz $m \times n$ de números aleatórios com distribuição Gaussiana, centrada em 0 e largura 1, neste caso chamada “distribuição normal”. Para obter-se um distribuição centrada em \bar{x} e com largura σ , basta multiplicar o resultado por σ e somar \bar{x} .

Ocorrência da Distribuição Gaussiana: Teorema Central de Limite

Consideremos uma variável aleatória X que é a soma de muitas variáveis aleatórias independentes Y_j ,

$$X = \sum_1^N Y_j , \quad (4.32)$$

com N muito grande (rigorosamente, deveríamos ter $N \rightarrow \infty$). Por exemplo, o momento magnético de um sólido é a soma de seus momentos magnéticos atômicos. É imediato que o valor médio de X é a soma dos valores médios dos Y_j ,

$$\bar{x} \equiv \langle X \rangle = \sum_{j=1}^N \langle Y_j \rangle = \sum_{j=1}^N \bar{y}_j . \quad (4.33)$$

Além disso, por serem independentes os Y_j , segue:

$$\sigma_X^2 \equiv \langle (X - \langle X \rangle)^2 \rangle = \sum_{j=1}^N \langle (Y_j - \langle Y_j \rangle)^2 \rangle \equiv \sum_{j=1}^N \sigma_j^2 . \quad (4.34)$$

Vamos supor ainda que a variância σ_j^2 de qualquer Y_j seja muito menor que as somas das variâncias. O *Teorema Central de Limite*, que admitiremos sem demonstração[8], diz que, nas circunstâncias mencionadas, a distribuição de X é Gaussiana, i.e., a f_X é dada pela Eq.(4.31). Distribuições Gaussianas ocorrem com grande frequência em Física porque as variáveis extensivas dos sistemas macroscópicos são, geralmente, somas de contribuições atômicas ou moleculares que, se não forem totalmente independentes, podem ser agrupadas em somas parciais, de maneira que X é uma soma de somas parciais, as quais são independentes entre si.

Exercício 4.4:

Gerar um vetor de N números aleatórios Gaussianos, numa distribuição centrada em c e de largura l , dividir o intervalo (aproximado) $[c - 4l, c + 4l]$ em n segmentos de tamanhos iguais e “contar quantos números aleatórios há em cada segmento; “plotar” esses números, tomando como abscissa os centros dos segmentos. Por ex., tome $N = 10000$, $c = 1$, $l = 2$, $n = 21$. Repita o cálculo com distintos valores de N e n . Sugestão: Para programação em SCILAB é conveniente usar a função gráfica “histplot”.

Outras Distribuições Consagradas

Os bons livros de Estatística (veja, por exemplo, [9]) costumam apresentar muitas distribuições consagradas pelo uso em certas áreas do conhecimento. SCILAB possui uma função especial, "grand", que pode ser usada para gerar números aleatórios com diversas distribuições. A forma de chamar é: $r = grand(m, n, 'distib', a, b, \dots)$, que gera uma matriz $m \times n$ de números aleatórios, com distribuição ' $distib$ ', de parâmetros a, b, \dots . As mais importantes são:

`grand(m,n,'nor',Av, Sd)`: distribuição Gaussiana com média Av e desvio padrão Sd;

`grand(m,n,'unf',min, max)`: distrib. uniforme no intervalo [min,max];

`grand(m,n,'uin',min, max)`: gera números inteiros com distribuição uniforme em [min,max];

`grand(m,n,'bet', A, B)`: distribuição beta, com parâmetros A, B;

`grand(m,n,'gam',forma, escala)`: distribuição gamma;

`grand(m,n,'bin',N, p)`: distribuição binomial;
`grand(m,n,'chi', ν)`: distribuição quiquadrado com ν graus de liberdade;
`grand(m,n,'poi', λ)`: distribuição de Poisson com média λ ;
etc. Outras distribuições e mais especificações sobre os parâmetros podem ser encontradas com "help grand".

4.2.5 Geração de Números Aleatórios para Outras Distribuições

Usando a função Distribuição

Seja X uma variável aleatória real, definida no intervalo $[a, b]$ e $f(x)$ sua distribuição de probabilidade, de modo que

$$\int_a^b f(x)dx = 1 \quad (4.35)$$

e seja $F(x)$ a correspondente "função distribuição",

$$F(x) = \int_a^x f(x')dx' . \quad (4.36)$$

Um procedimento conveniente para gerar números aleatórios x com distribuição $f(x)$ é por transformação de números gerados com distribuição uniforme, no intervalo $[0, 1]$ em números com distribuição $f(x)$. O procedimento é como segue: Seja r um número gerado com distribuição uniforme; fazemos

$$r = F(x), \text{ ou seja, } x = F^{-1}(r) . \quad (4.37)$$

Para demonstração desta "receita", veja *Numerical Recipes*, seção 7.2.

Exemplo 4.2:

Consideremos um dipolo elétrico μ em um fluído à temperatura T . Tal dipolo tem direção variável no tempo, devido ao movimento Browniano rotacional. Vamos supor que esteja aplicado um campo eletrostático uniforme E . A energia de interação do dipolo com o campo é

$$V(\theta) = \mu E \cos \theta ,$$

onde θ é o ângulo entre μ e E . Para um sistema em equilíbrio a distribuição de probabilidade para θ é (cf. Cap.VI, seção VI.1)

$$f(\theta) = C \sin \theta \exp(K \cos \theta), \quad (4.38)$$

onde $K = \mu E / k_B T$, k_B é a constante de Boltzmann e

$$C = \frac{K}{2 * \sinh(K)}$$

de maneira que

$$\int_0^\pi f(\theta) d\theta = 1.$$

A função $F(\theta)$, cf. Eq.(4.36), é

$$F(\theta) = \int_0^\theta C \sin(\theta') \exp(K \cos \theta') d\theta' = \frac{C}{K} (\exp(K) - \exp(K \cos \theta)). \quad (4.39)$$

Usando a Eq.(4.37) segue

$$\theta = \arccos \left(\frac{1}{K} \log \left(\exp(K) - \frac{K r}{C} \right) \right). \quad (4.40)$$

Exercício 4.5:

Gerar N (por ex. 1000000) números aleatórios com a distribuição dada pela Eq.(4.38), produzir um histograma desses números com n (por ex. 50) segmentos e comparar o resultado com um gráfico da $f(\theta)$. Obs: É necessário normalizar o gráfico de $f(\theta)$ de maneira a que

$$\sum_{j=1}^n f(\theta_j) = N, \quad (4.41)$$

ou seja, que o gráfico de f e o histograma tenham a mesma integral.

Caso em que $F(x)$ não é conhecida

Pode ser preciso gerar números aleatórios para uma distribuição cuja integral, Eq.(4.36), não nos fornece uma expressão analítica para a

$F(x)$. Consideremos, por exemplo, a bem conhecida distribuição "lognormal",

$$f(x) = \frac{\exp[-\log^2(x/x_0)/(2\sigma^2)]}{x\sigma\sqrt{2\pi}}, \quad x \geq 0, \quad x_0, \sigma > 0, \quad (4.42)$$

Um procedimento possível é obter a $F(x)$ por integração numérica. Discretizamos a região relevante $[a, b]$ em n pontos, $y = [y_1, y_2, \dots, y_n]$, onde $y_1 = a$, $y_{j+1} = y_j + \Delta y$, e $y_n = b$. Integraremos $f(x)$ numericamente, obtendo $F_j = F(y_j)$ para todo j , normalizamos F de maneira que $F_n = 1$, sorteamos r uniformemente distribuído entre 0 e 1 ($r = rand()$, em SCILAB), encontramos o menor j tal que $F_j > r$ e então obtemos x por interpolação entre y_{j-1} e y_j , ou seja,

$$x = y_{j-1} + \frac{(r - F_{j-1}) \Delta y}{F_j - F_{j-1}}. \quad (4.43)$$

O programa a seguir é um exemplo de programa de acordo com a receita acima, para o caso de distribuição lognormal.

Programa: logn_in.sce

```
// logn_in.sce : distribuição lognormal, gerada com
// integração numérica da densidade de probabilidade
// Exemplo de dados: [10, 2, .5, 1.e5]

par=input('give [xmax, x0, sig, N] \n');
xmax=par(1); x0=par(2); sig=par(3); N=par(4);
s2=2*sig^2; k=sig*sqrt(2*pi);
rn=zeros(1,N); // serão gerados N números aleatórios
xmin=xmax/1000;
x=linspace(xmin, xmax, 1000);
dx=x(2)-x(1);
f=exp(-(log(x/x0).^2)/s2)./(k*x);
F=zeros(1,1000);
for j=2:1000;
    F(j)=F(j-1)+f(j);
end
```

```

F=F/F(1000); // normalização da F
r=rand(1,N);
for j=1:999
    lista=find(r>F(j)&r<F(j+1));
    rn(lista)=x(j)+dx*(r(lista)-F(j))/(F(j+1)-F(j));
end
// A geração dos N números aleatórios está completa;
// o que segue é somente para observar o resultado;
nn=100; mi=min(rn); ma=max(rn);
X=linspace(mi,ma,101);
dX=X(2)-X(1);
y=linspace(X(1)+dX/2,X(101)-dX/2,100);
[cls,h]=dsearch(rn,X);
f=exp(-(log(y/x0).^2)/s2)./(k*y);
g=N*f/sum(f);
clf;
plot(y,h,y,g,'r')

```

Método Monte Carlo

Também conhecido como *Método da Rejeição* [5], o procedimento explicado a seguir tem uma aplicabilidade muito geral, sendo apropriado para gerar números aleatórios num espaço de d dimensões, $x = (x_1, x_2, \dots, x_d)$.

Sendo $f(x)$ a densidade de probabilidade, definida numa região $R = \{(a_1, b_1), (a_2, b_2), \dots, (a_d, b_d)\}$, ela deve ser normalizada de maneira que $f(x) \leq 1$ em R . Se possível, façamos $\max(f(x)) = 1$, para obter a melhor eficiência. A seguir, procedemos da seguinte maneira, na geração de cada um dos números da amostra que queremos produzir:

- 1) Geramos d números aleatórios, com distribuição uniforme entre a_j e b_j (em SILAB: $x_j = a_j + (b_j - a_j) * rand()$, $j = 1, \dots, d$)
- 2) Calculamos $f(x)$;
- 3) Geramos um número aleatório, uniforme entre 0 e 1, $r = rand()$;
- 4) Se $r \leq f(x)$ aceitamos x como um ponto gerado para nossa amostra; se $r > f(x)$, rejeitamos x e retornamos ao item 1.

Como um exemplo em duas dimensões, considere a seguinte distribuição para $x = (\theta, \phi)$, $0 \leq \theta \leq \pi$, $0 \leq \phi < 2\pi$:

$$f(\theta, \phi) = \sin(\theta) \exp(K(\cos^2(\theta) - 1) + H(\sin(\theta) \cos(\phi) - 1)). \quad (4.44)$$

Esta é a distribuição de equilíbrio da orientação de um dipolo, elétrico ou magnético, em presença de uma anisotropia uniaxial na direção z e um campo na direção x , sendo θ e ϕ seus ângulos polares. Note que f está normalizada de maneira a que $f \leq 1$ em toda região de definição. O programa a seguir segue a receita acima.

Programa: geracao_MC.sce

```
// geracao_MC.sce
// Gera N pontos em 2 dimensões, para a distribuição de
// probabilidade dos ângulos polares de um dipolo,
// em presença de campo e anisotropia (veja acima).
// Exemplo de dados:
// [K, H, N, ntheta, nphi]=[3, 1, 50000, 50, 50];
// Aqui ntheta e nphi são os números de divisões para
// os histogramas para testar os resultados.

par=input('deh [K, H, N, ntheta, nphi] \n');
K=par(1); H=par(2); N=par(3); ntheta=par(4); nphi=par(5);
theta=zeros(1,N); phi=zeros(1,N); pi=%pi;
n=0; dpi=2*pi;
while(n < N)
if(n==0); disp('AGUARDE'); n=1; end
    tt=pi*rand(); // sorteia theta
    ff=dpi*rand(); // sorteia phi
    r=rand();
    f=sin(tt)*exp(K*(cos(tt)^2-1) + ...
                  H*(sin(tt)*cos(ff)-1));
    if(r < f);
        theta(n)=tt; phi(n)=ff; n=n+1;
        if(modulo(n,10000)==0); disp([n,N]),end
    end
end
// O algoritmo termina aqui. O que segue é somente
```

```

// para ver o resultado, comparando o histograma com
// a expressão analítica. Inicialmente cria uma grade
// no espaço bi-dimensional, usando os nomes teta e
// fi para \theta e \phi, respectivamente:
teta=linspace(0, pi, ntheta);
fi=linspace(0,dpi,nphi);
g=sin(teta) .*ones(1,nphi).*exp((K*(cos(teta).^2-1) ...
    .*ones(1,nphi)+H*(sin(teta).*cos(fi)-1)));
g=N*g/sum(sum(g));
// obtém o histograma:
Hist=zeros(ntheta-1, nphi-1);
for j=1:ntheta-1
    if(j==1);disp('AGUARDE'); end
    for k=1:nphi-1
        Hist(j,k)=length(find(theta > teta(j) & ...
            theta < teta(j+1) & phi > fi(k) & phi < fi(k+1)));
    end
end
tetap=teta(1:ntheta-1); fip=fi(1:nphi-1);
xset('window',1); clf; surf(fi, teta, g);
legend('distribuição teórica')
xset('window',2); clf; surf(fip, tetap, Hist);
legend('distribuição gerada')
// obtém f(teta) integrando fi
ftheta=sum(Hist,2);
fteta=sum(g,2);
xset('window',3); clf; plot(tetap,ftheta,teta,fteta)
legend('f(teta) gerado', 'f(teta) analítico')
// obtém f(phi) integrando teta
fphi=sum(Hist,1);
ffi=sum(g,1);
xset('window',4); clf; plot(0,0,'w',fip,fphi,fi,ffi)
legend(' ','f(fi) gerado', 'f(fi) analítico')

```

Observação: No Exemplo de dados no programa acima fomos muito modestos na escolha de N, causando uma concordância ruim entre a distribuição gerada e a teórica. Com, por ex., N=1000000 teremos **muito boa concordância** mas também um tempo muito longo para

execução.

Notas sobre o método Monte Carlo:

- 1) Quando a distribuição é muito concentrada em uma pequena região do espaço, de maneira que na maior parte do espaço a probabilidade é muito pequena ou nula, este algoritmo se torna bastante ineficiente porque a maioria dos pontos sorteados serão rejeitados.
- 2) No caso de um número grande de dimensões, sortear pontos sem critério usualmente leva a uma alta taxa de rejeição; neste caso um algoritmo como *Metropolis*, explicado no capítulo 6, será preferível.

Exercício 4.6:

Gere N números aleatórios distribuídos conforme uma lognormal (veja Eq.(4.42)), usando o método de Monte Carlo. Obtenha um histograma desses números em n divisões e compare o resultado do gráfico deste histograma com o da função $f(x)$. Compare a velocidade e precisão de seu programa com as do programa "logn_in", acima, para vários valores dos parâmetros.

4.3 Processos Estocásticos (PE)

Uma classe muito importante de variáveis aleatórias é a daquelas que dependem do tempo, ou seja, aquelas cujo valor varia aleatoriamente ao longo do tempo. Uma tal variável aleatória se chama "processo estocástico", PE. Como exemplos de processos estocásticos podemos citar o número de partículas emitidas por uma fonte radioativa ou o número de acidentes que ocorrem em uma grande cidade (processos estocásticos discretos), a posição e a velocidade de uma partícula Browniana, ou a orientação (ângulos polares θ e ϕ) de uma partícula magnética coloidal num líquido (processos estocásticos contínuos). Seja $X(t)$ um PE. Em tudo o que segue, $X(t)$ pode representar uma lista de variáveis aleatórias, ou seja, um "vetor" de várias componentes ou também um escalar. Por exemplo, $X(t)$ pode ser o vetor posição de uma partícula Browniana num líquido ou pode ser o conjunto das 6 coordenadas da mesma partícula no espaço de fases, i.e., 3 coordenadas de posição e 3 de velocidade ou ainda o número de radiações emitidas por uma fonte. Vamos usar a notação $P_X(x, t)$ para indicar a probabilidade de X assumir o valor x no instante t , no

caso de PE discreto, ou a densidade de probabilidade no caso de PE contínuo. Chamamos de **realização** de um PE o conjunto de valores $x_R(t)$ assumidos por $X(t)$ num intervalo de tempo $[t_0, t_f]$, isto é, uma realização é uma função (possivelmente vetorial) de t , definida no intervalo $[t_0, t_f]$. Alternativamente, se pode dizer que a cada realização R do PE $X(t)$ corresponde uma função $x_R(t)$ cujo valor, em cada instante t , é o valor assumido por $X(t)$ na referida realização. De duas maneiras alternativas se pode calcular a média

$$\bar{G}(t) = \langle G(X(t)) \rangle$$

de uma função G de um PE $X(t)$:

a) Média sobre realizações:

$$\bar{G}(t) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{R=1}^N G(x_R(t))$$

b) Média usando a distribuição de probabilidade:

$$\bar{G}(t) = \sum_x P_X(x, t) G(x) ,$$

no caso de PE discreto, ou

$$\bar{G}(t) = \int d^n x P_X(x, t) G(x) ,$$

onde n é a dimensão de X , no caso de PE contínuo. Efetuar a média sobre realizações costuma ser o procedimento mais conveniente em simulações numéricas, enquanto que em cálculo analítico é geralmente mais conveniente usar a densidade de probabilidade para obter as médias. Uma média particularmente importante, definida em função de dois tempos consecutivos, é a “matriz de correlações”. Sejam X_1, X_2, \dots, X_n as componentes do PE $X(t)$. Definem-se os elementos da “matriz de correlações” por

$$C_{ij}(t_2, t_1) = \langle X_i(t_2) X_j(t_1) \rangle - \langle X_i(t_2) \rangle \langle X_j(t_1) \rangle .$$

Os elementos da “matriz de correlações” costumam ser chamados de “funções de correlação” e a função de correlação de uma variável com

ela mesma, em dois tempos, t_1 e t_2 , chama-se “função de autocorrelação”. Também para as funções de correlação a média pode ser obtida pelos dois métodos descritos acima, da seguinte forma:

a) Média sobre realizações:

$$C_{ij}(t_2, t_1) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{R=1}^N x_{Ri}(t_2)x_{Rj}(t_1) - \bar{x}_i(t_2)\bar{x}_j(t_1),$$

onde $\bar{x}_i(t_1) = \langle X_i(t_1) \rangle$, que se calcula conforme descrito acima (formas (a) ou (b));

b) Média usando a distribuição de probabilidade:

$$C_{ij}(t_2, t_1) = \sum_{x, x'} P_X(x, t_2; x', t_1) x_i x'_j - \bar{x}_i(t_2)\bar{x}_j(t_1),$$

no caso de PE discreto e

$$C_{ij}(t_2, t_1) = \int d^n x \int d^n x' f_X(x, t_2; x', t_1) x_i x'_j - \bar{x}_i(t_2)\bar{x}_j(t_1),$$

no caso de PE contínuo. Um PE é dito **estacionário** *latu sensu* se

$$\langle X(t) \rangle = \text{constante} \quad \text{e} \quad C_{ij}(t_2, t_1) = C_{ij}(t_2 - t_1),$$

ou seja, as funções de correlação dependem só da diferença entre os tempos t_1 e t_2 . Uma característica bastante frequente nos PE da Natureza é que as funções de autocorrelação tendem a relaxar a zero exponencialmente com o tempo, isto é,

$$C_{ii}(t_2, t_1) = C \exp \left[-\frac{t_2 - t_1}{\tau_i} \right],$$

onde τ_i se chama “tempo de correlação do PE $X_i(t)$ ”. As distribuições de probabilidade dos processos estocásticos variam com o tempo, sendo que geralmente seus valores em dois instantes t_1 e t_2 estão correlacionados. Isto significa que, geralmente, a probabilidade de X assumir o valor x_1 no instante t_1 e o valor x_2 na instante t_2 , $P_X(x_1, t_1; x_2, t_2)$ é diferente do produto $P_X(x_1, t_1)P_X(x_2, t_2)$. Isto nos sugere a definição de “probabilidade condicional” (ou “densidade de probabilidade condicional”, no caso de PE contínuos). Por conveniência de sistematização, vamos introduzir a convenção de

usar índices maiores para tempos maiores (i.e., posteriores), ou seja $t_m > t_{m-1} > \dots > t_2 > t_1 > t_0$. A “probabilidade (ou densidade) condicional” de X assumir o valor x_2 em t_2 dado que seu valor é x_1 em t_1 , $P_X(x_2, t_2 | x_1, t_1)$, é definida pela relação

$$P_X(x_2, t_2; x_1, t_1) = P_X(x_2, t_2 | x_1, t_1)P_X(x_1, t_1). \quad (4.45)$$

Também podemos definir a probabilidade (ou densidade) condicional quando é dada uma sequência de valores anteriores de X , i.e., $P_X(x, t | x_m, t_m; \dots; x_2, t_2; x_1, t_1)$, com $t > t_m > \dots > t_2 > t_1$. Quando o PE é tal que as informações em tempos anteriores ao da última informação são irrelevantes para a distribuição de probabilidade em tempos posteriores a esta, ou seja, quando

$$P_X(x, t | x_m, t_m; \dots, x_2, t_2, x_1, t_1) = P_X(x, t | x_m, t_m), \quad (4.46)$$

para quaisquer $t > t_m > \dots > t_2 > t_1$, então este PE é dito *Markoviano*. As probabilidades condicionais, $P_X(x_2, t_2 | x_1, t_1)$, neste caso, se chamam *probabilidades de transição*. No caso de PE discreto, uma sequência de valores que ele assume se chama *Cadeia de Markov*.

De longe os processos estocásticos mais trabalhados, matematicamente, são os processos Markovianos. E em Física, será que os processos Markovianos são importantes? A resposta é *sim*, são muito importantes. Por dois motivos: primeiro, porque quase todos processos estocásticos da Natureza são, pelo menos, aproximadamente Markovianos, nalguma escala de tempo, como explicaremos abaixo; segundo, por que se um processo estocástico de n variáveis, $X = (X_1, X_2, \dots, X_n)$ é não Markoviano, é sempre possível encontrar um conjunto Y de m variáveis, $m > n$, que é Markoviano e contém X , como também veremos abaixo, através de exemplos. Consideremos a posição de uma molécula selecionada em um gás. Numa escala de tempos menores do que o tempo médio entre duas colisões consecutivas da molécula a inércia da mesma é muito importante para a distribuição de probabilidade de sua posição no futuro próximo, dada a sua posição atual, pois sua velocidade se mantém até a próxima colisão. Assim é que dados dois valores consecutivos da posição, num pequeno intervalo de tempo, podemos estimar com razoável

grau de certeza sua velocidade e prever com alta probabilidade sua posição no futuro próximo. Portanto, dois dados consecutivos representam mais informação que somente o último dado na previsão da distribuição subsequente, ou seja, o processo não é Markoviano. Consideremos agora uma escala de tempos muito maiores que o tempo médio entre duas colisões consecutivas, e, consequentemente, escala de distâncias muito maiores que o livre caminho médio. Entre duas observações consecutivas terá ocorrido tantas colisões e a velocidade terá variado aleatoriamente tantas vezes que o valor da velocidade por ocasião da primeira observação é totalmente irrelevante para a distribuição de probabilidade da posição no momento da segunda observação. O processo é, portanto, Markoviano. Note que estamos falando do mesmo PE e concluímos que em uma escala de tempos ele é Markoviano e noutra (menor) ele não é Markoviano. Voltemos a considerar o mesmo PE como acima, mas agora com uma descrição diferente. Em vez da "posição" como variável estocástica, consideramos o conjunto "posição e velocidade", ou seja, $X(t) = (\mathbf{r}(t), \mathbf{v}(t))$. Neste caso a distribuição de probabilidade para $X(t + \tau)$, dado o valor de $X(t)$ pode ser calculada se conhecemos as características gerais do sistema, sendo que informações sobre os valores de X anteriores a t são irrelevantes para o cálculo, ou seja, o PE de $2n$ dimensões $X(t) = (\mathbf{r}(t), \mathbf{v}(t))$ é Markoviano mesmo numa escala de tempos pequenos.

4.3.1 Processos Estocásticos Contínuos

Alguns conceitos e algumas propriedades são específicos para PE contínuos, outros para PE discretos. Trataremos nesta seção apenas uma pequena parte dos conceitos e propriedades específicos dos PE contínuos, os quais são tratados em muito maior detalhe no capítulo sobre "Dinâmica Estocástica", Cap.VII.

Intensidade Espectral

Os sinais elétricos ou eletromagnéticos emitidos por circuitos eletrônicos e recebidos em receptores com o auxílio de filtros de frequência tem, usualmente, componentes estocásticas, chamadas *ruido*. Vamos considerar sinais estacionários, de média nula, $\langle X \rangle = 0$. A

intensidade da componente de frequência ω do sinal $X(t)$, chamada **intensidade espectral** $I(\omega)$, está relacionada com a função de autocorrelação $\langle X(t + \tau)X(t) \rangle$, pelo teorema de *Wiener-Khintchine*, que diz que

$$I(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \langle X(t + \tau)X(t) \rangle e^{-i\omega\tau} d\tau \quad (4.47)$$

ou seja, a intensidade espectral (também chamada de “densidade espectral”) é igual à transformada de Fourier da função de autocorrelação do sinal. Por analogia, dá-se o nome de intensidade espectral à transformada de Fourier da função de autocorrelação de qualquer processo estocástico.

Processo Estocástico Gaussiano

Consideremos um PE uni-dimensional $X(t)$ cuja densidade de probabilidade de ordem n , para tempos arbitrários t_1, t_2, \dots, t_n é dada por

$$f_X(x_1, t_1; x_2, t_2; \dots) = C \exp \left[-\frac{1}{2} \sum_{j=1}^n \sum_{k=1}^n a_{jk} (x_j - m_j)(x_k - m_k) \right] \quad (4.48)$$

onde $m_j = \langle X(t_j) \rangle$ e a matriz $A = \{a_{jk}\}$ é positiva definida. Os elementos da matriz inversa A^{-1} são as funções de correlação do PE $X(t)$ [18],

$$(A^{-1})_{jk} = \langle (X(t_j) - m_j)(X(t_k) - m_k) \rangle.$$

A importância dos processos Gaussianos pode ser entendida em vista do “Teorema Central de Limite”, TCL, seção 4.2.4. Consideremos um processo estocástico cuja variação entre dois instantes de tempo decorre de uma sequência de impulsos independentes, cujos efeitos somados são a variação total do processo no intervalo. Então, pelo TCL, a densidade de probabilidade para a variação do processo no intervalo é Gaussiana, o que, para o conjunto dos intervalos, corresponde à expressão acima.

4.3.2 PE Discretos: Equação Master

Consideremos um PE discreto, Markoviano, $A(t)$ que pode assumir os valores a_m , $m = 1, 2, 3, \dots$. Seja $P_m(t)$ a probabilidade de que A tenha, no instante t , o valor a_m . Seja Δt um intervalo de tempo tão pequeno que a probabilidade de haver mais de uma transição durante Δt é desprezível e seja $\Lambda_{nm'}\Delta t$ a probabilidade de, durante o intervalo Δt , A assumir o valor a_m dado que seu valor era $a_{m'}$ no início do intervalo, i.e.,

$$\Lambda_{m,m'}\Delta t = P_A(a_m, t + \Delta t | a_{m'}, t)$$

A variação, durante Δt , da probabilidade de o valor de A ser a_m ,

$$\Delta P_m(t) = P_m(t + \Delta t) - P_m(t)$$

é igual à soma de todas as probabilidades de transição dos valores $a_{m'}$ para a_m menos a soma das probabilidades de transição de a_m para os $a_{m'}$, i.e.,

$$P_m(t + \Delta t) - P_m(t) = \sum_{m'} (\Lambda_{mm'} P_{m'} - \Lambda_{m'm} P_m(t)) \Delta t$$

Dividindo esta equação por Δt e tomando o limite $\Delta t \rightarrow 0$ segue

$$\frac{dP_m(t)}{dt} = \sum_{m'} (\Lambda_{mm'} P_{m'} - \Lambda_{m'm} P_m(t)) . \quad (4.49)$$

Esta equação é conhecida como “Equação Master”, ou, em Português, equação “mestra”. Ela governa a evolução temporal da distribuição de probabilidade dos processos estocásticos Markovianos discretos.

Processo de Poisson

Consideremos um conjunto discreto de eventos que ocorrem sucessivamente no tempo a intervalos irregulares (aleatórios). Por exemplo, os eventos podem ser as emissões radioativas (partículas alfa, beta ou gama) de uma fonte ou as colisões sofridas por uma molécula em um gás. Vamos supor ainda que a probabilidade de ocorrer um evento num intervalo de tempo Δt é independente do tempo t e independente da “história” das ocorrências anteriores ao intervalo considerado. Sendo Δt suficientemente pequeno para que a probabilidade

de ocorrerem dois eventos durante Δt seja desprezível (a seguir, tomaremos o limite $\Delta t \rightarrow 0$) a probabilidade de ocorrer um evento em Δt é $\lambda \Delta t$, com $\lambda = \text{constante}$. A variável aleatória que nos interessa é o número $M(t)$ de eventos que ocorrem entre 0 e t . Vamos chamar de $P_m(t)$ a probabilidade de que exatamente m eventos ocorram até t . Temos assim precisamente as condições para a aplicabilidade da equação Master, com

$$\Lambda_{m,m'} = \lambda \delta_{m,m'+1}, \quad (4.50)$$

ou seja uma transição só pode aumentar de um a contagem de eventos. Substituindo a Eq.(4.50) na Eq.(4.49) segue

$$\frac{dP_m}{dt} = \lambda (P_{m-1} - P_m). \quad (4.51)$$

As condições iniciais para este sistema de equações são

$$P_m(t=0) = \begin{cases} 1 & \text{para } m=0 \\ 0 & \text{para } m \neq 0 \end{cases} \quad (4.52)$$

A solução deste sistema, com estas condições iniciais, é

$$P_m(t) = \frac{(\lambda t)^m}{m!} \exp(-\lambda t). \quad (4.53)$$

Esta distribuição de probabilidade é conhecida como “Distribuição de Poisson”.

Deixamos como exercício para o leitor mostrar que a média e a variância são dadas por

$$\langle M(t) \rangle = \lambda t, \quad (4.54)$$

$$\sigma_M^2 = \lambda t. \quad (4.55)$$

Simulação Numérica do Processo de Poisson

Apresentamos aqui uma maneira de fazer simulação numérica do Processo de Poisson, principalmente para que sirva como exemplo de simulação de PE discretos. Como em toda simulação numérica, discretizamos o tempo a intervalos Δt , i.e., $t = (t_1, t_2, \dots, t_f)$, com

$t_{j+1} = t_j + \Delta t$. O incremento Δt deve ser suficientemente pequeno para que a probabilidade de ocorrer mais de uma transição durante um Δt seja desprezível. No caso do Processo de Poisson, devemos ter $\lambda\Delta t \ll 1$. Como pretendemos fazer estatística sobre as realizações do processo, simulamos muitas (digamos N) realizações simultaneamente. Seja $m(t)$ o vetor dos números de eventos nas diferentes realizações, no instante t , i.e., $m(t_j) \equiv [m_1(t_j), m_2(t_j), \dots, m_N(t_j)]$, com $m(t_1) = 0$. O vetor $m(t_j)$ é obtido pelo seguinte procedimento: sorteiam-se N números aleatórios uniformemente distribuídos entre 0 e 1; se o n^{mo} número estiver num intervalo pré-determinado de comprimento $\lambda\Delta t$, por exemplo em $[0, \lambda\Delta t]$, então $m_n(t_j) = m_n(t_{j-1}) + 1$, se não, $m_n(t_j) = m_n(t_{j-1})$. A medida em que os $m(t_j)$ vão sendo simulados, calcula-se a média $\langle m(t_j) \rangle$ (média sobre as N realizações) e a correspondente variância.

Exercício 4.6: Programar a simulação do processo de Poisson, deixando $\lambda\Delta t$, N e t_f como parâmetros de entrada. Plotar a média e a variância em função de t e comparar com o resultado analítico. Executar o programa repetidas vezes, variando os parâmetros de entrada.

Capítulo 5

DINÂMICA MOLECULAR

Dinâmica Molecular é uma técnica computacional de resolver as equações de movimento dos átomos, considerados como partículas puntiformes que interagem com os demais átomos da amostra (sistema) e, possivelmente, com campos externos. À medida em que os átomos vão variando seus estados, caracterizados pelas posições e velocidades, calculam-se as variáveis dinâmicas relevantes do sistema, como temperatura ou densidade de energia, em função da posição, etc.

Na chamada “Dinâmica Molecular de Primeiros Princípios” as forças interatômicas são calculadas, a cada passo de integração das equações de movimento, por Mecânica Quântica, com base na aproximação adiabática: consideram-se os núcleos, ou íons com camadas eletrônicas fechadas, como fixos em suas posições instantâneas, e calcula-se a densidade eletrônica dos elétrons de valência, que é usada no cálculo das forças sobre os núcleos ou íons. Calculam-se então as novas posições e velocidades dos núcleos em consequência dos deslocamentos durante o intervalo de tempo Δt , sob a ação das referidas forças, por Mecânica Clássica, e assim sucessivamente. Este procedimento só pode ser usado quando a amostra contém um número relativamente pequeno de átomos (por exemplo, uma molécula ou umas poucas células unitárias de um sólido) pois em amostras com milhares de átomos o cálculo das densidades eletrônicas, por Mecânica Quântica, torna-se proibitivo.

Nos casos de amostras com muitos átomos usa-se, geralmente, um potencial de interação interatômica ou uma abordagem semi-clássica,

chamada “embedded atom method”. As forças são geralmente de curto alcance e o procedimento computacional não depende muito dos detalhes das forças. Por isso vamos introduzir o assunto considerando que a força entre os átomos i e j é do tipo gradiente de potencial central, dependente só da distância $| \mathbf{r}_i - \mathbf{r}_j |$.

5.1 Um Programa de Dinâmica Molecular

Inicialmente vamos introduzir um pequeno esquema, destacando as partes que usualmente compõem um programa de Dinâmica Molecular:

```

inicialização
for  $t = 0 : \Delta t : t_{max}$ 
    cálculo das forças
    cálculo das novas posições e velocidades
    cálculo das variáveis dinâmicas relevantes
end

```

Na *inicialização* introduzem-se todos os parâmetros do sistema, como as dimensões da amostra, constantes do potencial de interação, temperaturas nos contornos, tempo total e intervalo Δt , etc., além de se especificar o estado inicial do sistema, posições e velocidades iniciais dos átomos.

O *cálculo das forças* inclui a determinação das distâncias de cada átomo a seus vizinhos dentro do raio de ação do potencial de interação, além da determinação das componentes da força resultante sobre cada átomo. Esta é, usualmente, a parte do programa que mais consome tempo de CPU.

O *cálculo das novas posições e velocidades*, também chamado de *atualização das posições e velocidades*, é a integração das equações de movimento, usando-se um dos algoritmos introduzidos no capítulo sobre equações diferenciais ordinárias.

O cálculo das variáveis dinâmicas relevantes corresponde ao que seria o processo de medida num procedimento experimental. Usam-se as definições das variáveis de interesse em função das posições e velocidades atômicas. Muitas vezes este cálculo é feito apenas a cada n passos de integração.

Vamos detalhar cada uma das partes indicadas acima, usando para isto um exemplo simples, em duas dimensões espaciais. Por dois motivos preferimos o exemplo em duas dimensões em vez de três: primeiro porque em três dimensões, mesmo amostras com dimensões lineares correspondentes a um número modesto de distâncias interatômicas, já possuirão um número de átomos que é muito grande para um exemplo com pretensões apenas didáticas; segundo, a visualização gráfica de um modelo bi-dimensional é muito mais simples que em três dimensões. Entretanto, cabe salientar, que o procedimento computacional usado em duas dimensões pode ser trivialmente estendido para três.

5.2 Um Modelo Simples

Vamos tratar uma amostra bi-dimensional de N átomos esféricos, confinados em uma caixa retangular, de dimensões $X \times Y$. Cada átomo será identificado por um número $n = 1, 2, \dots, N$. O n^{mo} átomo terá posição e velocidade denotadas por $\mathbf{r}(n) = (x(n), y(n))$ e $\mathbf{v}(n) = (v_x(n), v_y(n))$. Todo átomo que toca uma das quatro paredes sofre reflexão elástica, isto é, a componente da velocidade perpendicular à parede troca de sinal e a componente paralela não se altera. Atribuem-se aos átomos as componentes v_x e v_y da velocidade, por sorteio de variáveis aleatórias, com distribuição Gaussiana, centrada em zero e variância T (temperatura). Isto corresponde à distribuição de Maxwell

$$P(\mathbf{v}) \propto \exp\left(\frac{-v^2}{2T}\right), \quad (5.1)$$

onde assumimos $m = 1$ e $k_B = 1$ e $v^2 = v_x^2 + v_y^2$

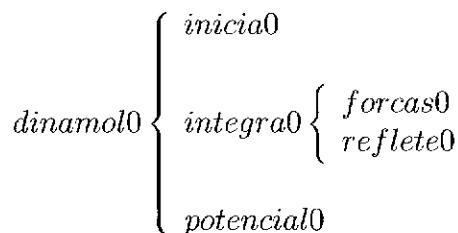
Como um programa completo de Dinâmica Molecular é bastante complexo, procuramos facilitar o entendimento do leitor introduzindo três níveis distintos, consecutivos, de programa, que chamaremos de *nível 0*, *nível 1* e *nível 2*, respectivamente. No nível 0, cujo programa principal chamaremos de *dinamol0.sce*, tratamos o sistema descrito acima sem fazer uso de células e listas, que serão introduzidas apenas no nível 1, *dinamol1.sce*, na seção 5.4.2. No nível 2, *dinamol2.sce*, nossa amostra será uma porção de um sistema maior, homogêneo, onde, em lugar de paredes refletoras, introduzimos condições de contorno periódicas.

5.3 Abordagem Nível 0

Introduziremos aqui todas as partes de um programa de Dinâmica Molecular, como descrito na seção 5.1. Sem recorrer à estratégia de células e listas (seção 5.4.2), para executarmos o cálculo das forças sobre todos os átomos teremos que calcular, a cada passo de integração, as distâncias entre todos os $N(N - 1)/2$ pares de átomos. Por isso, para ficarmos no limite de um tempo de CPU razoável, teremos que tratar amostras relativamente pequenas, digamos $\approx 10^2$ átomos. A motivação para introduzirmos esta abordagem é puramente didática, uma vez que o recurso a células e listas, poderoso na redução do tempo computacional, introduz uma complexidade bem maior nos subprogramas. O programa e seus subprogramas introduzidos neste nível servirão de base para os níveis seguintes.

5.3.1 O Programa Principal: *dinamol0.sce*

Nosso programa se constitui do programa principal, chamado “*dinamol0.sce*” e 5 subprogramas, conforme o esquema a seguir:



Além dos subprogramas, conforme o esquema acima, o programa “dinamol0.sce” contém o cálculo da temperatura e da energia total do sistema. A temperatura é recalculada ao final da integração, como o valor médio das energias cinéticas dos átomos, ou seja,

$$T_c = \frac{1}{2N} \sum_{n=1}^N (v_x(n)^2 + v_y(n)^2). \quad (5.2)$$

Para permitir ao operador continuar o cálculo por novo intervalo de tempo após examinar os resultados, o programa pede novo “tmax” e permite também alterar a temperatura.

Programa: dinamol0.sce

```
// Programa de Dinâmica Molecular, modelo de gás mono
// atômico num espaço bi-dimensional X por Y, com
// paredes perfeitamente refletoras; as posições
// iniciais dos N átomos são os sítios de uma rede
// quadrada de parâmetro de rede a; os átomos interagem
// por potencial de Lennard-Jones de parâmetro q=1 e
// mínimo em r=1 e a interação será considerada nula
// quando r>1; a integração é feita por Velocity-Verlet;
// permite continuação do cálculo e mudança da
// temperatura;
// Exemplo: [X=12 Y=12 l=3 N=144 dt=.01 tmax=1 T=1]

exec('inicia0.sce',-1);
while(length(tmax) > 0) // calcula enquanto o operador
  exec('integra0.sce',-1); //... fornecer mais tempo
  Ec=(sum(vx.^2)+sum(vy.^2))/2; // energia cinética
  exec('potencial0.sce',-1); // energia potencial V
  Energia=V+Ec; // energia total
  EcEpE=[Ec, V, Energia] // apresenta Ec, V, E na tela
  T=Ec/N // temperatura = energia cinética média
  tmax=input('novo tmax? se enter, encerra: \n');
  if(length(tmax)>0);
    Tnova=input('dê novo T; se enter, mantém; \n');
    if(length(Tnova) > 0);
```

```

renorm=sqrt(Tnova/T); // renormalização das
vx=vx*renorm;           // velocidades conforme
vy=vy*renorm;           // nova temperatura
T=Tnova;
end;
end
end

```

5.3.2 Inicialização

O subprograma “inicia0.sce”, apresentado a seguir, pede ao operador as dimensões X e Y da amostra (retângulo que contém os átomos), o número N de átomos, o incremento dt do passo de integração, o tempo total (inicial) tmax de integração, a temperatura T inicial da amostra e a a distância l entre pares de átomos a partir da qual a força de interação pode ser desprezada. Para colocar os átomos, inicialmente, numa rede quadrada de $N_y \times N_x$ sítios, calcula o parâmetro de rede, a, de modo a preencher, aproximadamente, toda a amostra. A seguir coloca os N átomos nos sítios da rede, isto é, atribui às coordenadas x(n), y(n), do átomo número n, valores tais que a cada átomo corresponde um sítio da rede.

Finalmente, são atribuídos, por sorteio, com distribuição Gaussiana centrada em zero e largura \sqrt{T} (veja Eq.(5.1)) valores para as coordenadas $v_x(n)$ e $v_y(n)$ das velocidades.

Subprograma: `inicia0.sce`

```

// Inicia programa de Dinâmica Molecular;
par=input('dê [X, Y, l, N, dt, tmax, T] \n');
X=par(1); Y=par(2); l=par(3); N=par(4);
dt=par(5); tmax=par(6); T=par(7);
l2=l^2;
S=X*Y;           // área da amostra;
a=sqrt(S/N); // parâmetro de rede para encher o espaço
               //... com N pontos;
Nx=ceil(X/a); // núm. de sítios iniciais ao longo de x;
Ny=ceil(Y/a); // mesmo para o eixo y;
a=min(X/Nx, Y/Ny); // se necessário, diminui ‘‘a’’ para

```

```

// ... caberem Nx átomos em X e Ny em Y;
x=zeros(1,N); y=zeros(1,N);
for n=1:N;
z=(n-1)/Nx;
x(n)=a*(Nx*(z-floor(z))+.5); //coordenada x inicial;
y(n)=a*(ceil(n/Nx)-.5); // coordenada y inicial;
end

//velocidades iniciais sorteadas à temperatura T:
vx=sqrt(T)*rand(1,N,'normal');
vy=sqrt(T)*rand(1,N,'normal');

```

5.3.3 Cálculo das Forças

A força sobre cada átomo é a soma das forças de interação do mesmo com os demais átomos da amostra. Um modelo de potencial de interação muito usado em Dinâmica Molecular, pela sua simplicidade, é o “Potencial de Lennard-Jones”, que descrevemos a seguir. A essência do procedimento de cálculo não depende muito dos detalhes do potencial de interação utilizado. Embora nós usaremos Lennard-Jones em nossos exemplos, os mesmos procedimentos podem ser utilizados com forças de interação bem mais complexas e realísticas, como as que são calculadas com base na Mecânica Quântica[11, 12, 13].

O potencial de Lennard-Jones

É claro que para uma amostra manter-se em estado sólido ou líquido, é necessário que a força de interação entre os átomos tenha uma componente atrativa. Por outro lado, para que os átomos não possam superpor-se uns aos outros é necessário que haja também um caroço repulsivo. O potencial de Lennard-Jones possui essas características:

$$V(r) = q \left\{ \left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^6 \right\} \quad (5.3)$$

onde r é a distância entre os átomos e q e σ são constantes. Esse potencial representa bastante bem a interação entre átomos de argônio, mas mesmo no caso de outros átomos não ionizados ele é uma

aproximação qualitativamente razoável. O mínimo desse potencial ocorre em $r_{min} = 2^{1/6}\sigma$. Por simplicidade vamos escolher a unidade de distância de maneira que o mínimo do potencial ocorra em $r = 1$. Com isso o potencial de Lennard-Jones será escrito na forma

$$V(r) = q \left\{ \frac{1}{r^{12}} - \frac{2}{r^6} \right\}. \quad (5.4)$$

Na Figura 5.1 vemos um gráfico do potencial de Lennard-Jones, onde escolhemos $q = 1$.

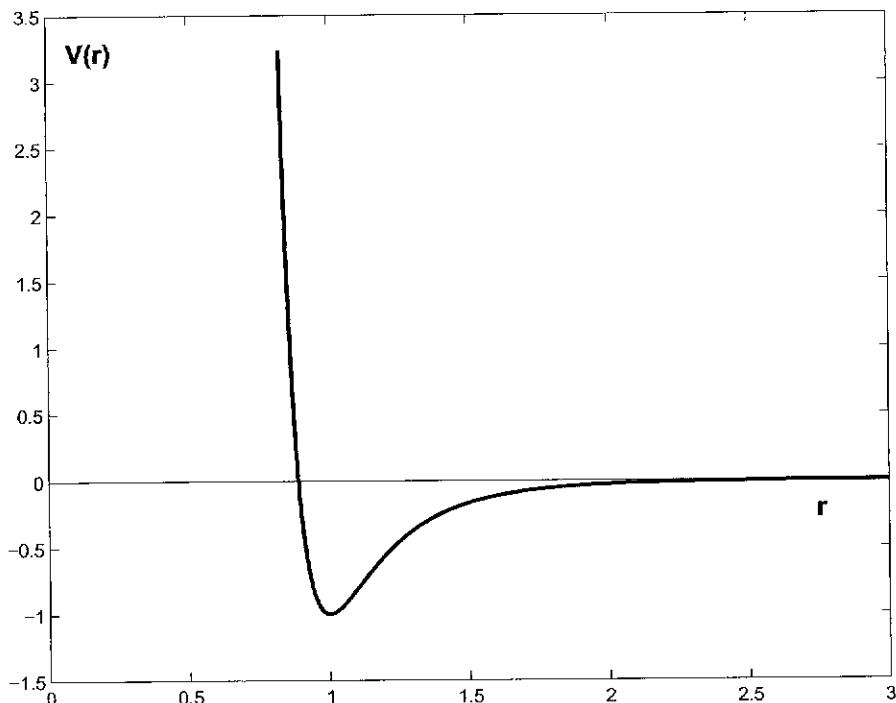


Figura 5.1: Potencial de Lennard-Jonnes

O cálculo da força, correspondente a esse potencial, é bem simples:

$$\mathbf{F}(r) = -\frac{dV}{dr} = 12 q \left\{ \frac{1}{r^{14}} - \frac{2}{r^8} \right\} \mathbf{r}. \quad (5.5)$$

Como podemos ver na figura, o potencial vai a zero muito rapidamente para distâncias maiores que $r = 2.5$. A força correspondente é atrativa a partir de $r = 1$, tendo seu máximo atrativo em

$r = 1.11$, com $F(1.11) = -2.69$, enquanto que $F(2.5) = -0.019$ e $F(3.0) = -0.0055$. Nós consideraremos $F = 0$ para $r \geq l$, com l a ser fornecido pelo operador. Uma sugestão é $l = 3.0$.

No subprograma “forcas0.sce”, a seguir, calculamos as distâncias entre todos os pares de átomos, mas as componentes da força só são calculadas entre aqueles pares cuja separação é menor que l .

Subprograma: forcas0.sce

```
// Calcula as forças de Lennard-Jones entre os átomos:
fx=zeros(1,N); // inicia as forças em zero;
fy=zeros(1,N);
fpx=zeros(N,N); fpy=fpx; // inicia matriz de forças;
for k=1:N-1;
    p=k+1:N // vetor com a numeração dos demais átomos;
    delx=x(k)-x(p); //delta x dos át. seguintes ao át. k;
    dely=y(k)-y(p); //delta y dos át. seguintes ao át. k;
    R2=delx.^2+dely.^2;//quadr. dist. do k aos seguintes;
    lista=find(R2<12);
    if(length(lista)>0);
        lista=k+lista;
        delx=x(k)-x(lista);
        dely=y(k)-y(lista);
        r2=delx.^2+dely.^2;
        flj=12*(ones(r2)./(r2.^7) - ones(r2)./(r2.^4));
            // ... fator força de Lennard-Jones;
        fpx(lista,k)=(flj.*delx)';
        fpy(lista,k)=(flj.*dely)';
        fpx(k,lista)=-fpx(lista,k)';
        fpy(k,lista)=-fpy(lista,k)';
    end
end
fx=sum(fpx,1); // força total sobre cada átomo
fy=sum(fpy,1); // ... devida aos demais átomos;
```

5.3.4 Integração das Equações de Movimento

Para realizar a integração das equações de movimento escolhemos o algoritmo, visto no Capítulo II, Velocity-Verlet. Este é um algoritmo que tem boa precisão e que tem a vantagem, em relação a Verlet, de calcular as velocidades como parte integrante do algoritmo, e nós precisaremos das velocidades para calcular a energia e a temperatura. Para realizar esta tarefa, o subprograma “integra0.sce”, apresentado a seguir, chama também o subprograma “forcas0.sce”, que vimos acima, e o subprograma “reflete0.sce”, que veremos a seguir. Ao final de cada passo de integração são mostradas na tela as novas posições atômicas.

Cabe aqui uma consideração a respeito do passo de integração, dt , cujo valor é pedido em “inicia0.sce”. Com as unidades escolhidas, a distância entre o mínimo do potencial de Lennard-Jones, $r = 1$, e o ponto onde ele se anula, $r \approx 0.9$, é $\Delta r \approx 0.1$. Não é desejável que num único passo de integração dois átomos se aproximem mais do que este valor. Por isso, chamando de v a velocidade relativa de aproximação entre eles, queremos $v dt \leq 0.1$. Uma estimativa do máximo valor de v pode ser feita como segue. Em nossas unidades a energia cinética média é igual à temperatura, T , e as distribuições de velocidades são Gaussianas centradas em zero e largura $\sigma = \sqrt{T}$. Nessas condições é muito pouco provável ocorrerem velocidades maiores que 4σ , ou velocidades relativas maiores que 8σ . Como orientação para a escolha de dt usamos, então, $8\sqrt{T}dt \leq 0.1$, ou, arredondando, $dt \leq 0.01/\sqrt{T}$.

Subprograma: integra0.sce

```
// Integra a equação de movimento de Dinâmica
// Molecular por Velocity-Verlet:
dt2=dt*dt; mdt2=.5*dt2; h=dt/2;
itmax=ceil(tmax/dt);
exec('forcas0.sce',-1); // forças entre os átomos;
for it=1:itmax;
    fxi=fx; fyi=fy;
    x=x+vx*dt+mdt2*fxi; // pos. no fim do intervalo;
    y=y+vy*dt+mdt2*fyi;
    exec('forcas0.sce',-1); // recalcula fx e fy;
    vx=vx+h*(fx+fxi); //veloc. no fim do intervalo;
```

```

vy=vy+h*(fy+fyi);
exec('reflete0.sce',-1);
plot(x,y,'o')
xset('auto clear','on')
end

```

Reflexão nas Paredes

Assim que são obtidas as novas posições e velocidades, ao final do passo de integração, é chamado o subprograma “reflete0.sce”. Este verifica se há algum átomo que ultrapassou os limites da amostra (as paredes). Neste caso a componente da velocidade perpendicular à parede ultrapassada é revertida e a componente paralela é mantida. Para que a apresentação das posições atômicas em um gráfico, que se altera a cada passo de integração, não fique mudando frequentemente de escala, produzindo uma imagem dinamicamente desagradável, nós colocamos as paredes em $x=0.1$, $x=X-0.1$, $y=0.1$ e $y=Y-0.1$;

Subprograma: reflete0.sce

```

// Átomos fora da amostra têm suas velocidades
// ... refletidas:
list1=find(x<=0.1); // identidade dos átomos que
// ... atingiram a parede da esquerda, que
    // ... está em 0.1 para melhorar os gráficos
if(length(list1) > 0);
vx(list1)=abs(vx(list1)); //reflete suas velocidades;
end
list2=find(x>=X-0.1); // identidade dos átomos que
    // ... atingiram a parede da direita;
if(length(list2) > 0);
vx(list2)=-abs(vx(list2)); //reflete suas velocidades;
end
list3=find(y<=0.1); // identidade dos átomos que
    // ... atingiram a parede de cima;
if(length(list3) > 0);
vy(list3)=abs(vy(list3)); //reflete suas velocidades;
end

```

```

list4=find(y>=Y-0.1); // identidade dos átomos que
// ... atingiram a parede de baixo;
if(length(list4) > 0);
vy(list4)=-abs(vy(list4)); // reflete suas velocidades;
end

```

5.3.5 Cálculo das Variáveis Dinâmicas Relevantes

Qualquer variável que dependa só das posições e velocidades dos átomos pode ser calculada ao final de cada passo de integração. Neste exemplo simples, desenvolvido acima, não temos muitas opções, fazendo sentido apenas calcular a energia interna, a temperatura e a pressão sobre as paredes. A temperatura é simplesmente a energia cinética média, pois elegemos unidades tais que a constante de Boltzmann é $k_B = 1$ e a massa atômica é $m = 1$. Como estamos em duas dimensões, a energia cinética média é

$$\langle E_c \rangle = \frac{1}{2} \langle v_x^2 + v_y^2 \rangle = T , \quad (5.6)$$

pois

$$\langle v_x^2 \rangle = \langle v_y^2 \rangle = T . \quad (5.7)$$

A energia interna, E , é uma variável muito conveniente de ser calculada. Como nosso modelo é um sistema isolado, sem dissipação de energia, a energia interna deve manter-se constante. Isto pode servir como um controle sobre a precisão do cálculo numérico. Como a energia interna é a soma da energia cinética, E_c , que já obtivemos ao “medir” a temperatura, com a energia potencial de interação entre os átomos, V , basta calcular esta última nos mesmos instantes em que “medimos” a temperatura e adicionar os resultados,

$$E = E_c + V , \quad (5.8)$$

onde V é dado pela Eq.(5.4).

O subprograma “potencial0.sce”, que é chamado pelo programa principal segue um esquema parecido com o subprograma “forcas0.sce”. Uma possível versão do mesmo é como segue:

Subprograma: potencial0.sce

```
// Calcula a energia potencial de Lennard-Jones
// somando sobre todos os pares de átomos vizinhos:
V=0;
for n=1:N-1;
    p=n+1:N; // lista dos átomos posteriores ao n;
    delx=x(n)-x(p)'; // delta x do átomo n aos átomos p;
    dely=y(n)-y(p)'; // delta y do átomo n aos átomos p;
    R2=delx.^2+dely.^2; // quadr. da dist. do n aos p;
    lista=find(R2<12);
    if(length(lista)>0);
        lista=n+lista;
        delx=x(n)-x(lista)';
        dely=y(n)-y(lista)';
        r2=delx.^2+dely.^2; // quadr. da dist. do n
                           // ... aos p próximos;
        Vp=ones(r2)./(r2.^6) - 2*ones(r2)./(r2.^3);
        V=V+sum(Vp);
    end
end
```

Exercício 5.1:

Entenda o programa “dinamol0.scc” e todos os seus subprogramas. Execute-o com diversos valores dos parâmetros. Introduza uma “medida” do tempo gasto no processo de integração. Procure explicar porque a temperatura varia durante o processo. Verifique a precisão na conservação da energia total, quando não for fornecida nova temperatura pelo operador.

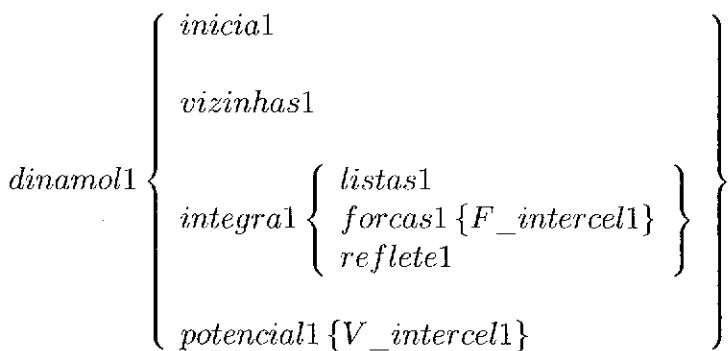
5.4 Abordagem Nível 1

Trataremos nesta seção o mesmo modelo apresentado na seção 5.2 e tratado na seção 5.3. A diferença fundamental é que agora faremos uso da estratégia de células e listas, o que nos permite aumentar muito o tamanho (número de átomos) da amostra, mantendo o tempo de

processamento num limite razoável. Não apresentaremos novamente todos os detalhes do processo, chamando atenção apenas aos aspectos nos quais o presente tratamento difere do anterior.

5.4.1 O Programa Principal: dinamol1.sce

Nosso programa se constitui do programa principal, chamado “dinamol1.sce” e 9 subprogramas, conforme o esquema a seguir:



Como no caso de “dinamol0.sce”, o programa “dinamol1.sce” também contém o cálculo da temperatura e da energia total do sistema e também propicia ao operador a possibilidade de continuar a integração por mais tempo e alterar a temperatura, se assim desejar.

Programa: dinamol1.sce

```

// Programa de Dinâmica Molecular, modelo de gás mono
// atômico num espaço bi-dimensional X por Y, com
// paredes perfeitamente refletoras; as posições
// iniciais dos N átomos são os sítios de uma rede
// quadrada de parâmetro de rede a; os átomos
// interagem por potencial de Lennard-Jones de
// parâmetro q=1 e mínimo em r=1; a integração é feita
// por Velocity-Verlet;
// usam-se células para o cálculo das forças e energia
// potencial; permite continuação do cálculo e mudança
// da temperatura;
// Ex.: [X=18, Y=18, l=3, N=324, dt=.01, tmax=1, T=.1]

```

```

exec inicial.sce;
exec vizinhas1.sce; // identifica as 4 células vizinhas
                     // ... a cada célula;
while(length(tmax) > 0) // calcula enquanto o operador
                     // ... fornecer mais tempo;
exec integral1.sce; // integra por Velocity-Verlet;
Ec=(sum(vx.^2)+sum(vy.^2))/2; // energia cinética;
exec potencial1.sce; // energia potencial total V;
Energia=V+Ec; // energia total;
EcEpE=[Ec, V, Energia] //apresenta na tela Ec, V, E;
T=Ec/N // temperatura = energia cinética média;
tmax=input('tempo para cont.? enter, encerra: \n');
if(length(tmax) == 0); break; end; // se não foi
                     // ... dado tempo, encerra;
Tn=input('dê nova temperatura; enter, mantém; \n');
if(length(Tn) > 0);
    renorm=sqrt(Tn/T); // fator de renormalização das
    vx=vx*renorm;      // velocidades conforme
    vy=vy*renorm;      // a nova temperatura;
    T=Tn;
end;
end

```

5.4.2 Células e Listas

Como em Dinâmica Molecular tratam-se sistemas com um grande número de átomos, sendo hoje comum tratarem-se amostras com $N \approx 10^6$ átomos, é claramente impossível calcular as forças de interação de cada um deles com todos ou outros. Teríamos que calcular $N^2/2$ distâncias interatômicas a cada passo de integração, uma tarefa impossível para N grande, mesmo para os mais velozes computadores. Como as forças de interação são geralmente de curto alcance, basta calcular as mesmas entre cada átomo e seus vizinhos próximos. Para isto precisamos de um procedimento pelo qual se sabe quais são os vizinhos de cada átomo, sem calcular as distâncias a todos os demais. Um procedimento possível é o que passamos a expor.

Dividimos o espaço da amostra em células quadradas de lado l , igual à máxima distância para as quais consideramos que as forças de interação devem ser computadas. Cada célula será identificada por um número, de 1 ao número de células, $Nc = Lx \times Ly$, sendo Lx e Ly os números de colunas e filas de células, respectivamente.

A cada célula será associada uma lista de números que correspondem aos átomos cujos centros se encontram na mesma. Assim, as forças de interação sobre cada átomo serão calculadas apenas com os átomos da mesma célula e com os das células vizinhas (8 células vizinhas no exemplo bi-dimensional).

5.4.3 Inicialização

A inicialização é semelhante à de dinamol0, exceto que aqui, no subprograma “inicia1.sce”, introduzem-se também as dimensões e a numeração das células. Se o lado l das células não for submúltiplo das dimensões X e Y da amostra, aumenta-se l o mínimo necessário para se ter $Lx \times l = X$, com L_x inteiro. É possível, dependendo do valor de Y , que $Ly \times l$ venha a ser maior que Y , ou seja, uma parte das células, da última fila fique fora da amostra. Isto não impede o funcionamento correto do programa, mas é conveniente sempre fornecer as dimensões X e Y de maneira que ambas sejam múltiplas de l . Teremos Lx colunas e Ly filas de células, que preenchem exatamente a amostra. A enumeração das células é feita na matriz $Ly \times Lx$, de elementos $ncel(i, j) = c$, ou seja, associa um número $c = 1, 2, \dots, Nc$ à célula que está na fila i e coluna j . A partir daí o subprograma “inicia1” é igual ao “inicia0”.

Subprograma: inicia1.sce

```
// Inicia programa de dinâmica molecular;
par=input('deh [X, Y, l, N, dt, tmax, T ] \n');
X=par(1); Y=par(2); l=par(3); N=par(4);
dt=par(5); tmax=par(6); T=par(7);
Lx=floor(X/l); // número de células ao longo do eixo x
Ly=ceil(Y/l); // número de células ao longo do eixo y
l=X/Lx; // acerta tamanho de l para ficar
```

```

// ... submúltiplo de X
ncel=zeros(Ly,Lx);
// enumeração sequencial das célula:
Nc=Lx*Ly; // número total de células
Nf=Nc+1; // célula fictícia vazia de átomos
for i=1:Ly;
for j=1:Lx;
ncel(i,j)=(i-1)*Lx+j; // número da célula (i,j)
end;
end;
// distribuição dos átomos na amostra:
S=X*Y; // área da amostra
a=sqrt(S/N); // parâmetro de rede para encher o espaço
// com N pontos
Nx=ceil(X/a); // núm. de sítios iniciais ao longo de x
Ny=ceil(Y/a);
a=min(X/Nx, Y/Ny); // se necessário, diminui a para
// caberem Nx átomos em X e Ny em Y
x=zeros(1,N); y=zeros(1,N);
for n=1:N;
z=(n-1)/Nx;
x(n)=a*(Nx*(z-floor(z))+.5); // define coordenada x
// ... inicial
y(n)=a*(ceil(n/Nx)-.5); // define coordenada y
// ... inicial
end
// velocidades iniciais sorteadas à temperatura T
vx=sqrt(T)*rand(1,N,'normal');
vy=sqrt(T)*rand(1,N,'normal');

```

5.4.4 Identificação das Células Vizinhas

No cálculo das forças entre átomos pertencentes a células vizinhas serão consideradas apenas 4 células vizinhas a cada célula, para evitar calcular duas vezes as mesmas forças. Para acompanhar a explicação a seguir, veja a figura 6.1. Por exemplo, após calcular a força f do átomo n pertencente à célula 11 sobre o átomo m pertencente

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20			24
25							32
33			C	V1			40
41		V2	V3	V4			48
49							56
57	58	59	60	61	62	63	64

Figura 5.2: Esquema de células enumeradas, com indicação das vizinhas para cálculo das forças

à célula 10, somamos f à força total sobre o átomo m e somamos $-f$ à força total sobre o átomo n . Por isso, ao escolhermos as células vizinhas à 11, para calcular as forças sobre seus átomos, não devemos incluir a célula 10, pois as interações com os átomos dessa foram incluídas ao calcularmos as forças sobre os da 10. Assim, como vizinhas à célula 11 incluímos as células 12, 18, 19 e 20, mas não as células 2, 3, 4 e 10. Generalizando, conforme figura 5.1, selecionamos como células vizinhas à c as células $v1$, $v2$, $v3$ e $v4$. O subprograma “vizinhos1.sce” cria os vetores $v1$, $v2$, $v3$ e $v4$, com elementos $v1(c)$, etc, como indicado na figura. Por exemplo, $v1(11)=12$, $v2(11)=18$, $v3(11)=19$ e $v4(11)=20$. A célula “virtual”, de número $Nf-Nc+1$ (65 no nosso exemplo), com zero átomos, assume a função das vizinhas inexistentes. Por exemplo, $v1(16)=65$, $v4(16)=65$, $v3(60)=65$, etc.

Como a célula Nc (64 no nosso exemplo) não possui vizinhas, conforme convenção acima (seriam todas 65), não incluímos a Nc no “for” que lista as vizinhas a cada célula.

Subprograma: vizinhast.sce

```
// Enumera as células vizinhas à direita e abaixo de
// cada célula enumerada sequencialmente;
v1=zeros(1,Nc-1); v2=zeros(1,Nc-1);
v3=zeros(1,Nc-1); v4=zeros(1,Nc-1);

// vizinhas das células do corpo principal:
for i=1:Ly-1;
    for j=2:Lx-1;
        c=ncel(i,j); // número da célula (i,j);
        v1(c)=ncel(i,j+1); // vizinha à direita;
        v2(c)=ncel(i+1,j-1); //vizinha abaixo à esquerda;
        v3(c)=ncel(i+1,j); // vizinha abaixo;
        v4(c)=ncel(i+1,j+1); // vizinha abaixo à direita;
    end
end

// vizinhas das células da primeira coluna:
for i=1:Ly-1;
    c=ncel(i,1);
    v1(c)=ncel(i,2);
    v2(c)=Nf; // como não há vizinha à esquerda,
//...define-se como a célula fictícia Nf, com 0 átomos;
    v3(c)=ncel(i+1,1);
    v4(c)=ncel(i+1,2);
end

// vizinhas das células da última coluna:
for i=1:Ly-1;
    c=ncel(i,Lx);
    v1(c)=Nf;
    v2(c)=ncel(i+1,Lx-1);
    v3(c)=ncel(i+1,Lx);
```

```

v4(c)=Nf;
end

// vizinhas das células da última fila:
for j=1:Lx-1;
    c=ncel(Ly,j);
    v1(c)=ncel(Ly,j+1);
    v2(c)=Nf;
    v3(c)=Nf;
    v4(c)=Nf;
end

```

5.4.5 Listas dos Átomos Presentes em Cada Célula

Definidas as coordenadas das posições atômicas, temos que construir as listas que indicam quais átomos se encontram em cada célula. Isto é feito no subprograma “listas1.sce”, apresentado a seguir.

Subprograma: listas1.sce

```

// Construção das listas:
num=zeros(1,Nf); //inicia com zero át. em cada célula;
lista=zeros(Nf,2*ceil(N/Nc)); // espaço reservado
    // ... inicialmente para as listas;
for n=1:N;
    i=ceil(y(n)/l); // fila da célula do átomo n;
    if(i<1);i=1;end
    if(i>Ly);i=Ly; end
    j=ceil(x(n)/l); // coluna da célula do átomo n;
    if(j<1); j=1; end
    if(j>Lx); j=Lx; end
    c=ncel(i,j); // número sequencial da célula (i,j);
    num(c)=num(c)+1; // número de át. na célula c;
    lista(c,num(c))=n; // o átomo n está na posição
                        // ... num(c) na lista da célula c;
end

```

5.4.6 Cálculo das Forças

O subprograma “forcas1.sce”, que apresentamos abaixo, difere substancialmente do “forcas0.sce” devido ao recurso às células e suas listas. Na primeira parte do subprograma, forças intracélulas, percorrendo todas as células, calculam-se as forças entre pares de átomos pertencentes à mesma célula. Na segunda parte, forças intercélulas, novamente percorrendo todas as células, exceto a última que não tem vizinhas à direita e abaixo, constrói-se em cada célula o vetor “listc”, que lista os átomos da célula c e o vetor “listv”, que lista os átomos que estão nas quatro vizinhas, $v1(c), v2(c), v3(c)$ e $v4(c)$. Feito isto, chama-se o subprograma “F_intercel1.sce” que calcula as forças entre os pares de átomos, sendo um da “listc” e outro da “listv”.

Programa: forcas1.sce

```
// Calcula as forças de Lennard-Jones entre os átomos:
fx=zeros(1,N); // inicia as forças em zero;
fy=zeros(1,N);

// Forças intracélula:
for c=1:Nc;
    nc=num(c); // número de átomos na célula (c);
    if(nc >= 2); // se nc<2 passa para célula seguinte;
        fpx=zeros(nc,nc); // inicia matriz de forças
        fpy=zeros(nc,nc);
        listc=lista(c,1:nc); //lista dos átomos na cél. c;
        for k=1:nc-1; // k=posição do átomo na lista c;
            p=k+1:nc; //posição dos demais átomos na lista c;
            nk=listc(k); //identidade do átomo k;
            np=listc(p); //identidade dos áts. p na lista c;
            delx=x(nk)-x(np)'; // delta x dos áts. p ao k;
            dely=y(nk)-y(np)'; // delta y dos áts. p ao k;
            r2=delx.^2+dely.^2; // quadrado da distância;
            flj=12*(ones(r2)./(r2.^7) - ones(r2)./(r2.^4));
            fpx(p,k)=(flj.*delx); //força dos p sobre o k;
            fpy(p,k)=(flj.*dely);
            fpx(k,p)=-fpx(p,k)'; // força do k sobre os p;
```

```

    fpy(k,p)=-fpy(p,k)';
end
fx(listc)=sum(fpx,1); // força total sobre cada
// ... átomo da célula devida aos demais;
fy(listc)=sum(fpy,1);
end
end

// Forças intercelulas:
exec F_intercell.sce;

```

No programa “forcas1.sce”, acima, para se calcularem as forças entre átomos situados em células diferentes, chama-se o subprograma “F_intercell.sce”, que apresentamos a seguir.

Programa: F_intercell.sce

```

// Forças intercelulas: consideram-se, para cada
// célula, os átomos das células vizinhas situadas
// à direita e abaixo da mesma.

// Inicialmente constroem-se as listas dos átomos
// que estão em cada célula, listc, e as listas dos
// átomos que estão nas 4 células vizinhas de c, listv:
for c=1:Nc-1;
    nc=num(c); // número de átomos na célula c;
    if(nc > 0);
        listc=lista(c,1:nc); // lista dos átomos na c;
        n1=num(v1(c)); // número de átomos na vizinha v1;
        n2=num(v2(c));
        n3=num(v3(c));
        n4=num(v4(c));
        nv=n1+n2+n3+n4; // número de átomos nas 4 vizinhas;
        if(nv > 0);
            // lista as identidades dos átomos das 4 vizinhas:
            listv=[];
            if(n1>0);listv=lista(v1(c),1:n1);end;
            if(n2>0);listv=[listv,lista(v2(c),1:n2)];end;

```

```

        if(n3>0);listv=[listv,lista(v3(c),1:n3)];end;
        if(n4>0);listv=[listv,lista(v4(c),1:n4)];end;
    end
end

// Dadas duas listas de átomos, listc e listv, cujas
// posições são conhecidas, calcula as forças de
// interação, derivadas do potencial de Lennard-
// Jonnes e soma com as forças, já calculadas, com
// os demais átomos da amostra:
xc=ones(nv,1)*x(listc); // matriz nv por nc onde
    // cada fila é a lista das coordenadas x dos
    // átomos da listc;
yc=ones(nv,1)*y(listc); // mesmo para coordenada y;
xv=x(listv)’*ones(1,nc); // matriz onde cada coluna
    // lista as coordenadas x dos átomos da listv;
yv=y(listv)’*ones(1,nc); // mesmo para coordenadas y;
delx=xc-xv; // matriz das distâncias x dos de listv
    // aos de listc;
dely=yc-yv; // mesmo para distâncias y;
r2=delx.^2+dely.^2; // quadr. das dist. dos c aos v;
flj=12*(ones(r2)./(r2.^7) - ones(r2)./(r2.^4));
fpx=flj.*delx; // forças dos listv sobre os listc;
fpy=flj.*dely; // mesmo, componente y;
fx(listc)=fx(listc)+sum(fpx,1); //força sobre os listc;
fy(listc)=fy(listc)+sum(fpy,1);
fx(listv)=fx(listv)-sum(fpx’,1); //força sobre os listv;
fy(listv)=fy(listv)-sum(fpy’,1);
end

```

5.4.7 Integração das Equações de Movimento

A atualização das posições e velocidades, no passo de integração, é feito pelo subprograma “integral.sce”, já apresentado como “integra0.sce” na “abordagem nível 0”, seção 5.3.4. As únicas diferenças estão nos nomes dos subprogramas que ele chama, “forças1” em vez de “forcas0” e “refletem1”, em vez de “reflete0”, e no acréscimo da chamada,

antes de “forcas1”, do subprograma “listas1”, em ambas as linhas em que aparece “forças1”.

Subprograma: integral.sce

```
// Integra a equação de movimento de Dinâmica
// Molecular com células por Velocity-Verlet
dt2=dt*dt; mdt2=.5*dt2; h=dt/2;
itmax=ceil(tmax/dt);
exec listas1.sce; // lista os átomos de cada célula
exec forcas1.sce; // calcula as forças entre os átomos
for it=1:itmax;
    fxi=fx; fyi=fy;
    x=x+vx*dt+mdt2*fxi; // posições no fim do intervalo
    y=y+vy*dt+mdt2*fyi;
    exec reflete1.sce;
    exec listas1.sce;
    exec forcas1.sce; // recalcula fx e fy
    vx=vx+h*(fx+fxi); //veloc. no fim do intervalo;
    vy=vy+h*(fy+fyi);
    exec reflete1.sce;
    plot(x,y,'o')
    xset('auto clear','on')
end
```

Subprograma: reflete1.sce

Este subprogramma é idêntico ao “reflete0.sce”. Por isso não vamos repeti-lo.

5.4.8 Cálculo das Variáveis Dinâmicas Relevantes

No cálculo da energia cinética e temperatura o procedimento é idêntico ao que seguimos na abordagem Nível 0. A energia potencial, entretanto, também faz uso das células, como no cálculo das forças. O subprogramma “potencial1.sce”, que é chamado pelo programa principal, segue um esquema parecido com o subprogramma “forcas1.sce”.

Subprograma: potencial1.sce

```

// Calcula a energia potencial de Lennard-Jones
// somando sobre todos os pares de átomos vizinhos:

// Potenciais intracélulas:
V=0;
for c=1:Nc;
  nc=num(c);
  if(nc >= 2);
    for k=1:nc-1;
      p=k+1:nc;
      nk=lista(c,k);
      np=lista(c,p);
      delx=x(nk)-x(np)';
      dely=y(nk)-y(np)';
      r2=delx.^2+dely.^2;
      Vp=ones(r2)./(r2.^6) - 2*ones(r2)./(r2.^3);
      V=V+sum(Vp);
    end
  end
end

// Potenciais intercelulares:
exec V_intercel1.sce;

```

No programa acima, para o cálculo do potencial "intercelulares", é chamado o subprograma "V_intercell.sce", que apresentamos a seguir.

Subprograma: V_intercel1.sce

```

// Potenciais intercelulares: inicialmente constrói
// as listas dos átomos que estão em cada célula c,
// listc, e as listas dos átomos que estão nas células
// vizinhas de c, listv.
for c=1:Nc-1;
  nc=num(c);
  if(nc>0);
    listc=lista(c,1:nc);

```

```

n1=num(v1(c));
n2=num(v2(c));
n3=num(v3(c));
n4=num(v4(c));
nv=n1+n2+n3+n4;
if(nv>0);
    listv=[];
    if(n1>0);listv=lista(v1(c),1:n1);end
    if(n2>0);listv=[listv,lista(v2(c),1:n2)];end;
    if(n3>0);listv=[listv,lista(v3(c),1:n3)];end;
    if(n4>0);listv=[listv,lista(v4(c),1:n4)];end;
end
end

// Dadas duas listas de átomos, listc e listv, cujas
// posições são conhecidas, calcula as energias de
// interação e soma com a energia de interação, já
// calculada, com os demais átomos da amostra

xc=ones(nv,1)*x(listc);
yc=ones(nv,1)*y(listc);
xv=x(listv)'*ones(1,nc);
yv=y(listv)'*ones(1,nc);
delx=xc-xv;
dely=yc-yv;
r2=delx.^2+dely.^2;
Vp=ones(r2)./(r2.^6) - 2*ones(r2)./(r2.^3);
V=V+sum(Vp);
end

```

5.5 Abordagem Nível 2: Condições de Contorno Periódicas

Por mais rápidos que sejam os modernos computadores, um programa de Dinâmica Molecular sempre tratará amostras com um número de átomos que é muito pequeno em comparação com o número de

átomos de uma amostra macroscópica, por menor que esta seja. No exemplo utilizado acima consideramos que os contornos da amostra são constituídos por paredes perfeitamente refletoras, o que pode ter influência decisiva sobre suas propriedades. Um truque muito usado para diminuir os efeitos advindos de se tratar uma amostra pequena é o uso de “condições periódicas de contorno”, pelo qual a amostra tratada simula um pequeno pedaço no interior de uma porção maior do mesmo material. O procedimento consiste em considerar como continuação de cada extremidade da amostra a extremidade oposta. Assim, se um átomo, em um passo de integração, sai uma distância Δx além do limite à direita da amostra, ele é recolocado na mesma a uma distância Δx à direita do limite à esquerda, mantendo as demais coordenadas. O mesmo vale para as interações. Os átomos que se encontram na coluna de células mais à direita interagem com os que se encontram na coluna do extremo esquerdo, como se esta estivesse imediatamente à direita daquela ou, equivalentemente, como se a última coluna, à direita, estivesse imediatamente à esquerda da primeira. Semelhantemente, os átomos da fila superior de células interagem com os da fila no extremo inferior, e igualmente se houver uma terceira dimensão.

Este procedimento substitui as “reflexões” usadas no programa de integração acima. Em vez de “refletir” as velocidades dos átomos que ultrapassam os limites da amostra, o programa troca o valor de suas coordenadas, ou seja, “repositiona” os átomos. Por exemplo, se na direção x a amostra vai de 0 a X , e ao final de um passo de integração a coordenada x de um átomo é $X + \Delta x$, com $\Delta x > 0$, nós a substituímos por Δx , e semelhantemente nos demais limites da amostra.

Vamos denominar o programa principal e seus subprogramas pelas mesmas palavras usadas em “dinamol1”, mas retirando o algarismo final “1”, ou seja, nossos programas se chamarão: “dinamol.sce”, “inicia.sce”, “vizinhas.sce”, etc., porque esta é a versão mais completa do programa de dinâmica molecular. O subprograma “reflete1” será substituído pelo subprograma “repositiona”.

Vamos examinar, um a um, todos os subprogramas, chamando atenção para as diferenças em relação à abordagem nível 1.

1) Programa principal: dinamol.sce

É igual a dinamol1.sce, exceto pelos nomes dos subprogramas que chama: inicia, vizinhas, integra, potencial. Nos comentários iniciais, substitui-se "paredes perfeitamente refletoras" por "condições de contorno periódicas". Por isso não vamos repeti-lo. Para executar dinamol.sce, o leitor pode copiar dinamol1.sce com o nome dinamol.sce e fazer as referidas substituições.

2) Subprograma: inicia.sce

É quase igual ao inicial.sce. Entretanto, convém introduzir um dispositivo que garante que nossa amostra tenha, no mínimo, 4 filas e 4 colunas de células, pois no cálculo das forças nós usaremos um artifício que só funciona corretamente se $Lx \geq 4$ e $Ly \geq 4$. Por isso sugerimos introduzir, logo após as linhas que calculam Lx e Ly , a seguinte instrução:

```
if(Lx<4 | Ly<4);
    disp('erro: ceil(X/1) ou ceil(Y/1) < 4')
    halt
end;
```

Recebendo esta mensagem o operador digita "ctrl c" para recomeçar ou "enter" para continuar, ignorando a mensagem de erro.

A linha que define Nf pode ser apagada.

3) Subprograma: vizinhas.sce

Neste subprograma há diferenças muito importantes em relação ao vizinhas1.sce. Naquele, células nas beiradas da amostra têm como vizinhas a célula fictícia Nf , vazia de átomos. Aqui, em lugar da Nf as vizinhas são as células reais que estão na beirada oposta da amostra. Por exemplo, reportando-nos à Figura 6.1, $V1(16) = 9$, $V2(9) = 24$, $V3(58) = 2$, $V2(57) = 8$ e $V4(64) = 1$. Por ser bastante diferente de vizinhas1.sce, apresentemos explicitamente o subprograma vizinhas.sce.

Subprograma: vizinhas.sce

```
// Enumera as células vizinhas à direita e abaixo de
// cada célula enumerada sequencialmente;
```

```
v1=zeros(1,Nc); v2=zeros(1,Nc);
v3=zeros(1,Nc); v4=zeros(1,Nc);
// vizinhas das células do corpo principal:
for i=1:Lx-1;
    for j=2:Lx-1;
        c=ncel(i,j); //número sequencial da célula (i,j);
        v1(c)=ncel(i,j+1); // vizinha à direita;
        v2(c)=ncel(i+1,j-1); //vizinha abaixo à esquerda;
        v3(c)=ncel(i+1,j); // vizinha abaixo;
        v4(c)=ncel(i+1,j+1); // vizinha abaixo à direita;
    end
end
// vizinhas das células da primeira coluna:
for i=1:Ly-1;
    c=ncel(i,1);
    v1(c)=ncel(i,2);
    v2(c)=ncel(i+1,Lx); // vizinha à esquerda é a
                           // ... correspondente da última coluna
    v3(c)=ncel(i+1,1);
    v4(c)=ncel(i+1,2);
end
// vizinhas das células da última coluna:
for i=1:Ly-1;
    c=ncel(i,Lx);
    v1(c)=ncel(i,1);
    v2(c)=ncel(i+1,Lx-1);
    v3(c)=ncel(i+1,Lx);
    v4(c)=ncel(i+1,1);
end
// vizinhas das células da última fila:
for j=2:Lx-1;
    c=ncel(Ly,j);
    v1(c)=ncel(Ly,j+1);
    v2(c)=ncel(1,j-1);
    v3(c)=ncel(1,j);
    v4(c)=ncel(1,j+1);
end
```

```

// vizinhas da célula no vértice inferior esquerdo
c=ncel(Ly,1);
v1(c)=ncel(Ly,2);
v2(c)=ncel(1,Lx);
v3(c)=ncel(1,1);
v4(c)=ncel(1,2);
// vizinhas da célula no vértice inferior direito
v1(Nc)=ncel(Ly,1);
v2(Nc)=ncel(1,Lx-1);
v3(Nc)=ncel(1,Lx);
v4(Nc)=ncel(1,1);

```

4) Subprograma: integra.sce

Este é o mesmo que o integral.sce, exceto pela troca dos nomes dos seus subprogramas: em vez de listas1, chama listas, em vez de forcas1, chama forcas, e em vez de reflete1, chama reposiciona.

5) Subprograma: listas.sce

Este possui algumas diferenças com listas1.sce. Por isso vamos apresentá-lo explicitamente abaixo:

Subprograma: listas.sce

```

// Construção das listas:
num=zeros(1,Nc); // inicia com zero át. em cada célula;
lista=zeros(Nc,2*ceil(N/Nc)); // espaço reservado
                           // ... inicialmente para as listas;
for n=1:N;
    i=ceil(y(n)/l); // fila de células onde está o át. n;
    j=ceil(x(n)/l); // coluna de células onde está o át. n;
    c=ncel(i,j); // número sequencial da célula (i,j);
    num(c)=num(c)+1; // número de átomos na célula c;
    lista(c,num(c))=n; // o átomo n está na posição
                         // ... num(c) na lista da célula c;
end

```

6) Subprograma: forcas.sce

Embora o cálculo das forças de interação entre os átomos seja diferente no presente programa do efetuado no caso de paredes refle-

toras, por que aqui teremos que considerar também as forças entre os átomos que se encontram próximos às laterais da amostra com as “imagens periódicas” dos que estão próximos às laterais opostas, esta diferença só aparece no subprograma `F_intercel`, chamado por forças. A única diferença a introduzir em `forcas.sce`, em relação a `forcas1.sce`, é a substituição do nome do subprograma chamado, `F_intercel1` por `F_intercel`.

7) Subprograma: `F_intercel.sce`

Neste subprograma são calculadas as forças de interação entre átomos pertencentes a células vizinhas, seja vizinhas físicas, seja vizinhas por imagem periódica. Inicialmente são feitas 2 listas, `listc`, dos átomos que estão na célula c e `listv`, dos que estão em suas quatro vizinhas, $V1(c)$, $V2(c)$, $V3(c)$ e $V4(c)$. Para que `F_intercel` possa saber, ao calcular a força do átomo $n2$, pertencente a uma célula vizinha, sobre o átomo $n1$, pertencente à célula c , se o $n2$ é de uma célula vizinha física ou vizinha por imagem periódica, pois neste último caso deve ser feita a correspondente translação de coordenadas, ele examina as diferenças entre as coordenadas, $\Delta x = x1 - x2$ e $\Delta y = y1 - y2$. Se encontrar $\Delta x > 2l$ ou $\Delta y > 2l$, trata-se de vizinha por imagem periódica. Para que este artifício funcione corretamente é necessário que a amostra possua pelo menos 4 filas e 4 colunas de células. Por exemplo, se o átomo $n1$ de uma célula no extremo direito tem coordenadas $[x1, y1]$ e o átomo $n2$ da célula correspondente no extremo esquerdo tem coordenadas $[x2, y2]$, então o vetor “separação” entre eles, para efeitos do cálculo da força do $n2$ sobre o $n1$ será $[x1 - (x2 + X), y1 - y2]$. Se o átomo $n1$ está numa célula da primeira fila (superior) e o $n2$ numa da última fila (inferior), então o vetor separação, para efeitos do cálculo da força do $n2$ sobre o $n1$ será $[x1 - x2, y1 - (y2 - Y)]$. Para o cálculo da força entre átomos que estão em células nos vértices opostos, como os da célula $(1,1)$ e os da célula (Ly, Lx) , as duas “translações”, i.e., tanto em x como em y , devem ser feitas.

Devido a estas importantes diferenças do subprograma `F_intercel` em relação ao `F_intercel1`, nós o apresentamos, explicitamente, a seguir.

Subprograma: F_intercel.sce

```

// Forças intercelulares: consideram-se, para cada
// célula, os átomos das células vizinhas situadas
// à direita e abaixo da mesma. Inicialmente, para
// cada célula c, constroem-se as listas dos
// átomos que estão em c, listc, e as listas dos
// átomos que estão nas 4 células vizinhas de c, listv

for c=1:Nc;
  nc=num(c); // número de átomos na célula c;
  if(nc > 0);
    listc=lista(c,1:nc); // lista os átomos na c;
    n1=num(v1(c)); // número de átomos na vizinha v1;
    n2=num(v2(c));
    n3=num(v3(c));
    n4=num(v4(c));
    nv=n1+n2+n3+n4; // número de átomos nas células
                      // ... vizinhas consideradas;
    if(nv > 0);
      // lista os átomos das 4 células vizinhas:
      listv=[];
      if(n1>0);listv=lista(v1(c),1:n1); end;
      if(n2>0);listv=[listv,lista(v2(c),1:n2)];end;
      if(n3>0);listv=[listv,lista(v3(c),1:n3)];end;
      if(n4>0);listv=[listv,lista(v4(c),1:n4)];end;
    end
  end

  // Dadas duas listas de átomos, listc e listv, cujas
  // posições são conhecidas, calcula as forças de
  // interação, derivadas do potencial de Lennard-
  // Jonnes e soma com as forças, já calculadas, com
  // os demais átomos da amostra:
  doisl=2*l;
  xc=ones(nv,1)*x(listc); // matriz nv por nc onde
                        // cada fila é a lista das coordenadas x dos
                        // átomos da listc;

```

```

yc=ones(nv,1)*y(listc); // mesmo para coordenada y;
xv=x(listv)'*ones(1,nc); // matriz onde cada coluna
    // lista as coordenadas x dos átomos da listv;
yv=y(listv)'*ones(1,nc); // mesmo para coordenadas y;
delx=xc-xv; // matriz das distâncias x dos de listv
    // aos de listc;
list1=find(delx > doisl); // encontra os vizinhos por
    // imagem periódica;
if(length(list1) > 0);
    delx(list1)=delx(list1)-X ;
end
list2=find(delx < -doisl);
if(length(list2) > 0);
    delx(list2)=delx(list2)+X;
end
dely=yc-yv;
list3=find(dely > doisl);
if(length(list3) > 0);
    dely(list3)=dely(list3)-Y;
end
list4=find(dely < -doisl);
if(length(list4) > 0);
    dely(list4)=dely(list4)+Y;
end

r2=delx.^2+dely.^2; // quadr. das dist. dos c aos v;
flj=12*(ones(r2)./(r2.^7) - ones(r2)./(r2.^4));
fpx=flj.*delx; // forças dos listv sobre os listc;
fpy=flj.*dely;
fx(listc)=fx(listc)+sum(fpx,1); // força x total
    // ... sobre os átomos da listc;
fy(listc)=fy(listc)+sum(fpy,1);
fx(listv)=fx(listv)-sum(fpx,2)'; // força total
    // ... sobre os átomos da listv;
fy(listv)=fy(listv)-sum(fpy,2)';
end

```

8) Subprograma: reposiciona.sce

Como o nome diz, este subprograma “repositiona” os átomos que saíram, num passo de integração, para fora da amostra por uma lateral, para dentro amostra, pela lateral oposta. Este subprograma de integra.sce tem o papel correspondente ao que o subprograma reflete.sce tem no integral.sce. A seguir apresentamos o reposiciona.sce:

Subprograma: reposiciona.sce

```
// Átomos fora da amostra são reposicionados na mesma,
// pelo outro lado:
eps=%eps;
list1=find(x <= 0); // identidade dos átomos que
                     // sairam da amostra à esquerda
if(length(list1) > 0);
x(list1)=X+x(list1)-eps; // reposiciona;
end
list2=find(x >= X); // identidade dos átomos que
                     // sairam da amostra à direita;
if(length(list2) > 0);
x(list2)=x(list2)-X+eps; // reposiciona;
end
list3=find(y <= 0); // identidade dos átomos que
                     // sairam por cima;
if(length(list3) > 0);
y(list3)=Y+y(list3)-eps; // reposiciona;
end
list4=find(y >= Y); // identidade dos átomos que
                     // sairam por baixo;
if(length(list4) > 0);
y(list4)=y(list4)-Y+eps; // reposiciona;
end
```

9) Subprograma: potencial.sce

É igual ao potencial1.sce, exceto pela substituição do nome de seu subprograma: chama V_intercel em vez de V_intercel1.

10) Subprograma: V_intercel.sce

Possui as mesmas diferenças em relação a V_intercell.sce como o

F_intercel.sce em relação ao *F_intercel1.sce*. Para facilitar ao leitor, apresentamo-lo também, explicitamente, a seguir:

Subprograma: *V_intercel.sce*

```
// Potenciais intercélulas:
for c=1:Nc;
    nc=num(c);
    if(nc>0);
        listc=lista(c,1:nc);
        n1=num(v1(c));
        n2=num(v2(c));
        n3=num(v3(c));
        n4=num(v4(c));
        nv=n1+n2+n3+n4;
        if(nv>0);
            listv=[];
            if(n1>0);listv=lista(v1(c),1:n1);end
            if(n2>0);listv=[listv,lista(v2(c),1:n2)];end;
            if(n3>0);listv=[listv,lista(v3(c),1:n3)];end;
            if(n4>0);listv=[listv,lista(v4(c),1:n4)];end;
        end
    end

// Dadas duas listas de átomos, listc e listv, cujas
// posições são conhecidas, calcula as energias de
// interação e soma com a energia de interação, já
// calculada, com os demais átomos da amostra

xc=ones(nv,1)*x(listc);
yc=ones(nv,1)*y(listc);
xv=x(listv)’*ones(1,nc);
yv=y(listv)’*ones(1,nc);
delx=abs(xc-xv);
dely=abs(yc-yv);
list1=find(delx > doisl);
list2=find(dely > doisl);
if(length(list1) > 0);
```

```
    delx(list1)=X-delx(list1);
end
if(length(list2) > 0);
    dely(list2)=Y-dely(list2);
end
r2=delx.^2+dely.^2;
Vp=ones(r2)./(r2.^6) - 2*ones(r2)./(r2.^3);
V=V+sum(Vp);
end
```

Exercício 5.2:

- Monte o programa **dinamol.sce**, com seus subprogramas, como explicado acima. Faça-o funcionar com diferentes valores dos parâmetros, observando, no caso de baixa temperatura, sua transição para uma estrutura cristalina hexagonal e em alta temperatura, para um estado gasoso. Observe também a conservação da energia total e a variação da temperatura que acompanha a mudança de estrutura;
- Introduza, nas condições iniciais, uma velocidade preferencial para um lado, e observe que se estabelece uma “corrente”; na medida da temperatura, neste caso, é preciso lembrar que a mesma é a energia cinética média **em relação à velocidade do centro de massa**.

Capítulo 6

MÉTODO MONTE CARLO

Neste capítulo apresentamos um poderoso método de simulação destinado principalmente ao estudo das propriedades de sistemas materiais em estado de equilíbrio termodinâmico. A ferramenta básica para este estudo é a Mecânica Estatística. O método Monte Carlo, de simulação numérica, está, por isto, baseado em conceitos e princípios da Mecânica Estatística de Equilíbrio. Para benefício dos leitores que não estejam muito familiarizados com este ramo da Física Teórica, iniciamos este capítulo por um resumo dos seus principais conceitos, princípios e resultados. Entre os bons livros de Mecânica Estatística destacamos, para orientação do leitor que desejar aprender mais sobre o tema, as referências [16, 17, 18, 19, 20].

Muitos trabalhos de pesquisa tem sido realizados com o método Monte Carlo e também sobre ele. Para um estudo mais detalhado do que pretendemos apresentar neste texto recomendamos o recente livro de Newmann e Barkema, “Monte Carlo Methods in Statistical Physics”[21].

6.1 Elementos de Mecânica Estatística

Iniciamos esta seção apresentando o *Objeto da Mecânica Estatística*, na segunda subseção apresentamos seus conceitos fundamentais, na terceira argumentaremos uma justificativa de porque a Mecânica Estatística de Equilíbrio funciona (explica as propriedades termodinâmicas) e na quarta apresentamos a chamada *Distribuição de Boltzmann*.

mann, que é a distribuição de probabilidade para os microestados no caso de um sistema termodinâmico em equilíbrio a uma dada temperatura e o *Princípio do Balanço Detalhado*, que relaciona as probabilidades de transição com a distribuição de equilíbrio.

6.1.1 O Objeto da Mecânica Estatística

Imaginemos descrever o comportamento dinâmico de uma porção macroscópica de matéria (gás, líquido ou sólido), tratando-a como sistema mecânico constituído de N partículas microscópicas (átomos ou moléculas), com $N \approx 10^{20}$. Se usarmos Mecânica Clássica, tratando cada átomo ou molécula como um corpo sólido, puntiforme ou não, teremos um sistema de $6N$, ou mais, equações diferenciais acopladas (devido às interações) a resolver. Embora essa tarefa seja impossível de realizar na prática, vamos admitir, por ora, que consigamos obter a solução geral e, além disso, aplicar condições iniciais suficientes para obter uma solução particular. Esta solução particular estaria armazenada na memória de um computador imenso (tão grandes não existem hoje) e nos interessaria comparar nossa solução com observações experimentais macroscópicas (pressão, temperatura, densidade, momento magnético, corrente elétrica, etc). Ora, essas quantidades macroscópicas são, essencialmente, somas das contribuições dos constituintes microscópicos (para o caso de corrente elétrica teríamos que ter estendido o tratamento microscópico ao nível eletrônico). Efetuadas as somas apropriadas, encontrariam, digamos, a quantidade macroscópica $d(\mathbf{x}, t)$ (que pode ser qualquer uma das mencionadas acima), sendo \mathbf{x} o vetor posição na amostra e t o tempo. Veríamos que não faria sentido comparar $d(\mathbf{x}, t)$, assim obtido, com o correspondente resultado experimental: $d(\mathbf{x}, t)$ apresentaria enormes variações no tempo e na posição, isto é, variações extremamente rápidas ($\tau \approx 10^{-12} \text{ seg}$) e em pequenas distâncias ($\lambda \approx 10^{-8} \text{ cm}$), que, evidentemente, nossas medidas macroscópicas não observam. O remédio seria então realizar médias sobre intervalos de tempo e regiões espaciais comparáveis aos da resolução experimental. Vamos denotar o resultado dessas médias por $\bar{d}(\mathbf{x}, t)$. Esta função será, então, comparável aos correspondentes resultados experimentais. Se d apresentar variações em \mathbf{x} e t , essas serão em distâncias e tempos macroscópicos.

A Mecânica Estatística é o ramo da Física que trata do cálculo de *d sem seguir o roteiro (impraticável) descrito acima!* A Mecânica Estatística calcula as referidas médias sem resolver as equações de movimento dos constituintes microscópicos e também obtém relações entre as correspondentes variáveis macroscópicas (termodinâmica) a partir da constituição microscópica da amostra macroscópica. A Mecânica Estatística geralmente não efetua as médias *espacio-temporais*, referidas acima, mas uma outra média, chamada *média de ensemble* que também resulta nas variáveis macroscópicas.

6.1.2 Conceitos Fundamentais da Mecânica Estatística

Microestado

Em linguagem de Mecânica Clássica um “microestado” é o conjunto de valores, num dado instante, de todas as coordenadas generalizadas e seus momenta conjugados dos constituintes microscópicos da amostra. Em outras palavras, dadas todas as posições e velocidades num dado instante, se tem a descrição do microestado naquele instante. Em Mecânica Quântica o microestado é descrito pela função de onda do sistema.

Espaço de Fases

É um conceito da Mecânica Clássica. Corresponde ao espaço formado pelo conjunto de todos os pontos cujas coordenadas são as coordenadas generalizadas e seus momenta conjugados. Denotando por s o número de coordenadas generalizadas, a dimensão do espaço de fases é $2s$.

Ensemble

É um conjunto (assembleia) muito grande ($N \rightarrow \infty$) de sistemas idênticos (cópias mentais) à amostra, do ponto de vista macroscópico.

A distribuição desses sistemas nos diversos microestados é da seguinte forma:

- a) Em Mecânica Clássica a densidade de pontos em cada “volume clementar” $d\Gamma$ do espaço de fases é proporcional à probabilidade de que a amostra esteja, naquele instante, num ponto dentro de $d\Gamma$ ($= dq_1 dq_2 \cdots dq_s, dp_1, \dots, dp_s$).
- b) Em Mecânica Quântica, o número de sistemas em um microestado m é proporcional à probabilidade de que a amostra esteja em m . A construção de um ensemble depende, portanto, do conhecimento da distribuição de probabilidade para os microestados.

Média de Ensemble

Média de ensemble de uma variável A (variável dinâmica, em Mecânica Clássica, ou função de uma ou mais variáveis dinâmicas, ou observável, em Mecânica Quântica), denotada por $\langle A \rangle$ é a média aritmética dos valores de A sobre todos os sistemas do ensemble,

$$\langle A \rangle \equiv \frac{1}{N} \sum_{\alpha=1}^N A(\alpha) \quad (6.1)$$

onde α é um sistema do ensemble e $A(\alpha)$ o valor de A nesse sistema.

Uma maneira alternativa, mas equivalente, de calcular $\langle A \rangle$, faz uso, na notação de estados discretos, da Mecânica Quântica, da probabilidade P_m de encontrar o sistema (amostra) no microestado m . Sendo A_m o valor de A neste microestado, então,

$$\langle A \rangle = \sum_m P_m A_m \quad (6.2)$$

Em Mecânica Clássica, em vez de soma sobre estados m se tem uma integral sobre o espaço de fases,

$$\langle A \rangle = \int d\Gamma \rho(q, p) A(q, p) \quad (6.3)$$

onde $d\Gamma = dq_1 dq_2 \cdots dq_s dp_1 \cdots dp_s$, $\rho(q, p)$ é uma função proporcional à densidade de pontos e é normalizada de maneira que

$$\int d\Gamma \rho(q, p) = 1$$

e $A(q, p)$ é o valor de A no ponto (q, p) do espaço de fases.

MacroEstado ou Estado Macroscópico

Também chamado de “Estado Termodinâmico”, fica especificado por um conjunto de variáveis macroscópicas, $M = (\bar{A}, \bar{B}, \bar{C}, \dots)$. Quando o sistema for homogêneo e em equilíbrio, corresponde à especificação dos tipos e quantidades de átomos ou moléculas que o compõem, seu volume, temperatura, campos aplicados, e, talvez, outras variáveis macroscópicas. Se não estiver em equilíbrio, a especificação do macroestado pode exigir o conhecimento de algumas densidades $\bar{d}(\mathbf{x}, t)$, mas aqui vamos tratar exclusivamente de sistemas em equilíbrio. O conjunto de variáveis macroscópicas que especificam o estado macroscópico é "completo" quando as outras variáveis macroscópicas do sistema em equilíbrio ficam univocamente determinadas em consequência da especificação dos valores das variáveis do conjunto. Por exemplo, um gás ideal com N átomos, em equilíbrio num volume V , à temperatura T , tem sua pressão univocamente determinada. Um monocrystal de magnetita, à temperatura T e em presença de um campo magnético uniforme \mathbf{H} , tem sua magnetização de equilíbrio univocamente determinada. Quando dizemos que uma variável “tem seu valor univocamente determinado”, não significa que nós o conhecemos, mas significa que ele existe. Se fizermos sucessivas medidas da variável, mantendo o sistema em equilíbrio e os mesmos valores das variáveis do “conjunto completo”, os valores obtidos para a referida variável serão sempre os mesmos, a menos do erro de medida.

Correspondência entre microestados e macroestados

A cada microestado m corresponde um macroestado $M = (\bar{A}, \bar{B}, \bar{C}, \dots)$, ou seja, especificados os valores das variáveis microscópicas do sistema, as variáveis macroscópicas também ficam com seus valores univocamente determinados. O inverso não é verdadeiro, pois a cada estado macroscópico há uma miríade de estados microscópicos que lhe correspondem, ou seja, dados os valores das variáveis macroscópicas, existe uma enormidade de possibilidades de distribuir as variáveis microscópicas para que os valores das macroscópicas sejam os dados. Vamos denotar por $m \rightarrow M$ todo microestado m que corresponde ao macroestado M .

6.1.3 Porque a Mecânica Estatística Funciona

Por que $\langle A \rangle$ corresponde ao valor macroscópico de A , ou seja, por que $\langle A \rangle = \bar{A}$? Responder a esta pergunta é responder porque a Mecânica Estatística “funciona”. À primeira vista esta equiparação parece infundada, uma vez que o resultado experimental \bar{A} é obtido pela observação (medida) em um único sistema, enquanto que $\langle A \rangle$ é o resultado de uma média sobre um número muito grande, $N \rightarrow \infty$, de sistemas, distribuídos sobre todos os possíveis microestados, inclusive os microestados cujos correspondentes valores de A são diferentes do observado, desde que os vínculos e as variáveis macroscópicas que definem o macro estado do sistema sejam as mesmas, ou seja, sobre todos os microestados $m \rightarrow M$.

Existem diversas maneiras de responder a esta pergunta. Eu entendo esta aparente contradição pelo argumento que segue.

Dados os valores das variáveis do conjunto completo, M , que especifica o estado macroscópico, a média de *ensemble* (valor esperado) $\langle A \rangle$, de A , fica determinado por

$$\langle A \rangle = \sum_{m \rightarrow M} P_m A(m)$$

Mesmo admitindo que todos os microestados $m \rightarrow M$ tenham a mesma probabilidade (caso do ensemble microcanônico da Mecânica Estatística), o número de microestados m para os quais a variável A tem o valor experimental \bar{A} é tão maior que o número daqueles para os quais $A(m) \neq \bar{A}$, que a contribuição destes últimos para a média acima é totalmente desprezível. Uma prova disto é que medindo-se várias vezes A , desde que o sistema permaneça no mesmo estado de equilíbrio, obtém-se sempre o mesmo valor \bar{A} , embora é claro que cada vez que se opera a medida macroscópica o microestado m seja outro. Uma explicação matemática, baseada na teoria de probabilidade, para este fenômeno, faz uso do teorema central de limite: Como A é a soma de um enorme número, digamos N , de contribuições microscópicas a_j , (aproximadamente) independentes

$$A = \sum_{j=1}^N a_j$$

a distribuição de probabilidade $P(A)$ tem largura proporcional a \sqrt{N} , enquanto que seu valor mais provável é proporcional a N , de maneira que sua largura relativa é

$$\frac{\Delta A}{A} \sim \frac{\sqrt{N}}{N} = \frac{1}{\sqrt{N}}$$

Se $N \sim 10^{20}$, então $1/\sqrt{N} \sim 10^{-10}$, totalmente desprezível frente aos erros experimentais macroscópicos.

6.1.4 A Distribuição de Boltzmann

Consideremos um sistema físico S em equilíbrio a uma temperatura T . Se S for um sistema macroscópico a temperatura T é uma de suas variáveis macroscópicas, bem definidas. Se S for microscópico ou mesoscópico (i.e., suficientemente pequeno para que suas “variáveis termodinâmicas” apresentem flutuações importantes), dizer que está à temperatura T significa dizer que está em equilíbrio térmico com um sistema $S_2 \gg S$, que chamamos de “banho térmico”, o qual está à temperatura T . Em qualquer uma das circunstâncias acima o microestado m de S num dado instante t , é uma variável aleatória. Nestas circunstâncias a distribuição de probabilidade para m que se chama *Distribuição de Boltzmann* ou *Distribuição Canônica*, é

$$P_B(m) = \frac{\exp(-E_m/k_B T)}{\sum_{m'} \exp(-E_{m'}/k_B T)} \quad (6.4)$$

onde E_m é a energia do microestado m e k_B é a constante de Boltzmann. A demonstração da Eq.(6.4) pode ser encontrada nos bons livros de Mecânica Estatística e está baseada nos princípios de conservação da energia e maximização da entropia nos sistemas isolados em equilíbrio. O denominador da Eq.(6.4),

$$Z = \sum_{m'} \exp(-E_{m'}/k_B T) \quad (6.5)$$

se chama *função partição* e o conhecimento de sua forma funcional em termos das variáveis macroscópicas permite o estabelecimento de várias relações importantes da Termodinâmica (veja, por exemplo, os livros de Mecânica Estatística, referenciados no final do livro).

6.1.5 Energia Interna e Calor Específico

A energia interna de uma amostra é definida como a média de ensemble das energias dos microestados,

$$U \equiv \langle E \rangle = \frac{\sum_m E_m \exp(-E_m/k_B T)}{\sum_{m'} \exp(-E_{m'}/k_B T)}. \quad (6.6)$$

A partir desta definição o leitor pode obter a seguinte relação entre a energia interna e a derivada logarítmica da função partição:

$$U = k_B T^2 \frac{\partial \ln Z}{\partial T}. \quad (6.7)$$

O calor específico, C_V , definido por

$$C_V = \frac{\partial U}{\partial T}, \quad (6.8)$$

está relacionado com a variância da energia,

$$\sigma_E^2 = \langle E^2 \rangle - \langle E \rangle^2 \quad (6.9)$$

por

$$C_V = \frac{1}{k_B T^2} \sigma_E^2. \quad (6.10)$$

Deixamos por conta do leitor derivar a Eq.(6.6) em relação a T e provar a Eq.(6.10).

6.1.6 O Princípio do Balanço Detalhado

Consideremos um ensemble de sistemas físicos cuja distribuição de probabilidade para os microestados m no instante t é $P(m, t)$. Podemos relacionar as distribuições de probabilidade em dois instantes diferentes por (cf. eq.(4.5))

$$P(m, t) = \sum_{m'} P(m, t|m't') P(m', t'), \quad (6.11)$$

onde $P(m, t|m', t')$ é a probabilidade condicional de que o microestado seja m no instante t , dado que era m' no instante t' . Vamos

supor agora que a distribuição de probabilidade do nosso ensemble obedeça à seguinte relação:

$$P(m, t; m', t') = P(m', t; m, t'). \quad (6.12)$$

ou, equivalentemente,

$$P(m, t|m', t')P(m', t') = P(m', t|m, t')P(m, t'), \quad (6.13)$$

Substituindo a Eq.(6.13) na Eq.(6.11) vem

$$P(m, t) = \sum_{m'} P(m', t|m, t')P(m, t') = P(m, t'), \quad (6.14)$$

pois $\sum_{m'} P(m', t|m, t') = 1$. Logo, a hipótese Eq.(6.12) implica em uma distribuição de probabilidade constante no tempo para os microestados, ou seja, $P(m, t) = P(m)$ é uma distribuição de equilíbrio. Vamos examinar um pouco melhor a Eq.(6.12). Como $P(m, t')$ não depende de t' , tomamos $t' = 0$:

$$P(m, t|m', 0)P(m') = P(m', t|m, 0)P(m). \quad (6.15)$$

A derivada de ambos os membros em relação a t , no instante $t = 0$, é

$$\frac{dP(m, t|m', 0)}{dt} \Big|_{t=0} P(m') = \frac{dP(m', t|m, 0)}{dt} \Big|_{t=0} P(m) \quad (6.16)$$

As derivadas acima são conhecidas como probabilidades de transição e denotadas por $\Lambda_{mm'}$ e $\Lambda_{m'm}$, respectivamente. A Eq.(6.16) pode, portanto, ser escrita na forma

$$\Lambda_{mm'}P(m') = \Lambda_{m'm}P(m). \quad (6.17)$$

A Eq.(6.17) costuma ser chamada de “Princípio do Balanço Detalhado (PBD)”. Como vimos, ela representa condição suficiente para que a distribuição $P(m)$ seja de equilíbrio. Se re-escrevemos a Eq.(6.17) na forma

$$\frac{P(m)}{P(m')} = \frac{\Lambda_{mm'}}{\Lambda_{m'm}}, \quad (6.18)$$

evidencia-se uma interpretação óbvia: Em equilíbrio um microestado m será tanto mais populado (maior $P(m)$) quanto maiores forem as probabilidades de transição dos demais estados para ele ($\Lambda_{mm'}$) e quanto menores forem as probabilidades de transição dele para os demais estados ($\Lambda_{m'm}$).

Dadas as probabilidades de transição, $\Lambda_{m,m'}$, a Eq.(6.18) determina as probabilidades $P(m)$, dos microestados, para o ensemble em equilíbrio, a menos de uma constante multiplicativa. Como a condição de normalização das probabilidades, $\sum P(m) = 1$, determina a constante multiplicativa, segue que a Eq.(6.18) determina as $P(m)$ univocamente.

6.2 Média de Ensemble por Amostragem

A expressão “Método de Monte Carlo” refere-se ao procedimento numérico de buscar a solução de um problema por uma abordagem que envolve o sorteio de uma grande quantidade de números aleatórios e o cálculo de médias apropriadas. Na seção 6.1.2 introduzimos o conceito de “média de ensemble” na Eq.(6.1). No ensemble ideal temos um número muito grande, N_m , de sistemas hipotéticos em cada microestado m , proporcional à probabilidade de se encontrar o sistema físico em estudo no microestado m . Num cálculo de média por Monte Carlo nós usamos a primeira das igualdades da Eq.(6.1), i.e.,

$$\langle A \rangle \equiv \frac{1}{N} \sum_{\alpha=1}^N A(\alpha), \quad (6.19)$$

mas, em vez de somar sobre todos os N sistemas do ensemble, o que seria, na maioria dos casos, impraticável, nós sorteamos uma amostra de N sistemas, distribuídos pelos microestados conforme a distribuição de probabilidade $P(m)$, ou seja, nós sorteamos N microestados, $m = m_1, m_2, \dots, m_j, \dots, m_N$ com probabilidades $P(m_j)$, e substituímos a soma sobre α pela soma sobre os microestados da amostra,

$$\langle A \rangle = \frac{1}{N} \sum_{m=m_1}^{m_N} A(m) . \quad (6.20)$$

O algoritmo de Metropolis, descrito na seção 6.3, é uma maneira de obter uma tal amostra, distribuída segundo a distribuição de Boltzmann, $P_B(m)$.

6.2.1 Média por Amostragem Tendenciosa

Vamos supor que a distribuição de probabilidade para os microestados do ensemble seja $P(m)$ e que a distribuição de microestados da nossa amostra seja $P_0(m) \neq P(m)$. Isto pode ocorrer por várias razões. Por exemplo, se $P(m)$ é uma distribuição próxima de Gaussiana em alguma variável. Neste caso será muito simples sortear uma amostra com a distribuição Gaussiana $P_0(m)$, próxima de $P(m)$. Ou então queremos aproveitar uma única amostra sorteada para calcular médias em sistemas físicos em macroestados diferentes daquele para o qual a amostra foi sorteada. A soma sobre a amostra sorteada com distribuição $P_0(m)$, na Eq.(6.20), se a amostra for suficientemente grande, deve levar ao mesmo resultado (ou próximo deste) que uma soma sobre todos os microestados m em que cada termo é multiplicado por $P_0(m)$. Mas o que procuramos é a média obtida pela soma sobre todos os m em que cada termo é multiplicado por $P(m)$. Por isso corrigimos a Eq.(6.20) multiplicando cada termo por $P(m)$ e dividindo por $P_0(m)$,

$$\langle A \rangle = \frac{1}{N} \sum_{m=m_1}^{m_N} \frac{P(m)}{P_0(m)} A(m) , \quad (6.21)$$

Esta forma de calcular médias é usada no chamado “método do histograma”, que descreveremos na seção 6.3.3.

6.3 O Algoritmo de Metropolis

Neste capítulo nós abordaremos uma versão particular do Método de Monte Carlo, conhecida como “Algoritmo de Metropolis”[22]. O problema que este algoritmo se propõe a resolver é encontrar os valores

das variáveis macroscópicas de um sistema físico em equilíbrio a uma dada temperatura, T .

Como vimos na seção anterior, nestas circunstâncias a distribuição de probabilidade para os microestados é a distribuição de Boltzmann, Eq.(6.4). Os valores de equilíbrio das variáveis macroscópicas podem ser calculados, portanto, em princípio, como as médias de seus valores em todos os microestados compatíveis com os vínculos, tendo a distribuição de Boltzmann como peso. Este procedimento, entretanto, só pode ser utilizado, na prática, em modelos particularmente simples, pois, em geral, o número de microestados é tão grande e sua complexidade é tal que efetuar a referida média é tarefa impossível. A ideia básica do método consiste em partir de um microestado inicial arbitrário e efetuar uma sequência, muito grande, de transições aleatórias, por um procedimento que descreveremos a seguir, até encontrar o macroestado de equilíbrio, ou seja, até que as variáveis macroscópicas, obtidas como médias sobre os microestados percorridos, adquiram valores constantes.

Um sistema físico é dito “ergódico” quando qualquer de seus microestados pode ser atingido, a partir de qualquer outro, por uma sequência de transições. Isto é mais uma condição sobre o conjunto de transições possíveis do que sobre o conjunto de microestados. Por exemplo, se uma das variáveis que definem os microestados assume valores inteiros e só ocorrem transições em que esta variável varia de dois, estados em que ela é par nunca podem ser atingidos a partir de estados em que ela é ímpar, e vice versa e este sistema não seria ergódico. Uma consequência importante da ergodicidade é que uma média temporal, sobre um tempo suficientemente longo, dá o mesmo resultado que uma média de ensemble. Alguns autores tomam a equivalência entre estas duas médias como definição de ergodicidade.

Como no algoritmo de Metrópolis efetuam-se médias temporais em vez de médias de ensemble, é claro que uma exigência para sua validade é que o sistema seja ergódico. Outra condição é que uma sequência aleatória de transições leve ao macroestado de equilíbrio. Como nós não estamos interessados no “caminho” pelo qual o sistema chega ao equilíbrio a partir do microestado inicial arbitrário, temos a liberdade de escolher qualquer sequência de transições, mesmo que ela não seja fisicamente plausível, desde que a distribuição final de mi-

croestados seja a de equilíbrio, ou seja, a distribuição de Boltzmann. Para atingir este fim recorremos ao “princípio do balanço detalhado (PBD)”, ou seja, vamos escolher os $\Lambda_{m,m'}$ de maneira a satisfazer a Eq.(6.17) com $P(m)$ dado pela distribuição de Boltzmann, Eq.(6.4).

No procedimento numérico efetua-se uma sequência de sorteios de transições entre os microestados. Por analogia com a evolução temporal podemos dizer que o primeiro sorteio ocorre no tempo 1, o segundo no tempo 2, etc. Assim os instantes discretizados de tempo serão $t = 1, 2, 3, \dots$. O PBD, na forma da Eq.(6.13), será escrito como

$$P(m, t+1 | m', t)P(m', t) = P(m', t+1 | m, t)P(m, t). \quad (6.22)$$

O procedimento de sorteio das transições pelo algoritmo de Metrópolis é como segue; digamos que no “instante t ” o sistema esteja no microestado m :

- 1) Sorteamos um microestado m' , numa forma que seja adequada ao modelo de sistema físico em questão; digamos que a probabilidade de sortearmos m' , dado que o sistema está em m é $S(m'|m)$.
- 2) Se a energia do estado m' for menor ou igual a de m , i.e., $E_{m'} \leq E_m$, aceita-se m' como o novo estado e passa-se ao item 4, abaixo; se $E_{m'} > E_m$, procede-se conforme o item 3;
- 3) Seja $\Delta E = E_{m'} - E_m > 0$; sorteamos um número r , com distribuição uniforme entre 0 e 1; se $r \leq \exp(-\Delta E/k_B T)$ aceita-se a transição; se $r > \exp(-\Delta E/k_B T)$, rejeita-se a transição e retorna-se ao item 1;
- 4) Calculam-se as macrovariáveis, que geralmente são somas sobre as variáveis microscópicas, e guardam-se os resultados.
- 5) Após um número relativamente grande de sorteios, que deve ser escolhido caso a caso, efetuam as médias das macrovariáveis que foram guardadas, cf. item 4.

Abaixo comentaremos sobre alguns cuidados que devem ser tomados ao executar o algoritmo acima, mas antes vamos mostrar que ele satisfaz o PBD, desde que o procedimento de sorteio de microestado, item 1, satisfaça a relação

$$S(m'|m) = S(m|m') , \quad (6.23)$$

ou seja, a probabilidade de ser sorteado m' quando se está em m é igual à de ser sorteado m quando se está em m' . Os ítems 2 e 3 acima podem ser resumidos na seguinte “probabilidade de aceitação do sorteio”, $A(m'|m)$, definida na forma

$$A(m'|m) = \begin{cases} 1 & \text{se } E_{m'} \leq E_m \\ \exp[(E_m - E_{m'})/k_B T] & \text{se } E_{m'} > E_m \end{cases} \quad (6.24)$$

Assim a probabilidade condicional de o estado ser m' em $t+1$ dado que seja m em t , pode ser escrita na forma

$$P(m', t+1 | m, t) = A(m'|m)S(m'|m) , \quad (6.25)$$

pois, para ocorrer a transição $m \rightarrow m'$ é necessário que m' seja sorteado e que a transição seja aceita; como estes dois eventos são independentes, a probabilidade de ocorrerem ambos é o produto das probabilidades de cada um. Usando a Eq.(6.25) e substituindo $P(m)$ pela distribuição de Boltzmann, $P_B(m)$, a Eq.(6.22) fica escrita na forma

$$A(m'|m)S(m'|m)P_B(m) = A(m|m')S(m|m')P_B(m') \quad (6.26)$$

Vamos supor $E_{m'} > E_m$, o que não representa nenhuma perda de generalidade, devido à simetria da Eq.(6.26) frente à troca de m por m' . O lado esquerdo da Eq.(6.26), usando as eqs.(6.4) e (6.24), vale

$$\begin{aligned} \exp(-(E_{m'} - E_m)/k_B T)S(m'|m)\exp(-E_m/k_B T) &= \\ &= \exp(-E_{m'}/k_B T)S(m'|m) \end{aligned}$$

e o lado direito vale

$$S(m|m') \exp(-E_{m'}/k_B T).$$

Portanto, se o procedimento de sorteio for tal que $S(m|m') = S(m'|m)$, os dois membros da Eq.(6.26) são iguais, ou seja, o PBD está satisfeito.

Como mencionado acima, alguns cuidados devem ser tomados ao se aplicar o algoritmo de Metrópolis:

1) É possível (e provável) que o estado inicial escolhido arbitrariamente seja consideravelmente diferente dos estados prováveis do sistema em equilíbrio. Neste caso a sequência de transições levará algum "tempo" (número de transições) até que os microestados passem a percorrer estados prováveis. Esse tempo se chama "tempo de relaxação ao equilíbrio". Durante esse "tempo" as variáveis macroscópicas não devem ser calculadas, ou, se elas forem calculadas, seus valores só devem ser considerados para as médias de equilíbrio após o tempo de relaxação, isto é, quando seus valores passarem a oscilar em torno de valores constantes.

2) A regra escolhida para os sorteios das transições deve ser tal que a probabilidade de serem sorteados microestados com energia muito mais alta que a do estado de origem seja pequena ou nula. Isto se consegue sorteando apenas transições para microestados muito próximos do microestado de origem (é usual sortear alteração no microestado de apenas um átomo, e, mesmo assim, alteração moderada). Microestados com energia muito mais alta que a do estado de origem serão quase sempre rejeitados, tornando o procedimento muito ineficiente.

3) Os sorteios não devem restringir-se a microestados demasia-damente próximos do estado de origem, pois assim necessitariam de um número exageradamente grande de transições para atingir o "equilíbrio" e para produzir um conjunto estatisticamente significativo de microestados.

Portanto, as transições permitidas não devem ser muito grandes (item 2 acima) nem exageradamente pequenas (item 3 acima). Nos exemplos que passamos a apresentar, a seguir, essas recomendações devem ficar mais claras.

6.3.1 Primeiro Exemplo: O Modelo de Ising

Em 1926 Werner Heisenberg e Paul Dirac, em trabalhos independentes, mostraram que, devido ao princípio de exclusão de Pauli, a energia de interação devida à repulsão Coulombiana entre dois elétrons depende das orientações relativas de seus spins e que este efeito pode ser incorporado ao Hamiltoniano de um sistema eletrônico acrescentando-se aos demais termos (energia cinética e energia Coulombiana Clássica) um termo da forma

$$\mathcal{H}_H = -\frac{1}{2} \sum_{k,l} J_{k,l} \mathbf{S}_k \cdot \mathbf{S}_l , \quad (6.27)$$

onde S_k é o spin do k^{mo} elétron e $J_{k,l}$ são constantes, conhecidas hoje por “integrais de troca”, e podem, em princípio ser calculadas por Mecânica Quântica [25]. A Eq.(6.27) é conhecida como “Hamiltoniano de Heisenberg”.

Muito trabalho foi realizado sobre este Hamiltoniano. Em especial, ele passou a ser usado como Hamiltoniano de interação entre spins atômicos. No caso de átomos vizinhos na rede cristalina a interação vem da superposição espacial das funções de onda dos elétrons dos dois átomos, caso em que a interação se chama “troca direta”. É também possível que a interação seja indireta, mediada por um terceiro átomo, geralmente não magnético, caso em que a interação se chama de “supertroca”. As componentes cartesianas S_k^x , S_k^y e S_k^z dos vetores \mathbf{S}_k da Eq.(6.27) são operadores quânticos que não comutam entre si. Não se conhece procedimento analítico exato pelo qual se possa calcular as variáveis termodinâmicas decorrentes do Hamiltoniano de Heisenberg. Por isso são necessários métodos de aproximação. As aproximações podem ser no próprio Hamiltoniano, simplificando-o, ou nos procedimentos para calcular as variáveis termodinâmicas. No primeiro caso encontram-se as aproximações que levam ao “modelo de Ising”, que passamos a expor.

Consideremos um cristal com uma grande anisotropia uniaxial, digamos na direção do eixo z , tal que os spins atômicos se alinharam paralelamente ou antiparalelamente ao eixo z . As componentes S_i^x e S_i^y não são exatamente nulas devido à natureza quântica dos spins, mas para spins grandes não cometemos um erro muito importante se

as desprezarmos no Hamiltoniano de Heisenberg, que então adquire a forma chamada de “Hamiltoniano de Ising”:

$$\mathcal{H}_I = -\frac{1}{2} \sum_{k,l} J_{k,l} S_k^z S_l^z . \quad (6.28)$$

Como neste Hamiltoniano não há operadores que não comutam, podemos tratá-lo por Mecânica Clássica, sendo os S_k^z variáveis independentes, que só podem assumir os valores $\pm S$. Incorporando S^2 na definição dos $J_{k,l}$, podemos atribuir a S_k^z , que passaremos a denotar por s_k , os valores ± 1 . A forma mais usual do Hamiltoniano de Ising, como costuma ser usado em Mecânica Estatística, considera apenas as interações entre os vizinhos mais próximos, todos os pares com a mesma constante de interação J , e acrescentando-se a interação de cada spin com um campo magnético aplicado, H , na forma $-H s$. Com isto o Hamiltoniano de Ising, também conhecido por “Modelo de Ising”, adquire a forma

$$\mathcal{H}_I = -J \sum_{\langle k,l \rangle} s_k s_l - H \sum_k s_k . \quad (6.29)$$

onde $\langle k,l \rangle$ indica que a soma é sobre os pares de vizinhos.

Este modelo, pela sua simplicidade, tem sido muito usado em Mecânica Estatística, principalmente no estudo de transições de fase. Seu emprego, hoje, em Mecânica Estatística transcende à sua origem no Hamiltoniano de Heisenberg, sendo considerado pela maioria dos pesquisadores da área como um modelo com existência própria e tem sido usado também no estudo de sistemas não magnéticos, cujas variáveis se caracterizam por poderem assumir dois valores. Em 1944 Lars Onsager conseguiu calcular exatamente a função partição do modelo de Ising numa rede bi-dimensional, na ausência de campo magnético, encontrando que a temperatura de transição de fase, T_c (temperatura crítica), em unidades em que $J/k_B = 1$, é $T_c = \frac{1}{2} \log(1 + \sqrt{2}) \simeq 2.27$.

O modelo de Ising é particularmente apropriado para aplicação do algoritmo de Metropolis. Consideremos uma amostra retangular de dimensões $L \times N$, ou seja, L spins em cada fila e N spins em cada coluna. A passagem para amostra tri-dimensional é trivial, mas usamos neste exemplo amostra bi-dimensional para facilitar a

visualização e para reduzir a exigência de tempo de CPU, já que nosso propósito aqui é apenas pedagógico. Cada spin será identificado pelo par de inteiros (i, j) , coordenadas do sítio onde ele se encontra, ou seja, em vez de s_k usaremos a notação $s(i, j)$. Inicia-se o processo de Monte Carlo atribuindo-se a cada spin $s(i, j)$ um valor +1 ou -1. Os itens 1 a 5 do algoritmo de Metropolis são realizados como segue:

- 1) Sorteamos, com distribuição uniforme, o par de números (i, j) e a transição proposta é a inversão (troca de sinal) do spin $s(i, j)$.
- 2) Calcula-se, pela Eq.(6.29), a variação de energia ΔE correspondente a esta transição. Se $\Delta E < 0$, aceita-se a transição, alterando-se a matriz S , cujos elementos são os $s(i, j)$, pela substituição $s(i, j) \rightarrow -s(i, j)$ e atualizando a energia, $E \rightarrow E + \Delta E$. Segue-se então para o item 4, abaixo.
- 3) Caso $\Delta E > 0$, calcula-s $W = \exp(-\Delta E/T)$ (usamos $k_B = 1$) e sorteia-se um número aleatório uniforme r entre 0 e 1. Se $r \leq W$ aceita-se a transição, procedendo-se como no item 2. Se $r > W$ rejeita-se a transição e retorna-se ao item 1.
- 4) Os valores das variáveis macroscópicas não precisam ser registrados a cada sorteio, pois de um sorteio ao seguinte eles variam muito pouco. Chamaremos de “um passo de Monte Carlo” um número de sorteios igual ao número de spins da amostra, de maneira que em um passo de Monte Carlo cada spin é sorteado, em média, uma vez. Registraremos os valores das variáveis macroscópicas a cada passo de Monte Carlo. No nosso exemplo as variáveis que registraremos são a energia $E(t)$ (t conta os passos de Monte Carlo) e o momento magnético,

$$\mu(t) = \sum_{i,j} s(i, j) . \quad (6.30)$$

- 5) Após um número conveniente, p , de passos de Monte Carlo, que será escolhido com a prática, calculamos as médias,

$$U \equiv \langle E(t) \rangle = \frac{1}{p} \sum_{t=1}^p E(t) , \quad (6.31)$$

que é a “energia interna”, e

$$M \equiv \langle \mu(t) \rangle = \frac{1}{p} \sum_{t=1}^p \mu(t) , \quad (6.32)$$

que é a magnetização.

Calculamos também o calor específico pelas relações Eq.(6.9) e Eq.(6.10). A suscetibilidade magnética é também uma grandeza muito própria para ser calculada por Monte Carlo. Ela é definida como a derivada da magnetização em relação ao campo aplicado, no limite de campo zero,

$$\chi = \left. \frac{\partial M}{\partial H} \right|_{H=0} . \quad (6.33)$$

Por um procedimento análogo ao que usamos para demonstrar a relação entre o calor específico e a variância da energia, na seção 6.1.5, pode-se mostrar a seguinte relação entre a suscetibilidade e a variância do momento magnético:

$$\chi = \frac{1}{k_B T} \sigma_\mu^2 , \quad (6.34)$$

onde

$$\sigma_\mu^2 = \langle \mu^2 \rangle - \langle \mu \rangle^2 . \quad (6.35)$$

Apresentamos abaixo as partes (subprogramas) de um programa de simulação de Monte Carlo para o modelo de Ising. O programa principal se chama "ising_mc.sce", o qual chama os subprogramas "inicia_ising.sce", "metro_ising.sce", e a "function" "plota_ising.sci".

O programa principal calcula, também, a magnetização, a energia interna, o calor específico e a susceptibilidade, a cada passo de Monte Carlo. Um gráfico em que cada sítio é indicado por um "o", em cor azul se o spin do sítio vale +1 e em cor vermelha se vale -1, é apresentado após p passos, quando também são apresentados os gráficos das variáveis calculadas. O programa permite ao operador continuar pelo número de passos que achar conveniente e mudar a temperatura quando desejar.

Programa ising_mc.sce

```
// ising_mc: Monte-Carlo para o modelo de Ising
// em uma rede retangular L x N;
clear
getf plota_ising.sci
tplot=0;
exec inicia_ising.sce
while(p>0);
    M1=zeros(1,p); M2=zeros(1,p);
    E1=zeros(1,p); E2=zeros(1,p);
    for mc=1:p;
        tplot=tplot+1;
        t(tplot)=tplot;
        exec('metro_ising.sce',-1)
        M=sum(S);      // momento magnético total
        M1(mc)=M;
        Mm(tplot)=M/LN;   // mom. magn. por spin
        M2(mc)=M*M; // quadrado do mom. magn. total
        E1(mc)=E;
        E2(mc)=E*E;
        Um(tplot)=E1(mc)/LN;
        Em(tplot)=E/LN; // energia por spin
    end
    U=sum(E1)/p;           // energia interna
    C=b^2*(sum(E2)/p-U^2); // calor específico
    disp(C,'calor especifico=')
    M=sum(M1)/p;           // magnetização de equilíbrio
    chi=b*(sum(M2)/p-M^2); // suscetibilidade
    disp(chi,'susceptibilidade=')
    xset('window',1); clf
    plota_ising(S)
    xset('window',2); clf
    plot(t,Mm)
    legend('magnetização x tempo')
    xset('window',3); clf
    plot(t,Um)
    legend('energia x tempo')
```

```

p=input('mais passos de Monte-Carlo? p > 0:sim \n ');
if(length(p)==0), break; end
Tn=input('deh temperatura, se enter mantém \n ');
if(Tn>0), T=Tn; b=1/T; end;
end

```

A inicialização é feita pelo subprograma “`inicia_ising.sce`”, que pede os parâmetros da amostra, L e N , as constantes de interação, J_f na fila e J_c na coluna, a temperatura T , o campo H , o número p de passos de Monte Carlo a serem dados inicialmente (mais passos serão, normalmente, pedidos ao final dos p passos iniciais) e uma variável T_0 que informa se queremos iniciar com os spins todos alinhados, $T_0 = 0$, ou se preferimos iniciar com os valores dos spins (± 1) sorteados, sem correlação, caso em que escolhemos $T_0 > 0$.

Subprograma: `inicia_ising`

```

// inicia_ising: inicialização do ising_mc; rede
// retangular  $L \times N$ , com opção de estado inicial
// ordenado ( $T_0=0$ ) ou totalmente desordenado ( $T_0 > 0$ );
// calcula energia inicial; Exemplo de dados:
// [ $L=25$ ,  $N=30$ ,  $T_0=1$ ,  $T=2$ ,  $J_f=1$ ,  $J_c=1$ ,  $H=0$ ,  $p=10$ ];

par=input('deh [L, N, T0, T, Jf, Jc, H, p] \n ');
L=par(1);N=par(2);T0=par(3); T=par(4);
Jf=par(5); Jc=par(6); H=par(7); p=par(8);
b=1/T; // parâmetro beta, com  $k_B=1$ ;
LN=L*N; LN2=LN*LN; // LN é o número de spins;
if(T0==0)
    S=ones(L,N);
    E=-Jf*LN-Jc*LN; // usaremos condições de
        // contorno periódicas; cada spin tem
        // 2 vizinhos na fila e 2 na coluna; logo,
        // há  $NL$  pares nas filas e  $NL$  nas colunas;
else
    E=0;
    S=sign(rand(L,N)-0.5);
    for i=1:L-1;

```

```

E=E-Jc*S(i,:)*S(i+1,:); // interação entre a
                           // fila i a e seguinte
end
E=E-Jc*S(1,:)*S(L,:); // interação entre primeira
                           // e última fila
for j=1:N-1;
    E=E-Jf*S(:,j)*S(:,j+1); // interação entre a
                           // coluna j e a seguinte
end
E=E-Jf*S(:,1)*S(:,N); // interação entre primeira
                           // e última coluna
end
xset('window',1);clf
plota_ising(S)

```

A seguir apresentamos o subprograma “metro_ising.sce” que executa o algoritmo de Metropolis. Ele é chamado pelo programa principal uma vez a cada passo de Monte Carlo e devolve a nova configuração de spins e a nova energia.

Subprograma: metro_ising.sce

```

// metro_ising é o algoritmo de Metropolis para
// o modelo de Ising na rede retangular, L x N;
// os parâmetros são os introduzidos em inicia_ising;
// realiza um passo de Monte-Carlo (i.e., sorteia L*N
// spins) a cada chamada; atualiza spins e energia;

for n=1:LN
    // sorteio de um spin:
    i=ceil(L*rand()); j=ceil(N*rand());
    s=S(i,j);

    // cálculo da variação de energia dE:
    i1=i;    i2=i-1; i3=i;    i4=i+1;
    j1=j+1;  j2=j;   j3=j-1;  j4=j;
    if(i==1)
        i2=L;

```

```

elseif(i==L)
    i4=1;
end
if(j==1)
    j3=N;
elseif(j==N)
    j1=1;
end
soma=Jf*S(i1,j1)+Jc*S(i2,j2)+Jf*S(i3,j3)+Jc*S(i4,j4);
dE=2*s*(soma+H);

// opta se aceita ou não mudar s em -s:
if(dE<=0)
    S(i,j)=-s;
    E=E+dE;
else
    W=exp(-b*dE);
    if(rand() < W), S(i,j)=-s; E=E+dE; end
end
end

```

Para visualizar-se na tela a configuração de spins a cada p passos de Monte Carlo, o programa principal chama a função “plota_ising(S)”.

Função: plota_ising.sci

```

function plota_ising(S);

// desenha a rede com círculos azuis para
// spin=+1 e círculos vermelhos para spin=-1;

L=size(S,1); N=size(S,2);
LN=L*N;
s=S(:); // transforma a matriz S num vetor coluna;
C=[1:N]*ones(1,L); // cria uma matriz em que cada
// coluna contém os números de 1 a N;
D=ones(N,1)*[1:L]; // cria uma matriz em que cada
// fila contém os números de 1 a L;

```

```

c=C(:); // transforma C em um vetor coluna
d=D(:); // transforma D em um vetor coluna
e=d.*s; // vetor coluna com os valores dos spins
          // multiplicando os números de 1 a L, fila a
          // fila da amostra, em sequência;
f=zeros(LN,1); g=f;
for j=1:LN;
    if(e(j)>0)
        f(j)=e(j); // coloca os e(j) correspondentes a
                      // spin ‘‘up’’ no vetor coluna f;
        g(j)=%nan;
    else
        f(j)=%nan;
        g(j)=abs(e(j)); // coloca os e(j) correspondentes
                          // a spin ‘‘down’’ no vetor coluna g;
    end
end
plot(c,f,’bo’,c,g,’ro’) // configuração dos spins;

```

Exercício 6.1

Entenda o programa “Ising_mc” e todos os seus subprogramas; execute-o com diversos valores dos parâmetros e observe o comportamento das “ilhas de correlação”, após o sistema atingir um aparente equilíbrio, em diferentes temperaturas; em particular, observe esse comportamento para temperaturas entre 2 e 3, verificando que nalgum valor de T nesse intervalo há indicação de uma transição de fase “ordenada-desordenada”.

6.3.2 Segundo Exemplo: O Gás Bi-Dimensional

Como segundo exemplo vamos considerar o mesmo modelo gasoso que usamos no capítulo V, sobre Dinâmica Molecular. Diferentemente do modelo de Ising, onde trabalhamos com variáveis discretas, $s_k = \pm 1$, aqui as variáveis são as coordenadas x e y dos átomos, que podem assumir quaisquer valores dentro da caixa que contém o gás. Diferentemente do tratamento que fizemos no capítulo anterior, por Dinâmica Molecular, aqui as velocidades não integram nosso conjunto

de variáveis. Esta escolha foi feita por simplicidade. Na verdade, as velocidades fazem parte da descrição dos microestados, mas como a energia potencial depende só das posições e a cinética só das velocidades, é possível aplicar o algoritmo de Metropolis no subespaço das coordenadas. Como mostramos no capítulo V, a energia cinética média, por átomo, neste modelo bi-dimensional, com $k_B = 1$ e massa=1, é igual à temperatura, T, de maneira que para obtermos a energia interna basta calcular a energia potencial média do sistema e somar $N \times T$, onde N é o número de átomos. Como condições de contorno consideramos paredes perfeitamente reflectoras, por simplicidade, mas a transformação do programa para o caso de condições periódicas de contorno não apresenta grande dificuldade e será deixada como exercício. As paredes perfeitamente reflectoras são como barreiras infinitamente altas de potencial, ou seja, se uma transição levar um átomo a ultrapassá-las não será aceita.

Nosso programa contém o programa principal, "gas_mc.sce" e cinco subprogramas, "inicia_gas.sce", "metro_gas.sce", "vizinhas_gas.sce", "listas_gas.sce" e "potencial_gas.sce".

O programa principal chama os subprogramas para executar as diversas tarefas. Ele chama "metro_gas.sce" p vezes, para executar, cada vez, um passo de Monte Carlo, ou seja, N sorteios de transições, atualizando as coordenadas e calculando a variação da energia potencial. Introduzimos, no programa principal, um "plot" das posições atômicas após cada passo de Monte Carlo, para criar um pouco de animação. O resultado parecerá mais ou menos com um filme do movimento atômico, dependendo da velocidade do processador e do número de átomos da amostra. Ao final dos p passos é apresentado um gráfico da evolução da energia potencial. O programa dá a opção de continuar por mais tempo (mais p passos) e de alterar a temperatura.

Programa principal: gas_mc.sce

```
// gas_mc: programa principal de gas por Monte Carlo;
// a inicialização é feita em "inicia_gas";
// chama "metro_gas", que executa o algoritmo de
// Metropolis durante um passo de Monte Carlo,
```

```

// atualizando as posições das moléculas, que são
// mostradas na tela; após cada p passos de Monte Carlo
// apresenta o gráfico da energia potencial

exec('inicia_gas.sce',-1); // distribui os N átomos
                           // numa rede quadrada
exec('vizinhas_gas.sce',-1);
V=[];
tplot=1; t=[]; t(1)=0;
V(tplot)=0;
while(p>0)
  for mc=1:p;
    exec('metro_gas.sce',-1);
    xset('window',1); clf; plot(x,y,'o')
    tplot=tplot+1;
    V(tplot)=V(tplot-1)+E;
  end
  tf=length(t);
  t=[t,tf+1:tf+p];
  xset('window',2);clf; plot(t,V);
  legend('V(t)',2)
  p=input('dê novo p; se enter, encerra; \n');
  if(length(p)==0);break;end;
  Tn=input('dê nova temperatura; se enter, mantém; \n');
  if(length(Tn)>0); T=Tn; end;
end

```

O subprograma “inicia_gas”, chamado por “gas_mc”, distribui os N átomos numa rede quadrada, dentro da amostra retangular $X \times Y$ e fornece os demais parâmetros: lado das células, l , parâmetro dos deslocamentos individuais, D , número de passos Monte Carlo a serem realizados inicialmente, p , e temperatura, T . Além disso, quantifica o número de células, N_c , o número da célula fictícia, N_f , e introduz a enumeração sequencial das células, $ncel(i,j)$;

Subprograma: inicia_gas.sce

```
// inicia_gas.m: inicia programa de Monte Carlo para o
```

```

// gás bi-dimensional com interação de Lennard-Jones;
// pede as dimensões X e Y da amostra, o lado
// l das células, o número N de átomos, o deslocamento
// máximo D na transição e a temperatura T.

disp('Ex: [15 15 3 225 .1 20 1]')
disp('')
par=input('deh [X, Y, l, N, D, p, T ] \n');
X=par(1); Y=par(2); l=par(3); N=par(4);
D=par(5); p=par(6); T=par(7);
Lx=ceil(X/l); //número de células ao longo do eixo x
Ly=ceil(Y/l); //número de células ao longo do eixo y
X=Lx*l; Y=Ly*l; // acerta dimensões, ajustando-as às
S=X*Y;           // células área da amostra
a=sqrt(S/N); // parâm. de rede para encher o espaço
Nx=ceil(X/a); //número de sítios iniciais ao longo em x
Ny=ceil(Y/a);
a=min(X/Nx, Y/Ny); // se necessário, diminui "a",
//para caberem Nx átomos em X e Ny átomos em Y
x=zeros(1,N); y=zeros(1,N);

// define coordenada x inicial de cada átomo:
for n=1:N;
    z=(n-1)/Nx;
    x(n)=a*(Nx*(z-floor(z))+.5);
    y(n)=a*(ceil(n/Nx)-.5); // define coordenada y
end                         // inicial de cada átomo
Nc=Lx*Ly;       // número total de células
Nf=Nc+1;         // célula fictícia vazia de átomos

//enumeração sequencial das células:
ncel= (0:Ly-1)'*ones(1,Lx)*Lx + ones(Ly,1)*(1:Lx);

```

O subprograma "metro_gas.sce" contém o algoritmo de Metropolis. As transições entre microestados podem ser escolhidas de muitíssimas maneiras. A variação de energia em uma transição individual, se positiva, deve ser menor ou da ordem de $k_B T$ para ter chance de ser aceita. Tipicamente, isto corresponde ao deslocamento de um

átomo de sua posição. Se as distâncias entre átomos vizinhos são da ordem de 1, nas unidades em que estamos trabalhando, então um deslocamento atômico individual a uma distância ≈ 1 ou maior corresponderá, muito provavelmente a um acréscimo considerável de energia e não será aceito. Por isso, para tornar o procedimento eficiente na busca de uma distribuição de equilíbrio, temos que limitar os deslocamentos atômicos individuais a distâncias $\approx D \ll 1$. Por outro lado, se escolhermos D exageradamente pequeno o processo de relaxação ao equilíbrio também será pouco eficiente por que se precisará de um número muito grande de deslocamentos individuais para se ter uma variação considerável na configuração do sistema. Deixamos D como parâmetro de entrada do programa, antecipando que uma boa escolha é $D \approx 0.1$. Para as variações individuais das coordenadas podemos escolher D vezes um número aleatório entre -0.5 e +0.5.

Subprograma: metro_gas.sce

```
// metro_gas.sce: algoritmo de Metropolis para o modelo
// de gás bidimensional de N átomos com interação de
// Lennard-Jones; usa os parâmetros de inicia_gas;
// realiza um passo de Monte Carlo (um passo significa
// sorteio de N átomos); atualiza posições e energia;
// atualiza listas sempre que um átomo trocar de célula.

exec('listas_gas.sce',-1); //identifica os át.das células
E=0;
for nt=1:N
    n=ceil(N*rand()); // número do átomo sorteado;
    i=ceil(y(n)/l); j=ceil(x(n)/l); // célula do átomo n;
    c=ncel(i,j); // número sequencial da célula;
        // deslocamentos sorteados em x e em y:
    dx=D*(rand()-0.5); dy=D*(rand()-0.5);
    xn=x(n)+dx; yn=y(n)+dy; // novas coordenadas
    a=0; // quando a=1, abaixo, aceitou a transição
    if(xn > 0 & xn < X & yn > 0 & yn < Y), // não permite
        // transição para fora dos limites da amostra
    exec('potencial_gas.sce',-1); //calcula a variação
        // da energia potencial
```

```

if(DE <= 0),
    x(n)=xn; y(n)=yn; E=E+DE; a=1;
elseif(rand() < exp(-DE/T)),
    x(n)=xn; y(n)=yn; E=E+DE; a=1;
end;
if(a==1) // se foi aceita a transição
    in=ceil(yn/l); jn=ceil(xn/l); // a célula de n
                                    // pode ter trocado
    if(in~=i | jn~=j), // se a célula de n trocou
        nc=num(c);
        lisc=lista(c,1:nc);
        k=find(lisc==n); // encontra a posição
                            // do n na lista lisc;
        lista(c,k)=lista(c,nc); // retira o n da c
        num(c)=nc-1;
        nova=ncel(in,jn); // nova célula do átomo n;
        num(nova)=num(nova)+1;
        lista(nova, num(nova))=n; // coloca o n na
    end                                // lista de nova
end
end
end

```

O programa “vizinhos_gas.sce” especifica quais são as 8 células vizinhas, $v1(c)$, $v2(c)$, … $v8(c)$ de cada célula c , conforme convenção mostrada na Figura 6.1. Assim, por exemplo, $v1(8) = 1$, $v2(10) = 4$, $v4(11) = 10$, $v6(8) = 13$, etc. Diferentemente do procedimento usado em Dinâmica Molecular (veja capítulo V), onde calculávamos as forças sobre cada átomo da amostra, aqui teremos que calcular a variação na energia de interação do átomo sorteado com seus vizinhos. Por isso precisamos considerar todas as 8 células vizinhas à célula c que contém o átomo sorteado.

O subprograma “vizinhos_gas.sce” precisa ser usado uma única vez, no início do programa, para construir os vetores $vk(c)$, $k = 1, \dots, 8$, $c = 1, \dots, Nc$, onde $Nc = Lx \times Ly$ é o número de células. Ele usa uma célula fictícia, de número $Nf = Nc+1$, com zero átomos, para ocupar o lugar das vizinhas inexistentes. Por exemplo, na Figura 6.1, $v1(7) = Nf$, $v2(3) = Nf$, $v5(12) = Nf$, $v7(32) = Nf$, etc.

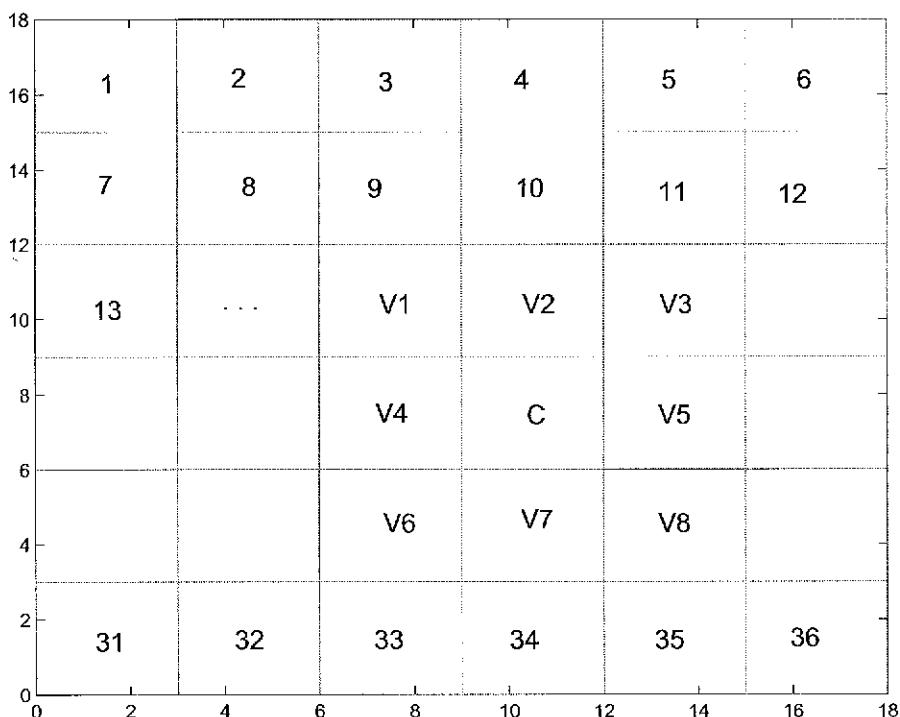


Figura 6.1: Esquema de células enumeradas, com indicação das vizinhas para cálculo da energia de interação

Este subprograma é apresentado na íntegra, a seguir.

Subprograma: vizinhas_gas.sce

```
// Enumera as 8 células vizinhas de cada célula c:
v1=zeros(1,Nc); v2=zeros(1,Nc);
v3=zeros(1,Nc); v4=zeros(1,Nc);
v5=zeros(1,Nc); v6=zeros(1,Nc);
v7=zeros(1,Nc); v8=zeros(1,Nc);

// vizinhas das células do corpo principal:
for i=2:Lx-1;
    for j=2:Lx-1;
        c=ncel(i,j); // número sequencial da célula (i,j)
```

```
v1(c)=ncel(i-1,j-1);
v2(c)=ncel(i-1,j);
v3(c)=ncel(i-1,j+1);
v4(c)=ncel(i,j-1);
v5(c)=ncel(i,j+1);
v6(c)=ncel(i+1,j-1);
v7(c)=ncel(i+1,j);
v8(c)=ncel(i+1,j+1);

end
end

// vizinhas das células da primeira fila:
for j=2:Lx-1;
c=ncel(1,j);
v1(c)=Nf; // célula fictícia, com zero átomos
v2(c)=Nf;
v3(c)=Nf;
v4(c)=ncel(1,j-1);
v5(c)=ncel(1,j+1);
v6(c)=ncel(2,j-1);
v7(c)=ncel(2,j);
v8(c)=ncel(2,j+1);
end

// vizinhas das células da primeira coluna:
for i=2:Ly-1;
c=ncel(i,1);
v1(c)=Nf;
v2(c)=ncel(i-1,1);
v3(c)=ncel(i-1,2);
v4(c)=Nf;
v5(c)=ncel(i,2);
v6(c)=Nf;
v7(c)=ncel(i+1,1);
v8(c)=ncel(i+1,2);
end
```

```
// vizinhas das células da última fila:  
for j=2:Lx-1;  
    c=ncel(Ly,j);  
    v1(c)=ncel(Ly-1,j-1);  
    v2(c)=ncel(Ly-1,j);  
    v3(c)=ncel(Ly-1,j+1);  
    v4(c)=ncel(Ly,j-1);  
    v5(c)=ncel(Ly,j+1);  
    v6(c)=Nf;  
    v7(c)=Nf;  
    v8(c)=Nf;  
end  
  
// vizinhas das células da última coluna:  
for i=2:Ly-1;  
    c=ncel(i,Lx);  
    v1(c)=ncel(i-1,Lx-1);  
    v2(c)=ncel(i-1,Lx);  
    v3(c)=Nf;  
    v4(c)=ncel(i,Lx-1);  
    v5(c)=Nf;  
    v6(c)=ncel(i+1,Lx-1);  
    v7(c)=ncel(i+1,Lx);  
    v8(c)=Nf;  
end  
  
// vizinhas da célula 1=ncel(1,1):  
v1(1)=Nf;  
v2(1)=Nf;  
v3(1)=Nf;  
v4(1)=Nf;  
v5(1)=2;  
v6(1)=Nf;  
v7(1)=Lx+1;  
v8(1)=Lx+2;  
  
// vizinhas de célula Lx=ncel(1,Lx):
```

```
v1(Lx)=Nf;
v2(Lx)=Nf;
v3(Lx)=Nf;
v4(Lx)=Lx-1;
v5(Lx)=Nf;
v6(Lx)=ncel(2,Lx-1);
v7(Lx)=ncel(2,Lx);
v8(Lx)=Nf;

// vizinhas da célula (Ly,1):
c=ncel(Ly,1);
v1(c)=Nf;
v2(c)=ncel(Ly-1,1);
v3(c)=ncel(Ly-1,2);
v4(c)=Nf;
v5(c)=ncel(Ly,2);
v6(c)=Nf;
v7(c)=Nf;
v8(c)=Nf;

// vizinhas da célula Nc=ncel(Ly,Lx):
v1(Nc)=ncel(Ly-1,Lx-1);
v2(Nc)=ncel(Ly-1,Lx);
v3(Nc)=Nf;
v4(Nc)=ncel(Ly,Lx-1);
v5(Nc)=Nf;
v6(Nc)=Nf;
v7(Nc)=Nf;
v8(Nc)=Nf;
```

O subprograma "listas_gas.sce" é chamado por "metro_gas.sce" para construir as listas dos átomos que estão em cada célula, as quais serão usadas no cálculo da variação de energia em cada transição. Estas listas serão alteradas em "metro_gas.sce" sempre que a transição trocar o átomo de célula.

Subprograma: listas_gas.sce

```

// Dadas as coordenadas cartesianas de N átomos
// numerados num espaço retangular de dimensões X
// por Y, dividido em células quadradas de lado 1,
// lista, em cada célula, os números que identificam
// os átomos cujos centros estão na mesma;
num=zeros(1,Nf); // inicia com zero áts. em cada célula;
celula=zeros(1,N);
for n=1:N;
    i=ceil(y(n)/1); // fila de células onde está o átomo
    j=ceil(x(n)/1); // coluna de cel. onde está o átomo
    c=ncel(i,j); // número sequencial da célula
    num(c)=num(c)+1; // número de átomos na célula c
    celula(n)=c;
end
numax=max(num)

lista=zeros(Nf,numax); //espaço p/ as listas
pos=zeros(1,Nc)
for n=1:N
    c=celula(n);
    pos(c)=pos(c)+1;
    lista(c,pos(c))=n; // o átomo n está na posição
end           // num(c) na lista da célula c

```

O subprograma "potencial_gas.sce" é chamado pelo subprograma "metro_gas.sce" para calcular a variação de energia potencial, ΔE logo após o sorteio de uma transição, para ser usada na decisão se a transição é aceita ou não.

Subprograma: potencial_gas.sce

```

// Calcula a variação na energia potencial de L.J.
// do átomo n com todos os seus vizinhos

```

```

nc=num(c); // número de átomos na célula c
listc=lista(c,1:nc); // lista os átomos da célula c;
k=find(listc==n); // encontra a posição do n na lista;

```

```

listc(k)=listc(1); //passa o primeiro para o lugar do n
list=listc(2:nc); //lista os ats. da cél. exceto o n
n1=num(v1(c)); //número de át. na célula vizinha v1(c);
n2=num(v2(c));
n3=num(v3(c));
n4=num(v4(c));
n5=num(v5(c));
n6=num(v6(c));
n7=num(v7(c));
n8=num(v8(c));
nv=n1+n2+n3+n4+n5+n6+n7+n8+nc-1; //núm. de vizinhos do n
if(nv>0)
  if(n1>0); list=[list,lista(v1(c),1:n1)]; end;
  if(n2>0); list=[list,lista(v2(c),1:n2)]; end;
  if(n3>0); list=[list,lista(v3(c),1:n3)]; end;
  if(n4>0); list=[list,lista(v4(c),1:n4)]; end;
  if(n5>0); list=[list,lista(v5(c),1:n5)]; end;
  if(n6>0); list=[list,lista(v6(c),1:n6)]; end;
  if(n7>0); list=[list,lista(v7(c),1:n7)]; end;
  if(n8>0); list=[list,lista(v8(c),1:n8)]; end;
// energia potencial antes do deslocamento:
delx=x(n)-x(list); // delta x do átomo n aos vizinhos
dely=y(n)-y(list); // delta y do átomo n aos vizinhos
r2=delx.^2+dely.^2; //quadr.da dist.do n aos vizinhos
Vk=ones(r2)./(r2.^6) - 2*ones(r2)./(r2.^3);
Vi=sum(Vk);
// energia potencial após o deslocamento:
delx=xn-x(list); // delta x do átomo n aos vizinhos
dely=yn-y(list); // delta y do átomo n aos vizinhos
r2=delx.^2+dely.^2; //quadr.da dist.do n aos vizinhos
Vk=ones(r2)./(r2.^6) - 2*ones(r2)./(r2.^3);
Vf=sum(Vk);
// Variação da energia de interação:
DE=Vf-Vi;
else
  DE=0;
end

```

Exercício: 6.2

Faça as modificações necessárias no programa acima para substituir as paredes refletoras por condições periódicas de contorno. Faça também os acréscimos necessários para o calor específico após o sistema ter atingido o equilíbrio.

6.3.3 O Método do Histograma

Geralmente estamos interessados em conhecer os valores das variáveis termodinâmicas não apenas em uma dada temperatura mas como função da temperatura. O procedimento descrito pelo algoritmo de Metrópolis se realiza com valor constante da temperatura. Por isso, sua aplicação direta no cálculo dos valores médios das variáveis exige tantas repetições do procedimento de simulação quantos forem os valores da temperatura para os quais queremos as referidas médias. Um método baseado em uma ideia de Valleau e Card[23], de 1972, mas desenvolvida principalmente por Ferrenberg e Swenson[24], em 1988, se tornou conhecido como o “método do histograma”. A ideia pode ser explicada a partir do que denominamos, na seção 6.2.1, “média por amostragem tendenciosa”. Digamos que realizamos uma simulação, conforme o algoritmo de Metropolis, a uma temperatura T_0 e queremos calcular $\langle A \rangle$ a uma temperatura T . As distribuições de probabilidade $P(m)$ e $P_0(m)$ que aparecem na Eq.(6.21) são as distribuições de Boltzmann, respectivamente, para as temperaturas T e T_0 , ou seja, a Eq.(6.21) tem, neste caso, a forma

$$\langle A \rangle = \frac{1}{N} \sum_{m=m_1}^{m_N} \exp(E(m)(\beta_0 - \beta)) A(m), \quad (6.36)$$

onde $\beta_0 = 1/k_B T_0$ e $\beta = 1/k_B T$.

Se executarmos a simulação à temperatura T_0 e construirmos um histograma dos estados visitados em função da energia, obteremos uma curva com aspecto semelhante a uma Gaussiana, centrada em uma energia E_0 . A largura relativa desta “distribuição de energia” será tanto menor quanto maior for o sistema. Se executarmos a simulação em outra temperatura, T , obteremos outra curva, centrada em outra energia, E . O valor estatístico da amostra de microestados obtidos ‘a temperatura T_0 para o cálculo da média à temperatura T

é proporcional à área da superposição entre os dois diagramas, pois a amostra colhida à temperatura T_0 deve conter informação sobre a distribuição de microestados à temperatura de interesse, T . Por isso as temperaturas T_0 e T não podem ser muito diferentes, ou seja, o intervalo de temperaturas sobre o qual se pode usar a mesma amostra de microestados não é muito grande. Existem procedimentos alternativos com os quais se pode alargar o intervalo de validade para uma amostra. O chamado método do “Histograma Amplo”, de P.M.C. de Oliveira, T.J.P.Penna e H.J.Herrmann[26], de 1996, é um bom exemplo.

Capítulo 7

DINÂMICA ESTOCÁSTICA

Neste capítulo veremos como se obtêm soluções numéricas para equações diferenciais que possuem termo de ruído, conhecidas por *equações diferenciais estocásticas*. Elas se classificam em equações de ruído aditivo ou multiplicativo, como explicaremos a seguir. Tais equações também são conhecidas como *Equações de Langevin*. O cálculo infinitesimal usado na sua interpretação e solução tem regras que diferem das do cálculo infinitesimal usual (integrais de Riemann) e por isso tem nome próprio, sendo chamado de *Cálculo Estocástico*. O termo *Dinâmica Estocástica* se refere à evolução temporal de variáveis aleatórias sujeitas a ruído.

7.1 Ruído Branco e Processo de Wiener

Um processo estocástico (PE) de importância especial para o “Cálculo Estocástico” é o chamado “Ruído Branco”, $\xi(t)$, definido por suas propriedades estatísticas:

- 1) $\langle \xi(t) \rangle = 0$ (média);
- 2) $\langle \xi(t_2)\xi(t_1) \rangle = \delta(t_2 - t_1)$ (função de correlação).

O motivo para se chamar este PE de ruído branco é simples: como vimos no Capítulo 4, a intensidade espectral de um PE é a transformada de Fourier da função de autocorrelação. Ora, a transformada de

Fourier da delta de Dirac é uma constante, ou seja, todas as frequências estão presentes com a mesma intensidade, o que caracteriza a luz branca. Notemos que o tempo de correlação do ruído branco é zero. É claro que na Natureza não existe ruído rigorosamente branco, pois todo PE tem um tempo de correlação finito. Existem, entretanto, circunstâncias em que o tempo de correlação de um PE é tão curto, na escala de tempo de interesse, que tratá-lo como ruído branco é uma boa aproximação. Consideremos, por exemplo, uma partícula coloidal num líquido. Esta partícula executará um movimento aleatório, chamado “movimento Browniano”, em consequência dos incessantes choques com as moléculas do líquido. Pelo tamanho da partícula, muito maior que o tamanho das moléculas do líquido, muitos choques ocorrem simultaneamente (ou quase), aplicando na partícula um impulso cuja intensidade e direção variam em tempos menores que nanosegundos, ou seja, muitas ordens de grandeza menores que as escalas de tempo de observação do movimento da partícula.

Relacionado ao ruído branco $\xi(t)$ e de grande importância em cálculo estocástico, é o “Processo de Wiener” $W(t)$, que pode ser definido por

$$W(t) = \int_0^t \xi(t') dt'. \quad (7.1)$$

Algumas propriedades importantes de $W(t)$ seguem facilmente de sua definição:

- 1) $\langle W(t) \rangle = 0$, que segue imediatamente de $\langle \xi(t) \rangle = 0$;
- 2) $\langle W(t)W(t') \rangle = \min(t, t')$, que se demonstra facilmente, usando a definição de $W(t)$ e a função de correlação de $\xi(t)$; deixamos a demonstração como exercício;
- 3) $W(t)$ é um PE Markoviano, Gaussiano, pois, sendo a integral (i.e., o limite de uma soma) de um PE cujo tempo de correlação é zero, sua Markovianidade é imediata e sua Gaussianidade segue do teorema central de limite; tendo média zero e variância $\sigma^2 = t$, como segue da propriedade 2 acima, sua distribuição de probabilidade é, portanto,

$$P_W(w, t) = \frac{1}{\sqrt{2\pi t}} \exp\left[-\frac{w^2}{2t}\right]. \quad (7.2)$$

4) A probabilidade condicional (ou probabilidade de transição)

$$T(w, t | w_0, t_0) = \frac{1}{\sqrt{2\pi(t-t_0)}} \exp\left[-\frac{(w-w_0)^2}{2(t-t_0)}\right] \quad (7.3)$$

segue da definição e da propriedade 3. Com efeito, dado que $W(t_0) = w_0$, pode-se escrever

$$W(t) = w_0 + \int_{t_0}^t \xi(t') dt',$$

cuja distribuição é a Gaussiana acima.

A largura da distribuição para o *incremento de Wiener*, $\Delta W = W(t + \Delta t) - W(t)$, é portanto

$$\sigma_{\Delta W} = \sqrt{\Delta t}. \quad (7.4)$$

Este fato confere ao processo de Wiener características muito especiais. Por exemplo, $W(t)$ não possui derivada, como se pode ver tentando definir a derivada de W na forma usual:

$$\frac{dW}{dt} = \lim_{\Delta t \rightarrow 0} \frac{\Delta W(\Delta t)}{\Delta t} \sim \frac{\sqrt{\Delta t}}{\Delta t} \rightarrow \infty$$

Esta constatação parece contraditória com o fato de termos definido o processo de Wiener como a integral de $\xi(t)$. Esperaríamos que a derivada de $W(t)$ fosse $\xi(t)$. Entretanto, a propriedade do cálculo usual de que a derivada da integral de uma função f é a própria f só é válida para funções contínuas e deriváveis, o que não é o caso do ruído branco.

Como o ruído branco não é um PE matematicamente bem definido, os matemáticos preferem definir o processo de Wiener pelas suas propriedades, listadas acima. Nós usamos como definição a Eq.(7.1) pelo seu significado físico, embora reconhecendo a falta de rigor matemático.

Não é difícil verificar que a probabilidade de transição $T(w, t | w_0, t_0)$, dada pela Eq.(7.3), é solução da equação de difusão em w , com coeficiente de difusão $D = 1$,

$$\frac{\partial T(w, t | w_0, t_0)}{\partial t} = \frac{1}{2} \frac{\partial^2 T(w, t | w_0, t_0)}{\partial w^2}, \quad (7.5)$$

com condição inicial

$$T(w, t_0 | w_0, t_0) = \delta(w - w_0), \quad (7.6)$$

ou seja, a matéria que se difunde está, no instante inicial t_0 , toda concentrada na posição inicial w_0 .

7.2 Derivada de PE e Equação de Langevin

Como um processo estocástico, $X(t)$, não é uma função, no sentido usual, temos que dizer o que entendemos pela derivada,

$$\dot{X}(t) \equiv \frac{dX(t)}{dt} : \quad (7.7)$$

“ $\dot{X}(t)$ significa o *processo estocástico* cujas realizações são as derivadas das realizações do PE $X(t)$ ”.

Antes de apresentarmos a forma geral das equações diferenciais estocásticas de interesse da Física, consideraremos, como exemplo, a equação de movimento de uma partícula Browniana de massa m e posição $\mathbf{R}(t)$ e que se move em consequência das colisões moleculares, cuja força modelamos pelo ruído branco $\boldsymbol{\xi}(t)$, com intensidade α , além de uma força dissipativa (atrito) linear na velocidade. Admitindo, ainda, uma força $\mathbf{F}(\mathbf{R})$, derivada de potencial, a equação de movimento fica

$$m\ddot{\mathbf{R}} = \mathbf{F}(\mathbf{R}) - \gamma \dot{\mathbf{R}} + \alpha \boldsymbol{\xi}(t), \quad (7.8)$$

conhecida como *Equação de Langevin*. Nossa primeira observação é que uma equação diferencial de segunda ordem, como a Eq.(7.8)

(neste caso, três equações, correspondentes às coordenadas cartesianas), pode ser escrita como um sistema de equações de primeira ordem:

$$\begin{aligned}\dot{\mathbf{R}} &= \mathbf{V} \\ \dot{\mathbf{V}} &= \frac{1}{m} \mathbf{F}(\mathbf{R}) - \frac{\gamma}{m} \mathbf{V} + \frac{\alpha}{m} \boldsymbol{\xi}(t).\end{aligned}\quad (7.9)$$

Para simplificar a notação, introduzimos o PE $X(t)$ representando o conjunto (\mathbf{R}, \mathbf{V}) , i.e., $X = (R_x, R_y, R_z, V_x, V_y, V_z)$, e, para incluir situações mais gerais, em que a intensidade do ruído, assim como as demais forças, possam depender de $X(t)$ e de t , escrevemos a equação estocástica na forma

$$\dot{X}(t) = A(X(t), t) + B(X(t), t)\boldsymbol{\xi}(t), \quad (7.10)$$

onde X , A e $\boldsymbol{\xi}$ devem ser entendidos como vetores coluna e B como matriz.

Como exemplo, vejamos o caso em que $\mathbf{F}(\mathbf{R})$ é a força de um oscilador harmônico isotrópico $\mathbf{F} = -k\mathbf{R}$ cuja massa é $m = 1$. Neste caso o sistema Eq.(7.9) se desacopla em três sistemas independentes, um para cada coordenada cartesiana. Para cada sistema os vetores coluna X , $\boldsymbol{\xi}$ e A , e a matriz B são

$$X = \begin{pmatrix} R_i(t) \\ V_i(t) \end{pmatrix} \quad (7.11)$$

$$\boldsymbol{\xi} = \begin{pmatrix} 0 \\ \xi_i(t) \end{pmatrix} \quad (7.12)$$

$$A = \begin{pmatrix} V_i(t) \\ -k R_i(t) - \gamma V_i(t) \end{pmatrix} \quad (7.13)$$

$$B = \begin{pmatrix} 0 & 0 \\ 0 & \alpha \end{pmatrix} \quad (7.14)$$

Voltemos à forma geral, Eq.(7.10). A matriz B pode ser constante, como no exemplo acima, ou ser função de X , como nos exemplos que veremos na seção 7.4. Quando B =constante a Eq.(7.10) se diz de ruído **aditivo** e quando $B = B(X(t))$ ela é dita de ruído **multiplicativo**. Esta diferenciação é muito importante, como veremos

$$(7.18) \quad dX(t) = A(X(t)) dt + B dW(t).$$

Logo, podemos escrever

A(X) por $A(X(t))$ na Eq.(7.17) e desprezível para dt infinitesimal.

ordem em dt seja no máximo dt , o erro que cometemos por substituir A(X) valor que depende de dt . Como só interessa manter termos cuja No limite $\Delta t \rightarrow dt$ o valor médio A(X) difference de $A(X(t))$ por alguma no último termo, usamos a definição do processo de Wiener, Eq.(7.1). onde A(X) é o valor médio de $A(X(t))$ no intervalo considerado e,

$$(7.17) \quad \equiv A(\underline{X}) \Delta t + B \Delta W(t),$$

$$= A(\underline{X}) \Delta t + B (W(t + \Delta t) - W(t))$$

$$(t)X - (t\Delta t + t)X \equiv (t)X \Delta t$$

ou seja,

$$(7.16) \quad \int_{t+\Delta t}^t A(X(t')) dt' + B \int_{t+\Delta t}^t = (t)X \Delta t$$

No intuito de transformar a Eq.(7.15) para a forma diferencial (i.e., infinitesimal):

vamos integrá-la de t a $t + \Delta t$ e após somarmos o limite $\Delta t \rightarrow dt$

$$(7.15) \quad X(t) = A(X(t), t) + B\xi(t),$$

Nesta segredo vamos considerar apenas equações de ruído aditivo,

7.3 Equações Diferenciais Estocásticas (EDE) com Ruído Aditivo

na seção 7.4. Uma eventual dependência explícita de B em t não é relevante para esta diferenciação.

Portanto, a transformação da Eq.(7.15) na Eq.(7.18) é equivalente a multiplicar a primeira por dt e substituir $\xi(t) dt$ por $dW(t)$, um procedimento que usaremos várias vezes. Para resolver numericamente esta equação temos que aprender a tratar o termo $B dW(t)$, que não estava presente quando estudamos os algoritmos de integração de equações diferenciais ordinárias, no capítulo 2 deste livro. Temos que lembrar que o “incremento de Wiener”, $dW(t)$, é um PE Gaussiano, de largura $\sigma = \sqrt{dt}$. Por isso, a cada passo de integração temos que sortear $dW(t)$ e normalizar o resultado apropriadamente. Vamos chamar de R_G um número aleatório, com distribuição Gaussiana, centrada em $R_G = 0$ e de largura 1. Em SILAB $R_G = rand(1, 1, 'normal')$. Com esta convenção o último termo da Eq.(7.18) pode ser escrito como

$$B dW(t) = \sqrt{dt} B R_G .$$

No caso de m ruídos independentes, $dW(t)$ será um vetor coluna de m componentes e devemos substituir $rand(1, 1, 'normal')$ por $rand(m, 1, 'normal')$. Por simplicidade, vamos tratar do caso unidimensional. O primeiro termo da Eq.(7.18) pode ser tratado por qualquer um dos algoritmos vistos no capítulo 2. Entretanto, como o termo em $dW(t)$, por ser $O(\sqrt{dt})$, é, geralmente, dominante, os pequenos erros de integração cometidos no termo em dt não alteram significativamente a solução. Por isso o algoritmo mais usado na integração do termo $A(X(t)) dt$ é, simplesmente, o de Euler. Nalgumas circunstâncias pode ser conveniente usar um procedimento conhecido por “algoritmo de Heun”, que consiste em substituir $A(X(t))$ por $A(X(t) + 0.5 dX)$, para o que deve ser possível explicitar, por procedimento algébrico, dX na Eq.(7.18).

Restringindo-nos ao tratamento por Euler, a discretização da Eq.(7.18) é feita da seguinte forma:

Dividimos o intervalo de integração $[0, t_{max}]$ em n intervalos de tamanho Δt . Criamos o “vetor” $t = (t_1, \dots, t_{n+1}) = (0, \Delta t, \dots, t_{max})$. Por eficiência do procedimento, sorteamos de uma vez todos os “incrementos de Wiener”, criando o vetor ΔW . Em SCILAB escrevemos:

$$\Delta W = sqrt(\Delta t) * rand(1, n, 'normal')$$

O j^{mo} passo de integração será, então,

$$x(j+1) = x(j) + A(x(j)) * \Delta t + B * \Delta W(j)$$

Exemplo 7.1: Poço Duplo com Ruído Aditivo

Como primeiro exemplo vamos resolver a seguinte equação:

$$dX = (\alpha X - X^3)dt + \beta dW(t). \quad (7.19)$$

Para interpretar esta equação, consideremos uma partícula em um poço de potencial com dois mínimos, em presença de ruído aditivo (veja figura 7.1):

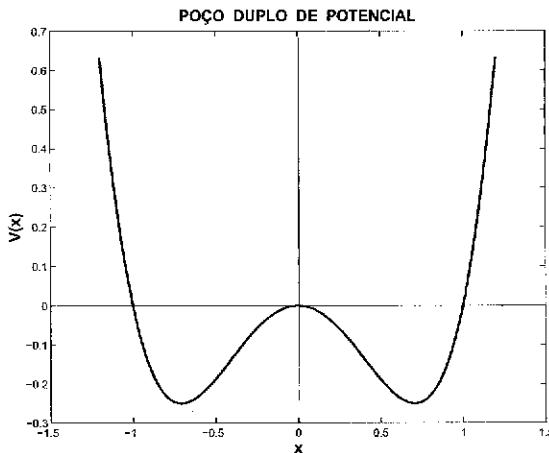


Figura 7.1: Potencial tipo poço duplo

O potencial da figura, chamado de “poço duplo”, tem a seguinte forma analítica:

$$V(x) = a x^4 - b x^2.$$

A força devida a este potencial é

$$F = -\frac{dV}{dx} = -4a x^3 + 2b x. \quad (7.20)$$

Considerando ainda uma força dissipativa $-\gamma \dot{x}$ e uma força de ruído $\beta \xi(t)$, a equação de movimento fica

$$m \ddot{X} = -\gamma \dot{X} - 4a X^3 + 2b X + \beta \xi(t). \quad (7.21)$$

No limite em que o termo de inércia é desprezível ($m \simeq 0$) e escolhendo $\gamma = 1$, $4a = 1$ e $2b = \alpha$ obtemos, pelo procedimento descrito acima (que implica em substituir ξdt por $dW(t)$), a Eq.(7.19).

Programa: poco_duplo.m

```
// poco_duplo.sce:
// Resolve a equação de movimento num poço duplo, com
// ruído aditivo, dx= (alfa x - x^3)dt + beta dw(t),
// entre t=0 e tmax;
// Exemplo: [tmax=100,dt=0.01,alfa=2,beta=1, x(0)=1]

par=input('deh [tmax, dt, alfa, beta, x(0)]-\n');
tmax=par(1); dt=par(2); alfa=par(3);
beta=par(4); x0=par(5);
raizdt=sqrt(dt)*beta;
num=floor(tmax/dt);
sig=1+alfa*dt;
x=zeros(1,num);
dW=raizdt*rand(1,num,'normal');
x(1)=x0;
for j=1:num-1;
    xj=x(j);
    x(j+1)=xj*(sig-xj*xj*dt)+dw(j);
end
t=[0:num-1]*dt;
plot(t,x)
```

Exercício 7.1:

- a) Execute o programa “poco_duplo” diversas vezes, com diferentes valores dos parâmetros α e β , de maneira que a partícula, para alguns valores, oscila durante algum tempo em torno de um dos mínimos do potencial e, eventualmente, transita para o outro mínimo e, para outros valores dos parâmetros, não transita entre os mínimos; observe ainda que com diferentes valores de dt (por exemplo, 0.002, 0.01, 0.04) as realizações são qualitativamente iguais (lembre que com os dt menores os gráficos são naturalmente mais “cheios” por

que possuem mais pontos;

- b) Complemente o programa “poco_duplo” para executar N realizações do PE e obter a média $\bar{x}(t) = \langle X(t) \rangle$ e a variância $\sigma^2(t) = \langle (X(t) - \bar{x}(t))^2 \rangle$; interprete os resultados;
- c) Modifique o programa “poco_duplo”, substituindo, na geração de dW , $\text{sqrt}(dt)$ por dt , como se o ruído branco fosse um processo contínuo e derivável, e execute o programa com diversos valores de dt ; verifique que os resultados são qualitativamente diferentes, o que evidencia que esta forma de resolver a equação de Langevin está errada.

7.4 EDE com Ruído Multiplicativo

Vamos agora examinar uma forma mais geral de equações diferenciais estocásticas. Voltemos a escrever um sistema de equações em termos das derivadas e do ruído branco, como na Eq.(7.10), permitindo, entretanto, que tanto A como B dependam de X e possam também ter dependência explícita em t :

$$\frac{dX}{dt} = A(X, t) + B(X, t)\xi(t) . \quad (7.22)$$

Como vimos na seção 7.2, quando B depende de X a Eq.(7.22) é denominada equação diferencial estocástica com ruído multiplicativo.

Vamos procurar transformar a Eq.(7.22) para a forma diferencial, como fizemos no caso de ruído aditivo, integrando-a entre t e $t + \Delta t$:

$$\Delta X(t) = \int_t^{t+\Delta t} A(X(t'), t') dt' + \int_t^{t+\Delta t} B(X(t'), t') \xi(t') dt' . \quad (7.23)$$

A primeira integral na Eq.(7.23) pode ser tratada da mesma forma como tratamos o termo análogo da Eq.(7.10), ou seja, podemos substituí-lo, no limite $\Delta t \rightarrow dt$ por $A(X(t), t)dt$. Já, no caso da segunda integral, o procedimento não é tão simples. Temos que lembrar que, conforme Eq.(7.4), $\Delta W(t) \sim \sqrt{\Delta t}$. Então ΔX também

pode conter termo $O(\sqrt{\Delta t})$ e o erro que se comete substituindo a integral por $B(X(t), t) \Delta W(t)$ pode ser igual ou maior do que $O(\Delta t)$. Por isso as equações estocásticas com ruído multiplicativo devem ser tratadas por um cálculo estocástico especial, que pode ser o “cálculo de Ito” ou o “cálculo de Stratonovich”, como passamos a apresentar.

7.4.1 Cálculo de Ito

Uma apresentação detalhada dos cálculos de Ito e de Stratonovich neste livro fugiria ao seu propósito básico que é o de apresentar as ferramentas operacionais para o cálculo numérico em problemas de Física. Alguns conceitos, entretanto, são necessários para o bom entendimento dessas ferramentas. Demonstrações longas não serão apresentadas, fornecendo-se, ao invés, as referências.

Para entendermos o conceito de integral estocástica é necessário aprender primeiro o conceito de *convergência em média quadrática*. Seja X_1, X_2, \dots, X_n uma sequência de variáveis aleatórias. Se dissermos que esta sequência converge, quando n tende a infinito, para a variável aleatória X , não estamos sendo claros. Como X_n e X são variáveis aleatórias, temos que dizer o que significa X_n convergir para X . Há, na teoria de variáveis aleatórias, muitas definições alternativas de convergência, como se pode encontrar, por exemplo, no livro de Gardiner[10]. Interessa-nos, para o cálculo estocástico, a convergência em média quadrática: Dizemos que X_n converge em média quadrática para X se

$$\lim_{n \rightarrow \infty} \langle (X_n - X)^2 \rangle = 0. \quad (7.24)$$

Dizemos, então, que o *limite em média quadrática* de X_n é X , o que representamos por

$$\text{ms-lim}_{n \rightarrow \infty} X_n = X \quad (7.25)$$

Consideremos, em especial, situações em que as variáveis aleatórias da sequência são funções de um processo estocástico, isto é, $X_n = X_n(t)$ e $X = X(t)$. Neste caso a média indicada na Eq.(7.24) é uma média sobre realizações do PE,

$$\langle (X_n(t) - X(t))^2 \rangle = \frac{1}{N} \sum_{R=1}^N (x_{n,R}(t) - x_R(t))^2 \quad (7.26)$$

onde $x_{n,R}(t)$ é o valor que adquire a função x_n na realização R de X , no instante t . Como cada termo da Eq.(7.26) é positivo ou nulo, para que $\langle (X_n(t) - X(t))^2 \rangle$ seja nulo, realizações para as quais $X_{n,R}(t) \neq X_R(t)$, se houver, devem constituir um subconjunto de “medida de probabilidade nula”, isto é, devem ser tão raras que sua contribuição para a média é nula. É claro que se os X_n forem variáveis não aleatórias, o limite em média quadrática coincide com o limite usual do cálculo.

Seja $G(X(t), t)$ uma função do PE $X(t)$ e, possivelmente, função explícita de t . Por sua vez, $X(t)$ depende de $W(t)$, no sentido de que a cada realização de $W(t)$ corresponde uma realização de $X(t)$. Geralmente, nas situações de interesse, $X(t)$ é solução de uma equação diferencial estocástica que tem $W(t')$ como ruído. O valor de $X(t)$ só é influenciado pelos valores de $W(t')$, com $t' \leq t$, ou seja, $X(t)$ e as variações de $W(t')$ em tempos posteriores a t são variáveis aleatórias independentes. Esta propriedade se expressa em cálculo estocástico pela qualificação “não antecipadora” para a variável $X(t)$. Define-se a **Integral de Ito** de $G(X(t), t)$, de t_0 a t , por

$$\int_{t_0}^t G(X(t'), t') \bullet dW(t') = \text{ms-lim}_{\Delta t \rightarrow 0} \sum_{n=0}^N G(X(t_n), t_n) \Delta W(t_n) , \quad (7.27)$$

onde

$$\begin{aligned} N &= (t - t_0)/\Delta t \\ t_n &= t_0 + n \Delta t \\ \Delta W(t_n) &= W(t_{n+1}) - W(t_n) \end{aligned} \quad (7.28)$$

O símbolo $\bullet dW(t')$ é usado na Eq.(7.27) para indicar que a integral é a de Ito. Na próxima seção veremos a integral de Stratonovich, que tem uma definição diferente da de Ito.

Algumas propriedades notáveis das integrais de Ito devem ser mencionadas:

- 1) As regras de integração são, geralmente, diferentes das regras de integração do cálculo usual; por exemplo,

$$\int_{t_0}^t W(t') \bullet dW(t') = \frac{1}{2}[W(t)^2 - W(t_0)^2 - (t - t_0)] . \quad (7.29)$$

Demonstração desta e de outras integrais podem ser encontradas, por exemplo, no livro de Gardiner[10]. Mais no espírito do presente texto, vamos apresentar uma evidência da correção da Eq.(7.29) pelo exercício 7.2:

- 2) Integrais de potências inteiras de $W(t)$ também têm solução conhecida:

$$\int_{t_0}^t W(t')^n \bullet dW(t') = \frac{1}{n+1} [W(t)^{n+1} - W(t_0)^{n+1}] - \frac{n}{2} \int_{t_0}^t W(t')^{n-1} dt' , \quad (7.30)$$

onde a última integral, à direita, é uma usual integral de Riemann. O leitor pode modificar o programa feito no exercício 7.2, que é o caso particular desta integral, com $n = 1$, para verificar a correção da Eq.(7.30) para outros valores de n .

- 3) A média sobre realizações de qualquer integral de Ito é nula,

$$\left\langle \int_{t_0}^t G(X(t'), t') \bullet dW(t') \right\rangle = 0 . \quad (7.31)$$

Com efeito, nas Eqs.(7.27) e (7.28) está claro que $X(t_n)$ só depende dos valores assumidos por W entre t_0 e t_n enquanto que $\Delta W(t_n)$ é a variação de W no intervalo posterior a t_n , e portanto, independente da $X(t_n)$; logo, a média do produto é o produto das médias e como $\langle \Delta W(t_n) \rangle = 0$ segue a Eq.(7.31).

4) A integral de Ito também pode ser definida com qualquer potência de dW como medida de integração,

$$\int_{t_0}^t G(X(t'), t') \bullet (dW(t'))^m \equiv \text{ms-lim}_{\Delta t \rightarrow 0} \sum_{n=1}^N G(X(t_n), t_n) (\Delta W(t_n))^m . \quad (7.32)$$

Pode-se demonstrar (veja referências [10] ou [27]) que

$$\int_{t_0}^t G(X(t'), t') \bullet (dW(t'))^2 = \int_{t_0}^t G(X(t'), t') dt' \quad (7.33)$$

e

$$\int_{t_0}^t G(X(t'), t') \bullet (dW(t'))^m = 0 \text{ para } m > 2 . \quad (7.34)$$

Além disso,

$$\int_{t_0}^t G(X(t'), t') \bullet dW(t') dt = 0 . \quad (7.35)$$

Como $(dW(t))^m$ só é usada em integrais, pode-se escrever, por simplicidade, as seguintes *relações diferenciais de Ito*:

$$\begin{aligned} (dW(t))^2 &= dt \\ dW(t)dt &= 0 \\ (dW(t))^m &= 0, \text{ para } m > 2 . \end{aligned} \quad (7.36)$$

Estas relações serão muito úteis a seguir.

Sobre Integração Numérica das Integrais Estocásticas

A integral de Ito, assim como a de Stratonovich, que veremos adiante, são definidas como um "limite em média quadrática". Entretanto, a integração numérica usa a discretização da variável independente (tempo). Neste caso o mencionado "limite" significa apenas "suficientemente pequeno", como no caso da integração ordinária, realizada nos capítulos anteriores, em especial no Cap.1, seção 1.3. Para sermos rigorosos, deveríamos realizar a integração com diversos valores de Δt , cada vez menores, até evidenciar-se um limite para $\Delta t \rightarrow 0$.

Para comparar os resultados, obtidos com diferentes Δt , temos que tomar o cuidado de integrar a mesma realização do ruído. Os valores dos ΔW , correspondentes a um Δt , que é igual a n vezes um $\Delta t_{pequeno}$, será a soma dos n correspondentes $\Delta W_{pequeno}$. No caso de se calcular variáveis estatísticas (como médias, variâncias, correlações, ...), obtidas de um número muito grande de realizações, este cuidado não precisa ser tomado.

Exemplo 7.2: Teste da Precisão

Para exemplificar o procedimento descrito acima, de integrar a mesma realização do ruído com dois Δt distintos, vamos considerar a Eq.(7.30). Para o caso $n = 2$, tomando $t_0 = 0$ e $W(0)=0$, podemos rearranjar esta equação para a forma

$$\int_0^t (W(t')^2 \bullet dW(t') + W(t')dt') = W(t)^3/3 \quad (7.37)$$

O programa abaixo, "teste_precisao.sce", realiza a integral acima com dtg (grande) e $dtp = dtg/10$ (pequeno), e compara os resultados com o resultado exato, dado pela Eq.(7.37). Este exemplo de programa pode servir de modelo de procedimento para o caso de se testar a precisão de um algoritmo de integração de equações diferenciais estocásticas, que estudaremos a seguir. Para cada par de valores $[tmax, dt]$ convém executar várias vezes o programa porque, como estamos integrando variáveis aleatórias, é possível ocorrerem flutuações que levem a resultado oposto do esperada. Executando várias vezes observaremos a tendência dominante.

Programa: teste_precisao.sce

```
// teste_precisao.sce: integral de Ito
// int (W^2 . dW + W dt)= W^3/3; Com dtg=10*dtp
// testa a qualidade dos resultados para a
// mesma realização do ruído
// Ex: [tmax, dt]=[10, 0.1]

par = input('deh [ tmax, dt] \n');
tmax=par(1); dt=par(2);
dtg=dt; // dt grande
```

```

dtp=dt/10; // dt pequeno
ntg=round(tmax/dtg); //número de intervalos grandes
sqdtp=sqrt(dtp);
Wp=0; // inicializa W para dt pequeno
Wg=0; // inicializa W para dt grande
W=zeros(1,ntg+1);
Intp=zeros(1,ntg+1); // inicia integral com dt pequeno
Intg=zeros(1,ntg+1); // inicia integral com dt grande
Intep=0;
t=linspace(0,tmax,ntg+1);
for j=1:ntg
    dWg=0; // inicia os dWg para este j
    for k=1:10
        dWp=sqdtp*rand(1,1,"normal");
        dWg=dWg+dWp; // soma os dWp
        Intep=Intep+Wp^2*dWp+Wp*dtp; // integra Wp
        Wp=Wp+dWp; // atualiza Wp
    end
    Intp(j+1)=Intep;
    Intg(j+1)=Intg(j)+Wg^2*dWg+Wg*dtg; // integra Wg
    Wg=Wg+dWg; // atualiza Wg
    W(j)=Wg;
end
Intito=(W.^3)/3;
xset('window',1); clf;
plot(t,Intito,t,Intp,t,Intg)
legend('Exata', 'dt pequeno', 'dt grande')

```

Exercício 7.2:

Programar o cálculo numérico da integral de Ito da Eq.(7.29), com $t_0 = 0$, $W(0) = 1$ e plotar o resultado em função de t no intervalo $0 \leq t \leq t_{max}$, juntamente com o segundo membro da Eq.(7.29); escolhendo sucessivos valores para Δt , verificar que no limite $\Delta t \rightarrow 0$ o resultado da integração coincide com a forma dada analiticamente pela Eq.(7.29); para comparar, plotar também o valor da integral correspondente no cálculo usual, $\frac{1}{2}[W(t)^2 - W(0)^2]$.

Diferencial de Ito

Consideremos uma função “bem comportada” de t e de W , $F(t) \equiv F(W(t), t)$ ou seja, F depende de t através de $W(t)$, podendo também depender explicitamente de t . Embora $W(t)$ dependa de t de forma “não diferenciável”, estamos supondo que F depende de W de forma diferenciável (por ex. $F = W^m$). A diferencial de F é obtida por expansão de Taylor até ordem $dt^1 = dt$,

$$\begin{aligned} dF(t) &\equiv F(t + dt) - F(t) = \\ &= \frac{\partial F}{\partial W} dW(t) + \frac{\partial F}{\partial t} dt + \frac{1}{2} \frac{\partial^2 F}{\partial W^2} (dW(t))^2 + \dots, \end{aligned} \quad (7.38)$$

ou seja,

$$dF(t) = \frac{\partial F}{\partial W} \bullet dW(t) + \left(\frac{\partial F}{\partial t} + \frac{1}{2} \frac{\partial^2 F}{\partial W^2} \right) dt, \quad (7.39)$$

onde usamos as Eqs.(7.36).

Equação Diferencial Estocástica de Ito

Voltemos a considerar a equação de Langevin na forma integrada, Eq.(7.23),

$$\begin{aligned} \Delta X(t) &\equiv X(t + \Delta t) - X(t) = \\ &= \int_t^{t+\Delta t} A(X(t'), t') dt' + \int_t^{t+\Delta t} B(X(t'), t') \xi(t') dt'. \end{aligned} \quad (7.40)$$

Já vimos que, no limite $\Delta t \rightarrow dt$, $A(X(t'), t')$ pode ser substituído por $A(X(t), t)$ no integrando da primeira integral acima. O mesmo procedimento não pode ser empregado na segunda integral, por causa do fator $\xi(t')$ que torna qualquer ponto do integrando descorrelacionado de qualquer outro ponto.

Para esclarecer melhor este ponto vamos supor que em vez de substituirmos $B(X(t'), t')$ por $B(X(t), t)$ fazemos a substituição

$$B(X(t'), t') \Rightarrow B\left(\frac{X(t) + X(t + \Delta t)}{2}, t\right), \quad (7.41)$$

ou seja, tomamos como argumento de B a média entre os valores de X no início e no fim do intervalo de integração, o que certamente corresponde a uma aproximação melhor para a integral. Neste caso

$$\Delta X(t) = A(X(t), t)\Delta t + B\left(X(t) + \frac{1}{2}\Delta X(t), t\right)\Delta W(t), \quad (7.42)$$

pois

$$X(t + \Delta t) = X(t) + \Delta X(t)$$

e

$$\int_t^{t+\Delta t} \xi(t')dt' = \Delta W(t).$$

Iterando a Eq.(7.42) em ΔX , isto é, substituindo o ΔX que aparece no argumento de B pelo valor de ΔX dado pela própria Eq.(7.42), e expandindo B em série de Taylor segue

$$\begin{aligned} \Delta X(t) &= A(X(t), t)\Delta t + B(X(t), t)\Delta W(t) + \\ &+ \frac{1}{2} \frac{\partial B}{\partial X} [A(X(t), t)\Delta t + B(X(t), t)\Delta W(t) + \dots] \Delta W(t) + \dots \end{aligned}$$

Tomando agora o limite infinitesimal, $\Delta t \rightarrow dt$, $\Delta X \rightarrow dX$, $\Delta W \rightarrow dW$ e desprezando termos de ordem maior que dt^1 , vem

$$\begin{aligned} dX(t) &= A(X(t), t)dt + B(X(t), t) \bullet dW(t) + \\ &+ \frac{1}{2} \frac{\partial B}{\partial X} B(X(t), t)(dW(t))^2, \end{aligned} \quad (7.43)$$

ou seja

$$dX(t) = A^{(I)}(X(t), t)dt + B(X(t), t) \bullet dW(t), \quad (7.44)$$

onde definimos

$$A^{(I)}(X(t), t) = A(X(t), t) + \frac{1}{2} \frac{\partial B}{\partial X} B(X(t), t) \quad (7.45)$$

e usamos a Eq.(7.36). A Eq.(7.44) se chama “Equação Diferencial Estocástica de Ito (EDE-Ito)”. Ela corresponde à forma usual de se

substituir a equação de Langevin com ruído branco, Eq.(7.22), não muito bem definida matematicamente, por uma equação diferencial bem definida, em cálculo de Ito. Por isso ela é também conhecida por “Equação de Ito-Langevin”.

Exemplo 7.3: Vetor Girante por Ito

Consideremos uma grandeza vetorial, μ , em duas dimensões, com módulo constante e direção que varia no tempo, sob a ação de um torque com componente estocástica. Um exemplo pode ser o momento magnético de uma partícula superparamagnética com alta anisotropia uniaxial negativa, de maneira que μ fique confinado ao plano $x \times y$, em presença de um campo magnético H_0 paralelo ao eixo de simetria (eixo z). Pondo, por simplicidade, $|\mu| = 1$, podemos escrever

$$\mu = (X(t), Y(t)) , \quad (7.46)$$

com

$$\begin{aligned} X(t) &= \cos \Phi(t) \\ Y(t) &= \sin \Phi(t) \end{aligned} \quad (7.47)$$

onde a fase $\Phi(t)$ obedece a seguinte equação diferencial estocástica de ruído aditivo:

$$d\Phi(t) = w_0 dt + \beta dW(t) . \quad (7.48)$$

No caso de μ ser o momento magnético, mencionado acima, a Eq.(7.48) pode ser interpretada como a rotação causada por um torque sistemático devido ao campo H_0 , que resulta na frequência angular $w_0 = \gamma H_0$, sendo γ o fator giromagnético, e um torque estocástico tipo ruído branco, de intensidade β , causado pelas vibrações térmicas da rede cristalina (fônons). A Eq.(7.48), por ser de ruído aditivo, pode ser integrada trivialmente. Escolhendo como condição inicial $\Phi(0) = 0$ segue

$$\Phi(t) = w_0 t + \beta W(t) , \quad (7.49)$$

e, consequentemente,

$$\begin{aligned} X(t) &= \cos(w_0 t + \beta W(t)) \\ Y(t) &= \sin(w_0 t + \beta W(t)) . \end{aligned} \quad (7.50)$$

Usando a regra de diferenciação de Ito, Eq.(7.39), no par de variáveis (X, Y) dado pela Eq.(7.50) obtém-se o seguinte sistema de equações de Ito-Langevin:

$$\begin{aligned} dX(t) &= \left(-w_0 Y(t) - \frac{\beta^2}{2} X(t) \right) dt - \beta Y(t) \bullet dW(t) \\ dY(t) &= \left(w_0 X(t) - \frac{\beta^2}{2} Y(t) \right) dt + \beta X(t) \bullet dW(t) . \end{aligned} \quad (7.51)$$

É claro que a solução deste sistema, com condições iniciais $X(0) = 1$ e $(Y(0) = 0)$, é dada pela Eq.(7.50). Por conhecermos sua solução exata, a Eq.(7.51) é um bom exemplo para examinarmos o procedimento de integração numérica, que é baseado na própria definição da integral de Ito, Eq.(7.27). O que se faz é encontrar as realizações do processo estocástico (X, Y) , correspondentes às realizações sorteadas para o processo $W(t)$, efetuando o somatório indicado na Eq.(7.27) para um Δt arbitrário e, diminuindo sucessivamente o valor de Δt , procurar convergência para a solução exata.

Evidentemente, este procedimento de integração numérica não teria nenhum valor se só pudesse ser usado quando se conhece a solução exata. Nas situações práticas de estudo de um fenômeno o que precisamos é calcular médias, variâncias e correlações. Para isso simulamos um número grande de realizações, para as quais calculamos as referidas médias. A convergência para o valor exato é obtida quando a diminuição do valor de Δt não altera significativamente os valores médios calculados, ou então, por extração destes valores para $\Delta t = 0$.

O programa “vetorito.sce”, apresentado a seguir, resolve o par de Eqs.(7.51) com as condições iniciais mencionadas acima, deixando t_{max} , Δt , w_0 e β como parâmetros de entrada.

Programa: vetor_ito.sce

```

// vetor_ito.sce:
// resolve a eq. de Langevin do vetor girante, de
// módulo 1 por Ito, (xi,yi), e compara com a
// solução exata:
// x=cos(wo t + beta*W(t), y=sen(wo t + beta*W(t)
// Exemplo: tmax=10, dt=0.01, w0=1, beta=1

par=input('dê parâmetros [tmax, dt, wo, beta] \n');
tmax=par(1); dt=par(2);
wo=par(3); beta=par(4);
srdt=sqrt(dt);
b2=.5*beta^2;
wodt=w0*dt;
nu=floor(tmax/dt);
num=nu+1;
xi=zeros(1,num);
yi=zeros(1,num);
xe=zeros(1,num);
ye=zeros(1,num);
dW=srdt*rand(1,nu,'normal');
xe(1)=1; xi(1)=1;
W=0;
for j=1:nu;
    W=W+dW(j);
    fase=wodt*j+beta*W;

    // solução por Ito
    xij=xi(j);
    yij=yi(j);
    xi(j+1)=xij-(wo*yij+b2*xij)*dt-yij*beta*dW(j);
    yi(j+1)=yij+(wo*xij-b2*yij)*dt+xij*beta*dW(j);

    // solução exata
    xe(j+1)=cos(fase);
    ye(j+1)=sin(fase);
end

```

```
t=[0:nu]*dt;
xset('window',1); clf
plot(t,xi,t,xe,'r')
legend('x_Ito','x_exata')
xset('window',2); clf
plot(xi,yi,xe,ye,'r')
legend('y versus x por Ito', 'y versus x exata')
```

Melhoria pelo algoritmo de Heun

Como mencionamos na seção 7.3, podemos melhorar o algoritmo de integração, usando para o termo em dt o algoritmo de Heun, isto é, em vez de $A(X(t))$, usar $A(X(t) + \frac{1}{2}\Delta X)$. Vamos deixar a aplicação deste procedimento ao exemplo do vetor girante, como exercício.

Exercício 7.3:

Modifique o programa "vetor_ito.sce", introduzindo o algoritmo de Heun e compare os resultados obtidos com o algoritmo assim modificado com os obtidos sem a modificação. Execute o programa com diferentes parâmetros e verifique que a melhoria é mais significativa quando o ruído não é muito intenso, ou seja, quando o termo determinístico é dominante.

7.4.2 Cálculo de Stratonovich

A principal diferença entre os cálculos de Ito e de Stratonovich decorre das diferentes definições de integral estocástica. Já vimos a definição da integral de Ito, Eq.(7.27). A integral de Stratonovich de uma função $G(X(t'), t')$ é definida por

$$\begin{aligned} \int_{t_0}^t G(X(t'), t') dW(t') &= \\ &= \text{ms-lim}_{\Delta t \rightarrow 0} \sum_{n=0}^N G\left(\frac{X(t_n) + X(t_{n+1})}{2}, t_n\right) \Delta W(t_n). \end{aligned} \quad (7.52)$$

Note que o argumento de B é a média aritmética entre os valores de X no início e no fim do intervalo $[t_n, t_{n+1}]$.

Quando $X(t)$ pode ser escrito explicitamente como função de $W(t)$ as regras de integração são as mesmas que as das integrais de Riemann (cálculo usual). Por isso não usamos nenhum símbolo especial para indicar que a integral é de Stratonovich, diferentemente da integral de Ito, que indicamos com o símbolo \bullet antes do dW . Por exemplo:

$$\int_{t_0}^t W(t') dW(t') = \frac{1}{2}[W(t)^2 - W(t_0)^2]. \quad (7.53)$$

Demonstração desta e doutras integrais podem ser encontradas, por exemplo, no livro de Gardiner[10]. Mais no espírito do presente texto, vamos apresentar uma evidência da correção da Eq.(7.53) pelo exercício 7.4:

Exercício 7.4:

Programar o cálculo numérico da integral de Stratonovich da Eq.(7.53), com $t_0 = 0$, $W(0) = 1$ e plotar o resultado em função de t no intervalo $0 \leq t \leq t_f$, juntamente com o segundo membro da Eq.(7.53); escolhendo sucessivos valores para Δt , verificar que no limite $\Delta t \rightarrow 0$ o resultado da integração coincide com a solução analítica, dada pela Eq.(7.53). Note que a convergência para o resultado analítico quando $\Delta t \rightarrow 0$ é consideravelmente mais rápido que no caso do cálculo de Ito, visto no exercício 7.2, ou seja, Δt não precisa ser tão pequeno para se obter precisão comparável.

Exercício 7.5:

Programar o cálculo numérico da integral de Stratonovich da equação

$$\int_{t_0}^t W^n(t') dW(t') = \frac{1}{n+1}[W(t)^{n+1} - W(t_0)^{n+1}]. \quad (7.54)$$

e comparar o resultado numérico com o analítico. Note que novamente o resultado analítico é equivalente ao de uma integral de Riemann.

Esta semelhança formal entre o cálculo de Stratonovich e o cálculo elementar se mantém na diferenciação, transformação de variáveis, etc. Na verdade o cálculo de Stratonovich foi construído [28] como

o limite branco (i.e., limite para tempo de correlação indo a zero) de processo estocástico com ruído colorido (i.e., ruído com tempo de correlação finito). Por isso quando os físicos escrevem uma equação de Langevin para um sistema da Natureza, o ruído branco é usado neste sentido, ou seja, como o tempo de correlação do ruído é muito pequeno na escala de tempo característica do fenômeno que se quer descrever, ele é modelado por ruído branco. Baseado nisto se vê que, ressalvadas as exceções, uma equação de Langevin da Física deve ser interpretada no sentido de Stratonovich. Devemos enfatizar, entretanto, que o cálculo de Ito tem seus próprios méritos. Em primeiro lugar, ele está assentado sobre bases matemáticas mais sólidas e transparentes que o cálculo de Stratonovich. Por isso várias das propriedades do cálculo de Stratonovich são obtidas a partir do cálculo de Ito, fazendo-se as modificações apropriadas. Por Exemplo, na seção 7.6 introduziremos a importante Equação de Fokker-Planck usando cálculo de Ito. Também, como veremos adiante, a programação numérica das soluções de equações estocásticas é muito mais simples com cálculo de Ito, motivo pelo qual a ampla maioria dos pesquisadores utiliza-o em vez do de Stratonovich nos casos de ruído multiplicativo. Na seção 7.5.1 voltaremos a este assunto em mais detalhe.

Equação Diferencial Estocástica de Stratonovich

Voltemos à Eq.(7.42). No limite $\Delta t \rightarrow 0$, ou seja, fazendo as substituições $\Delta t \rightarrow dt$, $\Delta X \rightarrow dX$ e $\Delta W \rightarrow dW$, já se tem uma equação diferencial estocástica de Stratonovich. Entretanto, não é usual escrever uma equação diferencial tendo diferenciais como argumentos de funções. Por isso usa-se o procedimento equivalente de escrever a equação diferencial simplesmente na forma

$$dX(t) = A(X(t), t)dt + B(X(t), t)dW(t), \quad (7.55)$$

ficando entendido que a integração desta equação deve ser feita segundo a integral de Stratonovich. As equações Eq.(7.44) e Eq.(7.55) são equivalentes, no sentido de que a primeira integrada por Ito e a segunda por Stratonovich devem resultar na mesma solução, ou seja, gerar as mesmas realizações $X_R(t)$ para as mesmas realizações $W_R(t)$.

Semelhantemente, se tivermos uma equação diferencial estocástica de Ito na forma

$$dX(t) = A(X(t), t)dt + B(X(t), t) \bullet dW(t), \quad (7.56)$$

então a seguinte equação de Stratonovich,

$$dX(t) = \left(A(X(t), t) - \frac{1}{2} \frac{\partial B}{\partial X}(X(t), t) \right) dt + B(X(t), t)dW(t), \quad (7.57)$$

Ihe é equivalente.

Exemplo 7.4: Vetor girante por Stratonovich

Consideremos novamente o vetor girante do exemplo 7.3. As correspondentes equações estocásticas de Stratonovich, como o leitor pode verificar facilmente, são:

$$\begin{aligned} dX(t) &= -w_0 Y(t) dt - \beta Y(t) dW(t) \\ dY(t) &= w_0 X(t) dt + \beta X(t) dW(t), \end{aligned} \quad (7.58)$$

ou, em notação de diferenças finitas,

$$\begin{aligned} \Delta X_j &= -w_0 Y_j \Delta t - \beta \left(Y_j + \frac{1}{2} \Delta Y_j \right) \Delta W_j \\ \Delta Y_j &= w_0 X_j \Delta t + \beta \left(X_j + \frac{1}{2} \Delta X_j \right) \Delta W_j. \end{aligned} \quad (7.59)$$

Melhora-se a qualidade da solução numérica substituindo-se, nos termos que multiplicam Δt , X_j por $X_j + \frac{1}{2}\Delta X_j$ e Y_j por $Y_j + \frac{1}{2}\Delta Y_j$, o que equivale a usar o algoritmo de Crank-Nicholson em vez do de Euler em equações diferenciais ordinárias. Em vez das Eqs.(7.59) escrevemos, portanto,

$$\begin{aligned} \Delta X_j &= -w_0 \left(Y_j + \frac{1}{2} \Delta Y_j \right) \Delta t - \beta \left(Y_j + \frac{1}{2} \Delta Y_j \right) \Delta W_j \\ \Delta Y_j &= w_0 \left(X_j + \frac{1}{2} \Delta X_j \right) \Delta t + \beta \left(X_j + \frac{1}{2} \Delta X_j \right) \Delta W_j. \end{aligned} \quad (7.60)$$

Exercício 7.6:

Por procedimento algébrico é fácil explicitar as incógnitas ΔX_j e ΔY_j nas Eqs.(7.60). Feito isto, programe a solução numérica por este algoritmo e compare com as soluções (já programadas anteriormente) de Ito e exata, para diversos valores de Δt . Você verá que a solução numérica por Stratonovich converge muito mais rapidamente à solução exata que a correspondente solução por Ito.

7.5 Cálculo Estocástico com Vários Ruídos

No tratamento apresentado até aqui admitimos que $X(t)$ pode ser um vetor de n componentes. Vamos generalizar mais o tratamento e considerar os casos em que o ruído também tenha várias componentes, digamos m , independentes, ou seja,

$$\xi(t) = [\xi_1(t), \dots, \xi_\alpha(t), \dots, \xi_m(t)] \text{ com}$$

$$\langle \xi_\alpha(t) \xi_\beta(t') \rangle = \delta_{\alpha,\beta} \delta(t - t'). \quad (7.61)$$

Semelhantemente, teremos m processos de Wiener, com

$$\langle W_\alpha(t) W_\beta(t') \rangle = \delta_{\alpha,\beta} \min(t, t') \quad (7.62)$$

e

$$\langle \Delta W_\alpha(t) \Delta W_\beta(t) \rangle = \delta_{\alpha,\beta} \Delta t. \quad (7.63)$$

A primeira das Eqs.(7.36), neste caso, adquire o forma

$$dW_\alpha(t) dW_\beta(t) = \delta_{\alpha,\beta} dt. \quad (7.64)$$

Escrita em componentes, a Eq.(7.10) fica

$$\dot{X}_i(t) = A_i(X(t), t) + \sum_{\alpha=1}^m B_{i,\alpha}(X(t), t) \xi_\alpha(t) \quad i = 1, \dots, m. \quad (7.65)$$

A EDE de Ito correspondente será, então,

$$dX_i(t) = A_i^{(I)}(X(t), t)dt + \sum_{\alpha=1}^m B_{i,\alpha}(X(t), t) \bullet dW_\alpha(t), \quad (7.66)$$

onde

$$A_i^{(I)}(X(t), t) = A_i(X(t), t) + \frac{1}{2} \sum_{j=1}^n \sum_{\alpha=1}^m \frac{\partial B_{i,\alpha}}{\partial X_j} B_{j,\alpha}(X(t), t). \quad (7.67)$$

Exemplo 7.5: Vetor girante com ruídos independentes

Consideremos novamente o exemplo do vetor girante, mas agora com ruídos independentes para as componentes X e Y . As EDE de Stratonovich são

$$\begin{aligned} dX(t) &= -w_0 Y(t) dt - \beta Y(t) dW_a(t) \\ dY(t) &= w_0 X(t) dt + \beta X(t) dW_b(t), \end{aligned} \quad (7.68)$$

com

$$\langle W_a(t)W_b(t') \rangle = 0.$$

Para encontrarmos as correspondentes EDE de Ito precisamos identificar as os $B_{i,\alpha}$:

$$B_{X,a} = -\beta Y$$

$$B_{X,b} = B_{Y,a} = 0$$

$$B_{Y,b} = \beta X.$$

Portanto, como $B_{X,a}$ só depende de Y e $B_{Y,a} = 0$, etc, as somas na Eq.(7.67) são nulas e, portanto, neste exemplo, $A^{(I)} = A$, ou seja, as correspondentes EDE de Ito são

$$\begin{aligned} dX(t) &= -w_0 Y(t) dt - \beta Y(t) \bullet dW_a(t) \\ dY(t) &= w_0 X(t) dt + \beta X(t) \bullet dW_b(t). \end{aligned} \quad (7.69)$$

Exercício 7.7:

Programar a simulação numérica das EDE acima, tanto na versão

de Stratonovich, Eq.(7.68), como na de Ito, Eq.(7.69), executar o programa para vários valores de Δt e verificar que as soluções ficam iguais quando $\Delta t \rightarrow 0$; executando N (por exemplo, 10000) realizações do processo, calcular $\langle X(t) \rangle$, $\langle Y(t) \rangle$ e $\langle R^2(t) \rangle = \langle X^2(t) + Y^2(t) \rangle$; você verá que, diferentemente do caso do vetor girante com ruído único, visto na seção anterior, agora o valor de $\langle R^2(t) \rangle$ aumenta com o tempo.

7.5.1 Sobre as Abordagens de Ito e de Stratonovich em Simulações Numéricas

Se você fez os exercícios sobre vetor girante, da seção 7.4, deve ter notado que na versão de Stratonovich a convergência para o resultado exato, quando $\Delta t \rightarrow 0$ é muito mais rápida que na versão de Ito, isto é, Δt precisa ser muito menor na versão de Ito que na de Stratonovich para se ter o mesmo grau de precisão. Isto representa uma importante vantagem do cálculo de Stratonovich sobre o de Ito para integração numérica. Cabe então a pergunta, por que a ampla maioria dos trabalhos reportados na literatura sobre simulação numérica de equações de Langevin usam a versão de Ito? A resposta é simples: porque é muito mais fácil programar a integração por Ito que por Stratonovich. Nos exemplos sobre o vetor girante nós conseguimos isolar ΔX na equação discretizada de Stratonovich, por procedimento algébrico, e por isso foi simples programar a integração numérica. Em geral, em problemas menos triviais, esse procedimento não é possível. Com efeito, a forma geral, discretizada, de uma EDE de Stratonovich é

$$\Delta X(t) = A \left(X(t) + \frac{\Delta X(t)}{2} \right) \Delta t + B \left(X(t) + \frac{\Delta X(t)}{2} \right) \Delta W(t), \quad (7.70)$$

onde X e A são, em geral, vetores de n componentes, B é uma matriz $n \times m$ e ΔW é um vetor de m componentes.

Não existe, ao que sabemos, algoritmo geral para resolver a Eq.(7.70). Os seguintes procedimentos funcionam bem em certos casos:

- a) Explicitar, algebricamente o vetor ΔX . Isto só funciona em caso especialmente simples.
- b) Uma solução tipo Runge-Kutta 2^a ordem: Calcula-se a primeira aproximação para ΔX usando apenas $X(t)$ nos argumentos de A e B ; a seguir usa-se o ΔX assim calculado nos argumentos de A e B , obtendo-se ΔX em 2^a ordem em ΔW . O resultado deste procedimento tende a ser semelhante ao do cálculo de Ito.
- c) Continua-se o procedimento descrito no item b substituindo ΔX assim obtido novamente nos argumentos de A e B , e assim sucessivamente, até que se obtenha convergência, ou seja, até que o novo ΔX seja tão próximo do anterior como se estabeleceu a priori. O critério de precisão pode ser, por exemplo,

$$\frac{\sum_{i=1}^n (\Delta X_i - \Delta X_i^a)^2}{\sum_{i=1}^n (\Delta X_i)^2} \leq \epsilon, \quad (7.71)$$

onde ΔX_i^a é o ΔX_i anterior e ϵ é escolhido convenientemente. Devido ao caráter estocástico das equações, é geralmente, desnecessário exigir-se grande precisão, sendo suficientes umas poucas iterações. Embora este procedimento iterativo possa parecer completamente geral, há circunstâncias em que a convergência ocorre e outras em que não há convergência. Neste último caso é melhor permanecer na primeira iteração, como descrito no procedimento b.

Exemplo 7.6: vetor girante por Ito e por Stratonovich
 O programa a seguir resolve o vetor girante, para a mesma realização, por Ito, por Stratonovich procedimento a) acima e compara com a solução exata. Executando-o com diferentes valores de dt, pode-se observar que Stratonovich converge mais rapidamente para a solução exata.

Programa: vetor_ito_strat.sce

```
// resolve a eq. de Langevin do vetor girante, de
// módulo 1 por Ito, (xi,yi), e por Stratonovich,
// (xs,ys), e compara com a solução exata:
```

```

// (x,y)=(cos(wo*t + beta*W(t)),sen(wo*t + beta*W(t)) )
// Exemplo: tmax=10, dt=0.01, w0=1, beta=1

par=input('dê parâmetros [tmax, dt, wo, beta] \n');
tmax=par(1);
dt=par(2);
wo=par(3);
beta=par(4);
srdt=sqrt(dt);
b2=.5*beta^2;
wodt=wo*dt;
nu=floor(tmax/dt);
num=nu+1;
xs=zeros(1,num);
ys=zeros(1,num);
xi=zeros(1,num);
yi=zeros(1,num);
xe=zeros(1,num);
ye=zeros(1,num);
dw=srdt*rand(1,nu,'normal');
xe(1)=1; xi(1)=1; xs(1)=1;
w=0;
for j=1:nu;
    w=w+dw(j);
    fase=wodt*j+beta*w;
    // solução por Stratonovich
    C=(wodt+beta*dw(j))/2;
    D=1+C^2;
    x=(xs(j)-C*ys(j))/D;
    y=(ys(j)+C*xs(j))/D;
    xs(j+1)=2*x-xs(j);
    ys(j+1)=2*y-ys(j);

    // solução por Ito
    xij=xi(j);
    yij=yi(j);
    xi(j+1)=xij-(wo*yij+b2*xij)*dt-yij*beta*dw(j);

```

```

yi(j+1)=yij+(wo*xij-b2*yij)*dt+xij*beta*dw(j);

// solução exata
xe(j+1)=cos(fase);
ye(j+1)=sin(fase);
end
t=[0:nu]*dt;
xset('window',1); clf
plot(t,xi,t,xs,t,xe,'-')
legend('xi','xs','xe')
xset('window',2); clf
plot(0,0,'w',xi,yi, xs,ys, xe,ye,'-')
legend('y versus x','Ito','Strat.','exata')

```

Exercício 7.8:

- Resolver o problema do vetor girante com ruído único (Exercício 7.5) pelo procedimento c e contabilizar, em cada passo de integração, quantas iterações são necessárias para se obter uma precisão aceitável; comparar com a solução pelo procedimento sugerido no exercício 7.5;
- Repetir o exercício com o problema do vetor girante com dois ruídos independentes, como no exercício 7.6.

7.5.2 Solução da Equação de Langevin com Ruído multiplicativo até $O(\Delta t^{\frac{5}{2}})$

Consideremos uma Equação de Langevin na forma

$$m \frac{d^2\mathbf{r}}{dt^2} = \mathbf{f}(\mathbf{r}, \mathbf{v}) \quad (7.72)$$

Vamos reescrevê-la na forma de um sistema de equações de primeira ordem:

$$\begin{aligned} \frac{d\mathbf{r}}{dt} &= \mathbf{v} \\ m \frac{d\mathbf{v}}{dt} &= \mathbf{f} = \mathbf{F}(\mathbf{r}) - \gamma \mathbf{v} + \alpha \boldsymbol{\xi}(t) \end{aligned} \quad (7.73)$$

onde a força \mathbf{F} é uma função contínua, derivável, da posição \mathbf{r} , $-\gamma\mathbf{v}$ é a força dissipativa, linear na velocidade \mathbf{v} e $\alpha\xi$ é ruído branco multiplicativo (i.e., $\alpha = \alpha(\mathbf{r})$).

Vamos integrar as Eqs.(7.73) entre t e $t + \Delta t$:

$$\mathbf{r}(t + \Delta t) = \mathbf{r}(t) + \int_t^{t+\Delta t} \mathbf{v}(t') dt' \quad (7.74)$$

$$\begin{aligned} \mathbf{v}(t + \Delta t) &= \mathbf{v}(t) + \frac{1}{m} \left\{ \int_t^{t+\Delta t} [\mathbf{F}(\mathbf{r}(t')) - \gamma\mathbf{v}(t')] dt' + \right. \\ &\quad \left. + \int_t^{t+\Delta t} \alpha(\mathbf{r}(t')) \xi(t') dt' \right\} \end{aligned} \quad (7.75)$$

Para facilitar a evaluação da integral na Eq.(7.74) e da primeira integral da Eq.(7.75), vamos usar a seguinte propriedade: seja $g(t)$ uma função contínua e derivável no intervalo $[t, t + \Delta t]$; então

$$\int_t^{t+\Delta t} g(t') dt' = \frac{g(t) + g(t + \Delta t)}{2} \Delta t + O(\Delta t^3). \quad (7.76)$$

Para provar esta relação partimos de outra relação, semelhante,

$$\int_{t-\Delta t}^{t+\Delta t} g(t') dt' = \frac{g(t - \Delta t) + g(t + \Delta t)}{2} 2\Delta t + O(\Delta t^3),$$

que é obviamente correta devido à anti-simetria dos termos frente à troca $\Delta t \leftrightarrow -\Delta t$. Definindo

$\tau = t - \Delta t$ e $\Delta\tau = 2\Delta t$ a equação acima fica

$$\int_{\tau}^{\tau+\Delta\tau} g(t') dt' = \frac{g(\tau) + g(\tau + \Delta\tau)}{2} \Delta\tau + O(\Delta\tau^3),$$

o que prova a Eq.(7.76).

Vamos desprezar os termos $O(\Delta t^3)$, e introduzir a seguinte notação:

$$\mathbf{r} \equiv \mathbf{r}(t + \Delta t), \quad \mathbf{r}_0 \equiv \mathbf{r}(t)$$

e, equivalentemente, para \mathbf{v} e \mathbf{F} , além de $h = \Delta t/2$. Além disso, interpretando nossas equações como equações de Stratonovich-Langevin, a segunda integral da Eq.(7.75) deve ser escrita como

$$\int_t^{t+\Delta t} \alpha(\mathbf{r}(t')) \xi(t') dt' = \frac{\alpha(\mathbf{r}_0) + \alpha(\mathbf{r})}{2} \Delta \mathbf{W}(t) = \bar{\alpha} \Delta \mathbf{W}(t) \quad (7.77)$$

Segue, devido à Eq.(7.76), desprezando termos $O(\Delta t^3)$,

$$\mathbf{r} = \mathbf{r}_0 + h(\mathbf{v} + \mathbf{v}_0); \quad (7.78)$$

$$\mathbf{v} = \mathbf{v}_0 + h[(\mathbf{F} + \mathbf{F}_0)/m - \frac{\gamma}{m}(\mathbf{v} + \mathbf{v}_0)] + \frac{\bar{\alpha}}{m} \Delta \mathbf{W}(t). \quad (7.79)$$

Vamos definir $\mathbf{u} = \mathbf{v} + \mathbf{v}_0$. A Eq.(7.79) pode ser escrita na forma

$$\mathbf{u} = 2\mathbf{v}_0 + h[(\mathbf{F} + \mathbf{F}_0)/m - \frac{\gamma}{m}\mathbf{u}] + \frac{\bar{\alpha}}{m} \Delta \mathbf{W}(t); \quad (7.80)$$

Podemos isolar \mathbf{u} e escrever

$$\mathbf{u} = \frac{1}{1 + \gamma h/m} \{2\mathbf{v}_0 + h(\mathbf{F} + \mathbf{F}_0)/m + \frac{\bar{\alpha}}{m} \Delta \mathbf{W}(t)\}, \quad (7.81)$$

que, substituída na Eq.(7.78), resulta em

$$\mathbf{r} = \mathbf{r}_0 + \frac{h}{1 + \gamma h/m} \{2\mathbf{v}_0 + h(\mathbf{F} + \mathbf{F}_0)/m + \frac{\bar{\alpha}}{m} \Delta \mathbf{W}(t)\}. \quad (7.82)$$

Como $\mathbf{F} = \mathbf{F}_0 + O(\Delta t)$ e como $\mathbf{F} + \mathbf{F}_0$, na Eq.(7.82) está multiplicado por h^2 , a substituição de \mathbf{F} por \mathbf{F}_0 nessa equação resulta em erro $O(\Delta t^3)$, que estamos desprezando. Um problema mais sério é a ocorrência de $\bar{\alpha} = (\alpha(\mathbf{r}_0) + \alpha(\mathbf{r}))/2$ na Eq.(7.82), pois a substituição de $\bar{\alpha}$ por $\alpha(\mathbf{r}_0)$ nesta equação causa um erro $O(\Delta t^{\frac{5}{2}})$, que não queremos desprezar. Então definimos um \mathbf{r} aproximado,

$$\mathbf{r}_a = \mathbf{r}_0 + \frac{h}{1 + \gamma h/m} \{2\mathbf{v}_0 + \frac{\Delta t \mathbf{F}_0}{m} + \frac{\alpha(\mathbf{r}_0)}{m} \Delta \mathbf{W}(t)\}, \quad (7.83)$$

cujo erro em relação a \mathbf{r} é $O(\Delta t^{\frac{5}{2}})$. Usando \mathbf{r}_a em vez de \mathbf{r} para recalcular $\bar{\alpha}$ a ser substituído na Eq.(7.82), o erro do termo de ruído será $O(\Delta t^4)$, desprezível.

Portanto, o sistema de equações a ser resolvido em cada passo de integração é constituído pela equação acima, Eq.(7.83) e mais as 3 equações seguintes:

$$\bar{\alpha} = \frac{\alpha(\mathbf{r}_0) + \alpha(\mathbf{r}_a)}{2}, \quad (7.84)$$

$$\mathbf{r} = \mathbf{r}_0 + \frac{h}{1 + \gamma h/m} \left\{ 2\mathbf{v}_0 + \frac{h(\mathbf{F}_0 + \mathbf{F})}{m} + \frac{\bar{\alpha}}{m} \Delta \mathbf{W}(t) \right\} \quad (7.85)$$

e, da Eq.(7.81) segue

$$\mathbf{v} = -\mathbf{v}_0 + \frac{1}{1 + \gamma h/m} \left\{ 2\mathbf{v}_0 + \frac{h(\mathbf{F} + \mathbf{F}_0)}{m} + \frac{\bar{\alpha}}{m} \Delta \mathbf{W}(t) \right\}, \quad (7.86)$$

O sistema de equações Eq.(7.83) até Eq.(7.86) pode ser resolvido, nesta ordem, em cada passo de integração, no intervalo $[t_j, t_{j+1}]$, sendo \mathbf{r}_0 , \mathbf{v}_0 , \mathbf{F}_0 os valores em t_j e \mathbf{r} , \mathbf{v} , \mathbf{F} os valores em t_{j+1} .

É interessante observar que, no caso não dissipativo, i.e. $\gamma = \bar{\alpha} = 0$, o sistema de equações acima se reduz ao algoritmo "Velocity-Verlet". É também interessante observar que, como $\Delta \mathbf{W}(t) = \sqrt{\Delta t} \mathbf{R}_G$, onde \mathbf{R}_G é um vetor cujas componentes são números aleatórios com distribuição normal, tem-se nas equações acima termos que são

$$O(\Delta t^0), O(\Delta t^{1/2}), O(\Delta t^1), O(\Delta t^{3/2}), O(\Delta t^2) \text{ e } O(\Delta t^{5/2})$$

Exemplo 7.7 : Equação de Langevin com potencial quârtico

Para exemplificar o algoritmo acima vamos programar o movimento de uma partícula de massa m em um ambiente com constante dissipativa G e ruído multiplicativo $\alpha = A * (1+r)$, sendo r a posição (unidimensional) e em presença de potencial $V = V_0 r^4/4$, de modo que a força será $F = -Ar^3$.

Programa: langevin_O(2.5).sce

```
// solução da eq. de Langevin até O(dt^(5/2))
// num potencial V=V0*r^4/4,tal que F=-V0*r^3;
// com massa não nula e com ruído multiplicativo:
// alfa=A*(1+r); Exemplo: dt=0.1, tmax=200, V0=.1,
// G=.2, A=.1, m=1
par=input('deh [dt, tmax, V0, G, A, m] \n');
```

```

dt=par(1); tmax=par(2); V0=par(3); G=par(4);
A=par(5); m=par(6);
a=A/m; g=G/m; vo=V0/m;
dtp=dt/10;
nt=round(tmax/dt)+1;
hg=dt/2; hp=dtp/2;
rg=zeros(1,nt); vg=zeros(1,nt);
rp=zeros(1,nt); vp=zeros(1,nt);
rg(1)=1; rp(1)=1;
sqdtp=sqrt(dtp);
Cp=1/(1+g*hp);
Cg=1/(1+g*hg);
for j=1:nt-1
// preciso (dtp=dt/10)
r0=rp(j); v0=vp(j);
dWg=0;
for k=1:10
dWp=sqdtp*rand(1,1,'normal');
dWg=dWg+dWp;
ra=r0+hp*Cp*(2*v0-dtp*vo*r0^3+a*r0*dWp);
armed=(a/2)*(2+r0+ra);
del=2*v0-hp*vo*(r0^3+ra^3)+armed*dWp;
r=r0+hp*Cp*del;
v=-v0+Cp*del;
r0=r; v0=v;
end
rp(j+1)=r;
vp(j+1)=v;
// solução com dt grande
rg0=rg(j); vg0=vg(j);
ra=rg0+hg*Cg*(2*vg0-dt*vo*rg0^3+a*rg0*dWg);
armed=(a/2)*(2+rg0+ra);
Del=2*vg0-hg*vo*(rg0^3+ra^3)+armed*dWg;
rg(j+1)=rg0+hg*Cg*Del;
vg(j+1)=-vg0+Cg*Del;
end

```

```
t=linspace(0,tmax,nt);
xset('window',1); clf
plot(0,0,'w',t,rp,t,rg,'r')
legend('---','dt pequeno', 'dt grande')
```

Exercício 7.9

Execute o programa acima, variando os parâmetros: Tome como padrão os seguintes valores: $dt=0.1$, $tmax=200$, $V0=0.1$, $G=0.2$, $A=0.1$, $m=1$. Execute-o com as seguintes combinações de parâmetros, observando o que segue (com cada combinação de parâmetros execute o programa várias vezes, pois poderá haver flutuações que causam resultados diferentes da tendência dominante):

A) Com os valores padrão:

- 1) Observe que r oscila em torno de um valor negativo; isto ocorre porque $\alpha(r)$ é menor na região $r < 0$, sendo nulo em $r = -1$, e, portanto, a partícula permanece mais tempo nessa região;
- 2) o erro (diferença entre as duas curvas) não é visível com $dt=0.1$ ($dtp=0.01$) para estes parâmetros.

B) Aumenta dt (demais valores, padrão):

- 1) Com $dt=1$ o erro já é visível;
- 2) com $dt=2$ o erro é bem mais pronunciado.

C) Diminui a massa ($m=0.1$, demais valores padrão):

- 1) Oscilações em torno de $r \approx -0.5$;
- 2) flutuações muito mais rápidas.

D) Aumenta o potencial ($V0=1$, demais valores padrão):

Oscilações aproximadamente periódicas, com período $\tau \approx 10$.

E) Aumenta constante dissipativa ($G=1$, demais valores padrão): Oscilações relaxam rapidamente, para flutuações em torno de $r \approx -0.4$

F) Aumenta o ruído ($A=0.5$, demais valores padrão): Erro muito mais notável e flutuações muito mais rápidas.

7.5.3 Movimento Browniano em Coordenadas Polares

As forças estocásticas (ruído branco) sobre uma partícula em movimento Browniano num líquido isotrópico podem ser decompostas em componentes ortogonais independentes. As coordenadas Cartesianas são, por isso, as mais naturais para se tratar as correspondentes equações de Langevin. Já o movimento Browniano rotacional de partículas extensas ou dipolos são mais naturalmente tratados em coordenadas polares. Algumas peculiaridades do cálculo estocástico são particularmente notáveis neste caso. Para facilitar o entendimento dos aspectos que queremos explicitar, vamos tratar inicialmente um exemplo bem simples, em duas dimensões.

Movimento Browniano Bidimensional em Coordenadas Planopolares

Consideremos uma partícula puntiforme na superfície de um líquido contido num recipiente circular. Escolhemos o centro do recipiente como origem do sistema de coordenadas. Sejam (x, y) as coordenadas Cartesianas e (r, ϕ) as coordenadas planopolares da partícula. Temos as seguintes relações:

$$\begin{aligned} x &= r \cos \phi \\ y &= r \sin \phi \\ r &= \sqrt{x^2 + y^2} \\ \phi &= \arctan \frac{y}{x} \end{aligned} \tag{7.87}$$

Vamos considerar o limite de massa nula e sem campo externo. As equações de movimento, neste caso, reduzem-se a um balanço entre a resistência viscosa e as forças estocásticas. Em coordenadas cartesianas,

$$\begin{aligned} \gamma dx &= \sigma dW_x(t) \rightarrow dx = b dW_x(t) \\ \gamma dy &= \sigma dW_y(t) \rightarrow dy = b dW_y(t) \end{aligned} \tag{7.88}$$

onde $b = \sigma/\gamma$. Para transformar essas equações para coordenadas polares precisamos decidir se usamos o cálculo de Ito ou de Stratonovich. Vamos fazer a transformação por ambos os cálculos, iniciando pelo de Ito.

Transformação por Ito:

Lembramos que como $r = r(x, y)$, sua diferencial, por cálculo de Ito deve ser calculada até segunda ordem em dx e dy ,

$$dr = \frac{\partial r}{\partial x} dx + \frac{\partial r}{\partial y} dy + \frac{1}{2} \left\{ \frac{\partial^2 r}{\partial x^2} dx^2 + \frac{\partial^2 r}{\partial y^2} dy^2 + 2 * \frac{\partial^2 r}{\partial x \partial y} dx dy \right\} \quad (7.89)$$

e semelhantemente para $d\phi$. Das Eqs. 7.87 e 7.88 segue, após algum cálculo,

$$\begin{aligned} dr &= b \cos(\phi) dW_x(t) + b \sin(\phi) dW_y(t) + \\ &+ \frac{b^2}{2r} \left\{ \sin^2(\phi) dW_x(t)^2 + \cos^2(\phi) dW_y(t)^2 + \right. \\ &\left. - 2 \cos(\phi) \sin(\phi) dW_x(t) dW_y(t) \right\} \end{aligned} \quad (7.90)$$

Usando-se a Eq.(7.64) a Eq.(7.90) simplifica-se para

$$dr = b \cos(\phi) dW_x(t) + b \sin(\phi) dW_y(t) + \frac{b^2}{2r} dt. \quad (7.91)$$

Para determinar as componentes $dW_x(t)$ e $dW_y(t)$ podemos escolher os eixos x e y convenientemente a cada passo de integração. A escolha mais simples é x paralelo a r e y perpendicular a r , pois então $\cos(\phi) = 1$ e $\sin(\phi) = 0$. Vamos chamar os correspondentes incrementos de Wiener $dW_r(t)$ e $dW_\phi(t)$. A Eq.(7.91) reduz-se a

$$dr = b dW_r(t) + \frac{b^2}{2r} dt. \quad (7.92)$$

Por procedimento análogo ao seguido para obter dr , obtemos para $d\phi$,

$$d\phi = \frac{b}{r} dW_\phi(t). \quad (7.93)$$

As correspondentes equações a diferenças finitas são

$$\begin{aligned}\Delta r &= b \Delta W_r + \frac{b^2}{2r} \Delta t \\ \Delta \phi &= \frac{b}{r} \Delta W_\phi.\end{aligned} \quad (7.94)$$

Cuidado deve ser tomado, para não introduzir grandes erros na integração numérica, que se tenha sempre $b^2 \Delta t \ll r$ e $b \Delta W_\phi \ll r$. Podemos minimizar o problema associado a se ter r no denominador das Eqs.(7.94) substituindo a equação para $\Delta \phi$ por

$$\Delta \phi = \arctan\left(\frac{b \Delta W_\phi}{r}\right) \quad (7.95)$$

que tem o mesmo valor da segunda das Eqs.(7.94) no limite $\Delta t \rightarrow 0$ mas evita variações absurdas de ϕ quando r é muito pequeno.

Transformação por Stratonovich:

Como no cálculo usual, no cálculo de Stratonovich dr e $d\phi$ são combinações das diferenciais primeiras dx e dy , ou seja, de dW_x e dW_y :

$$\begin{aligned}dr &= b \cos(\phi) dW_x(t) + b \sin(\phi) dW_y(t) \\ d\phi &= -\frac{b \sin(\phi)}{r} dW_x(t) + \frac{b \cos(\phi)}{r} dW_y(t)\end{aligned} \quad (7.96)$$

A forma para diferenças finitas exige agora os valores das coordenadas no início e no fim do intervalo Δt . Por isso, mesmo que escolhermos x paralelo a r no início do intervalo, esse não será mais o caso, em geral, no fim do intervalo. Por isso não podemos pôr simplesmente $\sin(\phi) = 0$ nas equações acima, como fizemos no caso da transformação por cálculo de Ito. Com esta escolha as diferenças finitas de Stratonovich ficam

$$\begin{aligned}\Delta\phi &= \arctan\left(\frac{b\Delta W_\phi}{r}\right) \\ \Delta r &= b\Delta W_r + b\sin\left(\frac{1}{2}\Delta\phi\right)\Delta W_\phi\end{aligned}\quad (7.97)$$

onde, na primeira das equações acima, usamos o mesmo argumento usado para escrevermos a Eq.(7.95). O leitor pode verificar que, até segunda ordem em ΔW as Eqs.(7.94) e 7.97 são iguais.

Discretização Ingênua:

Vamos chamar de "Discretização Ingênua" o procedimento de integrar numericamente as equações diferenciais de Stratonovich (que são as mesmas que seriam obtidas por cálculo usual), Eqs.(7.96), como se fossem equações diferenciais "ordinárias", isto é, sem usar a prescrição de Stratonovich. As equações de diferenças finitas ficariam, neste caso:

$$\begin{aligned}\Delta\phi &= \arctan\left(\frac{b\Delta W_\phi}{r}\right) \\ \Delta r &= b\Delta W_r;\end{aligned}\quad (7.98)$$

Resolvendo o exercício abaixo o leitor constatará que as Eqs.(7.94) e (7.97) levam a uma distribuição aproximadamente homogênea das partículas que se difundem, mas a Eq.(7.98) leva a uma concentração maior na região central, o que, claramente, está errado.

Exercício 7.10

Programe a solução numérica das Eqs.(7.94), (7.97) e (7.98) para um ensemble de N partículas Brownianas. Como condição de contorno suponha uma parede circular refletora em $r = 1$. Você implementa esta condição de contorno procurando, ao final do passo de integração, as partículas com $r > 1$ e fazendo, para elas, a substituição $r \rightarrow 2 - r$. Ponha as posições das partículas num gráfico e observe como o ensemble se difunde, ao longo do tempo.

Observação 1: Para r muito pequeno pode ocorrer $|\Delta r| > r$; então, se $\Delta r < 0$, resulta um novo r negativo; estes casos devem ser corrigidos para $r \rightarrow \text{abs}(r)$ e $\phi \rightarrow \phi + \pi$;

Observação 2: Para N muito grande o gráfico das posições das partículas torna-se um borrão único; neste caso é melhor substituir o gráfico das posições por um histograma de r , a ser observado de tempos em tempos ao longo do processo de integração.

7.6 A Equação de Fokker-Planck

As variáveis dinâmicas, ou observáveis, de um sistema físico de muitos corpos (sistema termodinâmico), correspondem, como vimos no capítulo V, a médias sobre os possíveis microestados do sistema. No caso de processos estocásticos, as referidas médias são sobre as realizações. Como vimos acima, conhecendo-se as equações que governam os processos estocásticos, é possível simular as realizações e calcular as médias desejadas. Uma maneira alternativa consiste em encontrar uma equação para a distribuição de probabilidade do processo, em função do tempo, e calcular as médias desejadas integrando sobre o espaço das variáveis do sistema, tendo a distribuição de probabilidade como peso, com vimos no capítulo III.

Ao sistema de equações diferenciais estocásticas, existe associada uma equação diferencial a derivadas parciais para a distribuição de probabilidade das variáveis, que se chama **Equação de Fokker-Planck**. Foge do espírito e das intenções deste livro texto sobre métodos computacionais derivar a Equação de Fokker-Planck. Ótimas deduções podem ser encontradas nas referências [10, 18]. Aqui basta-nos apresentar a referida equação e dela fazer uso. Consideremos inicialmente o caso unidimensional. Seja $X(t)$ um processo estocástico Markoviano, que satisfaz a seguinte equação diferencial estocástica de Ito:

$$dX(t) = A(X, t)dt + B(X, t) \bullet dW(t). \quad (7.99)$$

A correspondente equação de Fokker-Planck para a distribuição de probabilidade $P_X(x, t)$, que passaremos a denotar, por simplicidade, por $P(x, t)$, é

$$\frac{\partial P(x, t)}{\partial t} = \frac{\partial [K(x, t)P(x, t)]}{\partial x} + \frac{1}{2} \frac{\partial^2 [D(x, t)P(x, t)]}{\partial x^2}, \quad (7.100)$$

onde

$$K(x, t) = -A(x, t) \quad (7.101)$$

e

$$D(x, t) = B^2(x, t). \quad (7.102)$$

Para que a solução da Eq.(7.100) seja a mesma distribuição de $x(t)$ como gerada por uma simulação da equação diferencial estocástica, Eq.(7.99), a condição inicial $P(x, 0)$ deve ser igual à distribuição inicial de valores de X usada na simulação. Em particular, para uma simulação de realizações a partir de $X(0) = x_0$ a condição inicial da Eq.(7.100) é $P(x, 0) = \delta(x - x_0)$. Neste caso a distribuição de probabilidade, solução da equação de Fokker-Planck, é denotada por $T(x, t | x_0, 0)$, isto é,

$$T(x, t | x_0, 0) = P(x, t) \quad \text{quando} \quad P(x, 0) = \delta(x - x_0) \quad (7.103)$$

e é chamada de *probabilidade de transição* de x_0 em $t = 0$ para x em t .

Se a equação diferencial estocástica em questão não for de Ito mas de Stratonovich, Eq.(7.55), com os mesmos coeficientes $A(X, t)$ e $B(X, t)$, então o correspondente $A(X, t)$ de Ito é dado pela Eq.(7.45) e, portanto, a equação de Fokker-Planck correspondente tem a mesma forma da Eq.(7.100) mas com coeficientes dados por

$$K(x, t) = -A(x, t) - \frac{1}{2} \frac{\partial B}{\partial X} B(X(t), t), \quad (7.104)$$

e

$$D(x, t) = B^2(x, t). \quad (7.105)$$

É claro que se a EDE for de ruído aditivo, ou seja, B independente de X , então não há diferença entre as versões de Ito e de Stratonovich. O leitor pode verificar, além disso, que também em caso de ruído multiplicativo, a Equação de Fokker-Planck que se obtém de uma EDE de Stratonovich é a mesma que se obtém da correspondente EDE de Ito. É natural que seja assim uma vez que a Equação de Fokker-Planck não é uma equação estocástica e sua solução é uma

função bem determinada do tempo a das coordenadas, que representa a distribuição de probabilidade para a grandeza física $X(t)$, não havendo, portanto, lugar para ambiguidades.

Os coeficientes K e D da equação de Fokker-Planck são conhecidos como “coeficiente de deriva” (drift) e de “difusão”, respectivamente.

Exemplo 7.8: O Processo de Ornstein-Uhlenbeck

Consideremos uma partícula de massa m em movimento Browniano unidimensional, sobre a qual, além das forças de ruído, $\xi(t)$, e viscosa, $-\gamma\dot{X}$, atua uma força de oscilador harmônico, $-KX$. A equação de Langevin correspondente é

$$m\ddot{X} + \gamma\dot{X} = -KX + B\xi(t). \quad (7.106)$$

Para viscosidades típicas de líquidos o termo de inércia é, geralmente, muito menor que os demais e pode ser desprezado. A Eq.(7.106) se reduz, então a

$$dX(t) = -kX(t)dt + \beta dW(t), \quad (7.107)$$

onde $k = K/\gamma$ e $\beta = B/\gamma$. Este é o processo de Ornstein-Uhlenbeck. A correspondente equação de Fokker-Planck é

$$\frac{\partial P(x, t)}{\partial t} = \frac{\partial[kxP(x, t)]}{\partial x} + \frac{1}{2}D\frac{\partial^2 P(x, t)}{\partial x^2}, \quad (7.108)$$

onde $D = \beta^2$.

No caso em que $k = 0$ a Eq.(7.108) se reduz à familiar equação da difusão. O aspecto interessante do processo de Ornstein-Uhlenbeck é que possui solução estacionária, localizada em torno de $x = 0$.

Exercício 7.11:

Simular N realizações do processo de Ornstein-Uhlenbeck com condição inicial $X(0) = 1$ e calcular média e variância de X em função do tempo, até que estas funções se tornem aproximadamente constantes. Resolver a correspondente equação de Fokker-Planck, com a mesma condição inicial, e calcular média e variância de X integrando sobre a distribuição de probabilidade. Comparar os resultados obtidos pelos dois métodos.

7.6.1 Relação de Einstein

Em Mecânica Estatística de Não-Equilíbrio é conhecida uma relação, chamada de *Teorema da Flutuação-Dissipação* [18], entre a dissipação de energia de uma partícula em movimento dentro de um fluido e as flutuações nas coordenadas da partícula devidas às colisões das moléculas do fluido com a mesma. É natural esperar que tal relação exista, uma vez que o mecanismo físico que provoca as flutuações nas coordenadas é o mesmo que transfere energia da partícula para o fluido. Além disso, num sistema Browniano em equilíbrio, a energia transferida das moléculas do fluido para as partículas é, por definição de equilíbrio, igual à transferida das partículas para as moléculas.

Uma manifestação particularmente simples e útil do Teorema da Flutuação-Dissipação é a chamada *Relação de Einstein* entre o coeficiente dissipativo da Equação de Langevin (γ , no exemplo do PE de Ornstein-Uhlenbeck) e o coeficiente de difusão, D . Essa relação pode ser obtida sem se recorrer à forma geral do Teorema da Flutuação-Dissipação, pelo procedimento que segue.

Consideremos uma partícula Browniana em presença de um potencial $V(x)$ capaz de localizar a mesma numa região do espaço, de maneira a se estabelecer, após um tempo suficiente, uma distribuição de equilíbrio, $P(x, t) = P(x)$. A correspondente equação de Langevin é

$$m\ddot{X} + \gamma\dot{X} = -V'(X) + \alpha\xi(t). \quad (7.109)$$

onde $V'(X)$ é a derivada de V em relação a X , e α e γ são constantes. Se desprezamos o termo de inércia obtemos

$$dX(t) = \frac{-V'(X)}{\gamma}dt + \frac{\alpha}{\gamma}dW(t) \quad (7.110)$$

e a correspondente equação de Fokker-Planck é

$$\frac{\partial P}{\partial t} = \frac{1}{\gamma} \frac{\partial[V'(x)P]}{\partial x} + \frac{D}{2} \frac{\partial^2 P}{\partial x^2}, \quad (7.111)$$

onde $D = \alpha^2/\gamma^2$. Quando se estabelece o equilíbrio, P se torna independente do tempo e a equação acima fica

$$\frac{1}{\gamma} \frac{\partial[V'(x)P]}{\partial x} + \frac{D}{2} \frac{\partial^2 P}{\partial x^2} = 0 .$$

Integrando em x segue

$$\frac{V'(x)P}{\gamma} + \frac{D}{2} \frac{\partial P}{\partial x} = \text{constante} .$$

Como P é a distribuição de equilíbrio sua dependência em x é dada pela distribuição de Boltzmann, i.e.,

$$P \propto \exp -\frac{V(x)}{k_B T},$$

que, substituída na equação acima dá

$$\frac{V'(x)}{\gamma} \exp \left[-\frac{V(x)}{k_B T} \right] - \frac{DV'(x)}{2k_B T} \exp \left[-\frac{V(x)}{k_B T} \right] = \text{constante}.$$

Como nos pontos de mínimo do potencial se tem $V'(x) = 0$, segue que a constante é nula, o que nos leva à *Relação de Einstein*,

$$\frac{1}{\gamma} = \frac{D}{2k_B T} \quad \text{ou} \quad \gamma D = 2k_B T \quad \text{ou} \quad \frac{\alpha^2}{2\gamma} = k_B T . \quad (7.112)$$

É notável que, embora γ e D dependam das características da partícula e do fluido, o produto γD depende só da temperatura, não importando se a partícula é grande ou pequena ou se o fluido é mais ou menos viscoso. Outro ponto importante é que, embora tenhamos obtido as relações acima no limite $m = 0$, se usarmos a equação completa, Eq.(7.109), com $m \neq 0$, obteremos as mesmas relações, embora a demonstração seja mais trabalhosa. Neste caso a distribuição de Boltzmann deve incluir a energia cinética.

7.6.2 Equação de Fokker-Planck para PE Vetorial com Vários Ruídos

Quando $X(t)$ é um processo estocástico vetorial, i.e.,

$$X = [X_1, \dots, X_i, \dots, X_n],$$

então a equação de Fokker-Planck adquire a forma

$$\frac{\partial P(x, t)}{\partial t} = \sum_{i=1}^n \frac{\partial [K_i(x, t)P(x, t)]}{\partial x_i} + \frac{1}{2} \sum_{i,j=1}^n \frac{\partial^2 [D_{i,j}(x, t)P(x, t)]}{\partial x_i \partial x_j}. \quad (7.113)$$

Se o PE $X(t)$ satisfaz a EDE de Ito com ruídos independentes, Eq.(7.66), então os coeficientes K e D estão relacionados com A e B por

$$K_i(x, t) = -A_i(x, t) \quad (7.114)$$

e

$$D_{i,j}(x, t) = \sum_{\alpha=1}^m B_{i,\alpha}(x, t)B_{j,\alpha}(x, t). \quad (7.115)$$

Se, entretanto, o PE $X(t)$ satisfaz a EDE de Stratonovich, Eq.(7.55), então os coeficientes K e D estão relacionados com A e B por

$$K_i(x, t) = -A_i(x, t) - \frac{1}{2} \sum_{j=1}^n \sum_{\alpha=1}^m \frac{\partial B_{i,\alpha}(x, t)}{\partial x_j} B_{j,\alpha}(x, t).$$

e

$$D_{i,j}(x, t) = \sum_{\alpha=1}^m B_{i,\alpha}(x, t)B_{j,\alpha}(x, t).$$

Exemplo 7.9: Vetor girante por Fokker-Planck

Consideremos novamente o vetor girante, como no exemplo 7.5, ou seja, com ruídos independentes sobre as componentes X e Y . Os coeficientes da equação de Fokker-Planck são dados pelas Eqs.(7.114) e (7.115), que, comparando com as Eqs.(7.69) dão

$$\begin{aligned} K_x &= w_0 y \\ K_y &= -w_0 x \\ D_{xx} &= \beta^2 y^2 \\ D_{yy} &= \beta^2 x^2 \\ D_{xy} &= D_{yx} = 0. \end{aligned} \quad (7.116)$$

Logo, a equação de Fokker-Planck é

$$\frac{\partial P(x, y, t)}{\partial t} = w_0 y \frac{\partial P(x, y, t)}{\partial x} - w_0 x \frac{\partial P(x, y, t)}{\partial y} + \\ + \frac{\beta^2 y^2}{2} \frac{\partial^2 P(x, y, t)}{\partial x^2} + \frac{\beta^2 x^2}{2} \frac{\partial^2 P(x, y, t)}{\partial y^2} \quad (7.117)$$

Exercício 7.12:

Resolvendo numericamente a Eq.(7.117), por algum dos algoritmos vistos no capítulo III, a partir de uma distribuição inicial qualquer, o leitor encontrará $P(x, y, t)$ e poderá então calcular

$$\langle X(t) \rangle = \int P(x, y, t) x \, dx \, dy,$$

$\langle Y(t) \rangle$ e $\langle R^2(t) \rangle$ e comparar os resultados com os obtidos por simulação na seção 7.5. Se usar a mesma distribuição inicial e os mesmos parâmetros w_0 e β , os resultados devem ser os mesmos.

7.7 Teoria da Resposta Linear para Processos Estocásticos

Consideremos um sistema físico em equilíbrio termodinâmico a uma dada temperatura. Sobre esse sistema aplicamos um campo perturbativo $F(t)$ (por ex., um campo elétrico ou magnético). Seja $M(x)$ uma variável dinâmica (observável) do sistema, cujo valor em equilíbrio é $\langle M(x) \rangle_0$. O valor esperado de $M(x)$ em presença do campo perturbativo será denotado por $\langle M(x) \rangle_t$ e o desvio do valor esperado de $M(x)$ devido ao campo será

$$\delta M(t) = \langle M(x) \rangle_t - \langle M(x) \rangle_0, \quad (7.118)$$

que se chama “resposta de M ao campo perturbativo”.

Supomos que o início da aplicação do campo perturbativo ocorreu no passado remoto, $t_0 = -\infty$ e que ele é suficientemente fraco para que $\delta M(t)$ seja um funcional linear de $F(t)$, i.e.,

$$\delta M(t) = \int_{-\infty}^t \Phi(t, t') F(t') dt'. \quad (7.119)$$

A função $\Phi(t, t')$ se chama “função resposta”. Como estamos supondo que $\delta M(t)$ é linear em F , vê-se da Eq.(7.119) que a Φ não pode depender de F e como, na ausência de F o sistema está em equilíbrio, vemos que a dependência temporal de Φ só pode ser na forma $t - t'$, i.e., Φ só depende do intervalo de tempo entre a “causa” $F(t')$ e o efeito no instante t , ou seja, $\Phi(t, t') = \Phi(t - t')$.

Consideremos um caso especial em que $F(t')$ é um pulso unitário de duração muito curta no instante $t_0 < t$, i.e., $F(t') = \delta(t' - t_0)$. A integral na Eq.(7.119) é, então, trivial, resultando em

$$\delta M(t) = \Phi(t - t_0), \quad (7.120)$$

ou seja, a função resposta $\Phi(t - t_0)$ é a resposta no instante t a um pulso unitário no instante t_0 .

No laboratório dificilmente se aplica um pulso, localizado no tempo, mas muito mais comumente, um campo oscilante com uma dada frequência, ω . Façamos, por isso, a transformada de Fourier da Eq.(7.119),

$$\begin{aligned} \delta M(\omega) &= \int_{-\infty}^{\infty} dt \exp(-i\omega t) \int_{-\infty}^t dt' \Phi(t - t') F(t') \\ &= \int_{-\infty}^{\infty} \exp(-i\omega t) dt \int_{-\infty}^{\infty} dt' \Phi(t - t') F(t') \\ &= \chi(\omega) F(\omega), \end{aligned} \quad (7.121)$$

onde, na segunda linha, estendemos a integral em t' para $+\infty$, invocando o princípio da causalidade, ou seja, $\Phi(t - t')$ é nula para $t' > t$, e na terceira linha usamos a propriedade de que a transformada de Fourier de uma convolução é o produto das transformadas das funções e definimos

$$\chi(\omega) = \int_0^{\infty} \exp(-i\omega t) \Phi(t) dt. \quad (7.122)$$

sendo a integral no intervalo $[0, \infty)$ porque a $\Phi(t)$ é nula para $t < 0$. A função $\chi(\omega)$ se chama “suscetibilidade dinâmica” e a transformada que a define se chama “transformada de Fourier-Laplace”, por que usa o “kernel” $\exp(-i\omega t)$, da transformada de Fourier e os limites de integração, $[0, \infty)$, da transformada de Laplace.

A resposta $\delta M(t)$ a um campo periódico $F(t)$ de frequência ω é também periódica com frequência ω , como se vê na Eq.(7.121). Entretanto, $\delta M(t)$ está, geralmente defasada de $F(t)$. Podemos decompôr $\delta M(t)$ em dois termos,

$$\delta M(t) = \delta M'(t) + \delta M''(t),$$

onde $\delta M'(t)$ está em fase com $F(t)$ e $\delta M''(t)$ está defasada de $\pi/2$. É usual escrever a função complexa $\chi(\omega)$ em termos de suas partes real e imaginária,

$$\chi(\omega) = \chi'(\omega) + i\chi''(\omega). \quad (7.123)$$

A parte real, $\chi'(\omega)$, é a constante de proporcionalidade entre a amplitude da resposta em fase com o campo, $\delta M'(t)$, e o campo e a parte imaginária, $\chi''(\omega)$, é a constante de proporcionalidade entre a amplitude da resposta defasada, $\delta M''(t)$, e o campo. Numa experiência de ressonância magnética a potência dissipada na cavidade ressonante, que é a grandeza física medida, é proporcional a $\omega \chi''(\omega)$.

Na formulação usual da Teoria da Resposta Linear [18], a função resposta é calculada com auxílio da Equação de Liouville, que dá a dependência temporal da distribuição de probabilidade para os microestados, tanto na abordagem clássica como na quântica. A aplicação de um campo perturbativo $F(t)$ sobre um sistema físico altera sua energia. A energia de interação do campo com o sistema pode, em geral, ser escrita na forma

$$V(t) = -\mu(t) F(t), \quad (7.124)$$

onde $\mu(t)$ é uma variável dinâmica do sistema. Observamos que em muitos casos μ e F são vetores e a Eq.(7.124) deve ser entendida como produto escalar. Como exemplos podemos citar a interação de um campo elétrico $\mathbf{E}(t)$ com os dipolos elétricos $\boldsymbol{\mu}$ das moléculas da amostra ou a interação de um campo magnético $\mathbf{H}(t)$ com os momentos magnéticos atômicos, moleculares ou de partículas superparamagnéticas. Existe uma relação entre a "função resposta" e a função de correlação das variáveis dinâmicas $M(t)$ e $\mu(t')$, que se chama "fórmula de Kubo". Vamos introduzir a notação

$$A(t) = M(t) - \langle M \rangle_0 \quad \text{e} \quad B(t) = \mu(t) - \langle \mu \rangle_0$$

onde $\langle M \rangle_0$ e $\langle \mu \rangle_0$ são os valores de equilíbrio, na ausência do campo perturbativo $F(t)$. No caso em que M e μ são variáveis dinâmicas clássicas, a "fórmula de Kubo" adquire a forma

$$\Phi(t - t') = \frac{-1}{k_B T} \langle \dot{A}(t) B(t') \rangle \quad (7.125)$$

onde $\dot{A}(t)$ é a derivada de $A(t)$ em relação a t . Uma demonstração desta fórmula pode ser encontrada, por exemplo, no livro de Kubo [18]

Exemplo 7.10: Função Resposta e Susceptibilidade

Como exemplo de cálculo de uma função resposta e correspondente susceptibilidade dinâmica vamos considerar uma partícula de massa m num potencial de oscilador harmônico, com ruído e força dissipativa. Para resolver a equação de movimento usamos o método introduzido na seção 7.5.2. Resolvemos simultaneamente as equações de movimento de N (por exemplo, 100000) partículas. Obtendo r e v das partículas a cada tempo de integração, podemos obter $\phi(t)$ pela fórmula de Kubo, e, por transformada de Fourier discreta, obter a susceptibilidade dinâmica, $\chi(\omega)$. O programa usa a "function" tfd.sci, apresentada no Capítulo I. Por isso uma cópia de "tfd.sci" deve ser incluída no mesmo diretório em que está "ressonancia.sce".

Programa: ressonancia.sce

```
// solução da eq. de Langevin até 0(dt^(5/2))
// num potencial de oscilador harmônico,
// V=V0*r^2/2, tal que F=-V0*r,
// com massa não nula e com ruído aditivo:
// alfa=A; Calcula a função resposta em função
// do tempo e a susceptibilidade dinâmica.
// Exemplo: dt=0.1, tmax=80, V0=.2,
// G=.1, A=.1, m=1, N=10000

par=input('deh [dt, tmax, V0, G, A, m, N] \n');
dt=par(1); tmax=par(2); V0=par(3); G=par(4);
A=par(5); m=par(6); N=par(7);
```

```
a=A/m; g=G/m; v0=V0/m;
nt=round(tmax/dt)+1;
h=dt/2;
r=ones(1,N); v=zeros(1,N);
sqdt=sqrt(dt);
C=1/(1+g*h);
fi=zeros(1,nt);
for j=2:nt
    r0=r; v0=v;
    dW=sqdt*rand(1,N,'normal');
    ra=r0+h*C*(2*v0-dt*v0*r0+a*dW);
    del=2*v0-h*v0*(r0+ra)+a*dW;
    r=r0+h*C*del;
    v=-v0+C*del;
    fi(j)=-v*r'/N;
end

t=linspace(0,tmax,nt);
xset('window',1); clf
plot(t,fi)
getf tfd.sci;
chi=tfd(t,fi);
```

Exercício 7.13:

Execute o programa acima, inicialmente com os valores sugeridos para os parâmetros, e a seguir, alterando cada vez um dos parâmetros. Observe e descreva os efeitos das mudanças de cada parâmetro sobre a susceptibilidade. Lembre que no diretório deve haver uma cópia da "function" tfd.sci, introduzida na secção 1.4.3.

Apêndice A: INTRODUÇÃO A SCILAB

SCILAB é uma linguagem de programação de alto nível para cálculo numérico. É “software” livre e existem distribuições gratuitas, tanto para LINUX como para WINDOWS. Sua sintaxe é semelhante à do MATLAB, mas há um número significativo de diferenças em relação a MATLAB. Por isso criamos o Apêndice B, apresentado a seguir, que lista as principais modificações que devem ser feitas nos programas escritos para SCILAB se o leitor optar por executá-los em MATLAB. O texto a seguir foi escrito como uma introdução à linguagem SCILAB de programação.

Trata-se de uma linguagem computacional para cálculo numérico, cujas variáveis são naturalmente matrizes: um escalar é uma matriz 1×1 , um vetor fila ou coluna de n componentes é uma matriz $1 \times n$ ou $n \times 1$, além das matrizes propriamente ditas, $m \times n$. Isto evita muitas recorrências (loops) na programação, pois as operações são naturalmente interpretadas como “operações matriciais”. Além disso SCILAB trabalha com números complexos sem que o programador precise declará-los como tal. Em suma, trata-se de uma linguagem de programação extremamente fácil e eficiente. Ela não se propõe a competir com FORTRAN ou C em rapidez de execução nos casos de computação de alto desempenho, mas apresenta enormes vantagens de simplicidade e praticidade, como o leitor verá com a prática. Nesta introdução procuraremos apresentar as noções mais básicas de SCILAB, para levar o leitor ao ponto de escrever programas. Um conhecimento mais sofisticado poderá ser adquirido por consulta aos manuais de SCILAB ou ao “help on line” (ajuda na linha de comando; por ex., “help format”). Abaixo, apresentamos uma espécie de “recei-

tuário mínimo” para se trabalhar com SCILAB. Sugerimos ao leitor que leia o texto com a janela SCILAB aberta, executando todas as instruções e os exemplos mencionados. Aspas, “...”, são usadas no texto para destacar os termos técnicos de SCILAB ou texto escrito por nós no “prompt”, mas não devem ser usadas pelo operador ao empregar estes termos. A tecla *enter* é indicada, em algumas instruções, por \leftarrow .

1.– Formatos das variáveis na tela: Há dois “tipos” de formatos de apresentação de números na tela: ‘v’ e ‘e’. O tipo ‘v’ apresenta o número na forma usual, com um ponto separando a parte inteira da fracionária; por exemplo, “1234.5678”; ‘e’ escreve na forma de “notação científica”, como 1.235D+03. Um segundo argumento pode ser usado em “format”, o número de “toques”. Por exemplo, escrevendo “format(‘e’,14) \leftarrow , %pi \leftarrow aparece na tela 3.1415927D+00. Neste tipo ‘e’ contém 6 algarismos menos que o número de toques: 1 para o sinal (oculto no caso de ser +), 1 para o ponto, 4 para D+00. No caso de tipo ‘v’ o número de algarismos é apenas de 2 menor que o número de toques. Exemplo: “format(‘v’, 8) \leftarrow %pi \leftarrow produz 3.14159. O nome “ans” aparece sempre associado ao resultado da última operação que não recebeu nome próprio; por exemplo, 3+5 \leftarrow produz ans=8; entretanto se escrevemos x=3+5 \leftarrow segue: x=8. Se queremos na tela apenas o valor de x, sem o símbolo “x=”, escrevemos “disp(x)”. É interessante notar que se escrevermos disp(x,y,z) aparece na tela os valores de z,x,y, isto é, na ordem inversa à que está no argumento de “disp”.

2.– Operações fundamentais: SCILAB é uma ótima máquina de calcular na linha de comando. Por exemplo, escreva 3+5 \leftarrow (soma), 8-3 \leftarrow (subtração), 5*6 \leftarrow (multiplicação) 20/4 \leftarrow (divisão), 2^3 \leftarrow (potenciação), sqrt(-25) \leftarrow (raiz quadrada; note que números complexos são tratados normalmente), sin(% pi/4) \leftarrow ou x=4*atan(1) \leftarrow (funções trigonométricas), etc;

3.– Workspace: Chama-se assim o conjunto de variáveis disponíveis para serem usadas na linha de comando ou em programas. O comando “who” fornece na tela a lista dessas variáveis, tanto as que

foram definidas pelo operador como as que são intrínsecas a SCILAB, variáveis numéricas, booleanas e de texto, assim como uma lista de “bibliotecas” disponíveis. Por exemplo, defina $x=2.3$, $y=1+2*sqrt(-2)$, $z=[1, 0.4, -2]$ e entre o comando “who” e veja o resultado. O comando “whos” apresenta os mesmos objetos que “who”, mas acompanhados de suas dimensões e ocupação de memória. Entre “whos” e observe o resultado. O comando “who_user” apresenta apenas os objetos (variáveis, funções, etc.) definidos ou usados pelo operador.

4.- Variáveis são “case sensitive”: Isto é, a visível A é diferente da variável a. Letras minúsculas são usadas em todos os comandos de SCILAB.

5.- O comando “clear”: Apaga todas as variáveis; “clear a b c” só apaga as variáveis explicitadas (a, b, c, neste caso); “clear” é muito útil em programas (veja script-files, abaixo) para evitar erros em repetições do programa com dimensões diferentes das matrizes.

6.- Variáveis especiais: SCILAB possui um conjunto de variáveis que estão permanentemente disponíveis. As variáveis “intrínsecas” numéricas são:

$\%i$ =unidade imaginária= $\sqrt{-1}$; $\%pi=\pi=3.141592\dots$; $\%nan=Nan$ (not a number)=“não tem valor definido”, como $0/0$; $\%eps$ é o menor valor que SCILAB consegue distinguir de 1, geralmente $\approx 10^{-16}$; $\%inf=\infty$; $\%e=e=2.718281828\dots$; As variáveis intrínsecas lógicas (booleanas) são: $\%t=T(true=verdadeiro)$; $\%f=F(false=falso)$; há outras menos relevantes (veja com “whos”).

7.- Tipos de variáveis: *Escalar*= número, real ou complexo; *vetor*= fila ou coluna de números; *matriz* $m \times n$; entra-se uma matriz no “workspace” assim: $[a_{11}, a_{12}, \dots, a_{1n}; a_{21}, a_{22}, \dots, a_{2n}; \dots; a_{m1}, \dots, a_{mn}]$; a vírgula entre elementos de uma fila é opcional (espaço é suficiente), mas o “ponto e vírgula” para começar nova fila é necessário. Exemplo: entrando $A=[1\ 2\ 3\ 10;\ 4\ 5\ 6\ 11;\ 7\ 8\ 9\ 12]$, segue

A=

1	2	3	10
4	5	6	11
7	8	9	12

Há também as variáveis *caracter* (character strings). Estas são definidas por expressões entre apóstrofes. Por exemplo, digite t='eis' uma variável tipo caracter' \leftarrow t \leftarrow e veja o resultado.

8.– Dimensões de matrizes: O comando "size" dá as duas dimensões da matriz e o comando "length" dá o número total de elementos; "length" é mais usado para vetores. Por exemplo, escreva size(A) \leftarrow . As dimensões das matrizes (e vetores) são muito importantes nas operações matriciais e, por isso, os comandos acima são muitas vezes usados em programas.

9.– Matrizes especiais: Assim como as variáveis escalares especiais, há também matrizes especiais definidas em SCILAB: "zeros(m,n)"= matriz $m \times n$ de elementos nulos; "ones(m,n)"= matriz $m \times n$ com elementos iguais a 1; "rand(m,n)"= matriz $m \times n$ de elementos aleatórios, uniformemente distribuídos, no intervalo [0, 1]; "rand(m,n,'normal')"= matriz $m \times n$ de elementos aleatórios, com distribuição Gaussiana (normal), centrada em 0 e largura 1; "grand(m,n,'distrib',[p1,p2, ...])"= matriz $m \times n$ de números aleatórios com distribuição 'distrib' e parâmetros p1, p2, ... (veja "help grand"); "eye(n,n)"= matriz identidade $n \times n$. Há várias outras matrizes especiais (menos importantes), que podem ser encontradas nos manuais de SCILAB.

10.– Funções elementares: Funções como $\sin(x)$, $\cos(x)$, \sqrt{x} , $\exp(x)$, $\log(x)$, $\text{atan}(x)$, $\text{abs}(x)$, etc., têm o significado usual em matemática. Se x é uma matriz, $y=\cos(x)$, por exemplo, é a matriz cujos elementos são $y(i,j)=\cos(x(i,j))$, ou seja, estas funções são aplicadas elemento a elemento.

11.– Funções especiais de SCILAB: Há um grande número de funções definidas em SCILAB, das quais destacamos apenas algumas:

“ceil(x)” e “floor(x)”=número inteiro que corresponde ao arredondamento do real x, para cima ou para baixo, respectivamente;
 “fix(x)”= arredondamento para o inteiro mais próximo de zero, ou seja, despreza a parte fracionária;
 “round(x)”=arredondamento para o inteiro mais próximo;
 “conj(x)”=complexo conjugado de x;
 “imag(x)”= parte imaginária da variável complexa x;
 “real(x)”= parte real de x;
 “sign(x)”= sinal(+ ou -) do real x;
 ”modulo(x,y)”= resto da divisão de x por y;

Se x e y forem matrizes as funções acima aplicam-se elemento a elemento.

12.- Funções matriciais: Há um grande número de funções matriciais, que podem ser encontradas no Manual ou no “help”. Apresentamos abaixo as mais importantes, para cada uma das quais o leitor pode encontrar mais detalhes e alternativas digitando help “função”:

“det(A)”: determinante de A;
 “spec(A)”: conjunto de autovalores de A;
 se escrevemos $[V,D] = \text{spec}(A)$, então V será a matriz cujas colunas são os autovetores de A e D será a matriz diagonal cujos elementos são os autovalores de A;
 “expm(A)”: exponencial matricial de A, i.e., e^A ; lembramos que $\exp(A)$ é a exponencial elemento a elemento;
 “logm(A)”: logaritmo matricial, função inversa de expm;
 “inv(A)”: matriz inversa de A;
 A' : conjugada hermitiana de A;
 “sum(A)”: soma dos elementos de A; “sum(A,1)”: vetor fila cujos elementos são a soma dos elementos das respectivas colunas de A;
 “sum(A,2)”: vetor coluna cujos elementos são a soma dos elementos das respectivas filas de A;
 “trace(A)”: traço, i.e., soma dos elementos da diagonal;
 “norm(V)”: norma do vetor V, i.e., $\sqrt{\sum(V.^2)}$;
 “sort(V)”: ordena os elementos do vetor V em ordem decrescente; para matriz V, ordena os maiores elemertos na primeira coluna, os seguintes na segunda, etc.

13.- As instruções “ponto e vírgula (;)” e “dois pontos (:): “ponto e vírgula (;)” é usado ao final de uma instrução, quando se quer evitar que o resultado da mesma seja escrito na tela; isto é muito importante em programas (veja “script file”, abaixo), quando não se quer mostrar todos os resultados intermediários; Por outro lado, usa-se o recurso de não colocar o “ponto e vírgula” em alguma variável que se quer que seja apresentada na tela durante a execução para informar que ponto do programa está sendo executado, especialmente em programas com longos tempos de CPU;
 “dois pontos” se usa para definir uma sequência, primeiro elemento : incremento : último elemento. Assim $x=1:3:10$ significa o vetor $x=[1, 4, 7, 10]$. O incremento pode ser negativo, como $x=10:-2:5$; significa $x=[10, 8, 6]$;

14.- Programas: Um conjunto de comandos e operações, salvas no arquivo “nome.sce” (a terminação “.sce” é convencional, mas não é obrigatória), são compiladas e executadas quando se entra “exec nome.sce” na linha de comando. SCILAB possui um editor conveniente para escrever e editar programas; basta clicar em “Editor” na janela SCILAB. É usual (e muito conveniente) escrever algumas linhas de comentários (tudo que vem na linha após o sinal //) no início de cada programa.

Para executar um programa comande “exec nome.sce” ou, alternativamente, “exec(‘nome.sce’,n)”, onde n é um dos algarismos que dizem o que deve ser escrito na tela (veja “help exec”).

Exemplo de programa: “seno.sce”

```
// calcula e plota a função sin(p1 x) entre 0 e p2,
// sendo p1 e p2 parâmetros a serem fornecidos pelo operador.
```

```
param=input('deh parametros [p1,p2] \n'); // Forma de entrar
// dados. O símbolo \n ordena o cursor a mudar de linha.
// Note também que o texto entre parênteses vem entre '...';
// este texto vai aparecer na tela e o operador deve fornecer
// os parâmetros na forma de vetor, ou seja [p1, p2]
p1=param(1); p2=param(2);
x=0:0.01:p2;
```

```
y=sin(p1*x);
plot(x,y);
```

Observação: As variáveis dos “programas” são globais, isto é, são reconhecidas, após execução, como variáveis do “workspace” e também reconhecem, em execução, as variáveis previamente definidas no “workspace”; por exemplo, após executar “seno.sce” escreva $x \leftarrow$ ou $p1 \leftarrow$.

15.– Controle de arquivos: Para ver a lista de arquivos do diretório atual, escreva “dir” \leftarrow ; “cd” \leftarrow recua para o diretório base do atual; “cd .” \leftarrow mostra o diretório atual; usa-se “save(‘arq’)” \leftarrow para salvar todas as variáveis do “workspace” no arquivo ‘arq’, em formato binário, ou “save(‘arq’, X, Y, Z)” \leftarrow para salvar as variáveis X Y e Z no arquivo ‘arq’; o comando “load(‘arq’)” \leftarrow carrega o conteúdo de ‘arq’ no “workspace”. Para excluir o arquivo ‘arq.x’, digite “mdelete arq.x”

16.– Construção de sequências: Mencionamos no item –**As instruções “ponto e vírgula (;)” e “dois pontos (:)**– que se pode construir uma sequência de valores igualmente espaçados usando a instrução “dois pontos (:); Existem outras maneiras. Muito útil é:

$x=linspace(x_{inicial}, x_{final}, n)$: n pontos igualmente espaçados, começando em $x_{inicial}$ e terminando em x_{final} ;

17.– Operações matriciais e elemento a elemento: Sejam X e Y duas matrizes. Soma: $X+Y$ só pode ser realizada se X e Y tiverem as mesmas dimensões; o resultado é a matriz soma dos elementos correspondentes; o mesmo vale para subtração (-). Multiplicação: $X*Y$ só pode ser realizada se as dimensões internas forem iguais, isto é, se X for $m \times n$ e Y for $n \times k$, resultando numa matriz $m \times k$; X/Y significa X^*Y^{-1} e $X\setminus Y$ significa $X^{-1}*Y$. Potenciação: $X^2=X*X$; esta operação só pode ser realizada com matrizes quadradas, mas $X*X'$ e $X'*X$ pode ser realizada com qualquer matriz retangular. Se c é um escalar, $X.^c$ significa a matriz em que cada elemento de X é

elevado a c. Se X e Y têm as mesmas dimensões, podem realizar-se as operações de multiplicação, divisão e potenciação elemento a elemento; para isto usa-se um “ponto” antes do sinal da operação: $Z = X.*Y$ resulta na matriz de elementos $Z(i,j) = X(i,j)*Y(i,j)$; Semelhantemente se usa $./$ e $.^$ para divisão e potenciação, respectivamente, elemento a elemento. Qualquer das operações acima, quando um dos operandos é um escalar, será realizada elemento a elemento com o mesmo escalar, mas não é conveniente escrever $1./X$ para inverter os elementos de X, que pode resultar também em transposição; neste caso é melhor escrever “ones(X)./ X ”. As funções elementares, como sin, cos, exp, log, abs, etc., quando aplicadas a matrizes, são sempre realizadas elemento a elemento.

18.- Referenciamento e manipulação de partes de matrizes:

Muitas vezes se necessita ler ou editar partes de uma matriz. Seja $A=[1,2,3; 4,5,6; 7,8,9]$. Usa-se o “dois pontos (:)” para referenciar um conjunto de elementos. Assim $A(:,2)$ significa o vetor coluna formado pelos elementos de todas as filas e segunda coluna, ou seja, pelos elementos 2, 5 e 8; $A(1:2,2:3)$ contém as duas primeiras filas com as duas últimas colunas, ou seja, a matriz [2,3; 5,6].

19.- Variáveis, relações e operações lógicas: variáveis lógicas assumem os valores *verdadeiro* e *falso*, representados, respectivamente, pelas letras T(true) e F(false); se x for um número e se solicitar seu valor lógico (por ex. em um *if*) então será F se x for 0 e será T se x for não nulo. Por outro lado, se uma variável lógica x for tratada como um número (por ex. em $1+x$) então x assume o valor 1 se $x=T$ e o valor 0 se $x=F$. Relações lógicas são *testes*, cujos resultados são variáveis lógicas (i.e., T ou F). Seja $a=2$ e $b=3$; a relação $a < b$ vale T e a relação $a > b$ vale F. Seja $x=(a < b)$; x uma variável lógica de valor T. Outros *operadores relacionais*, além de $<$ (menor) e $>$ (maior), são: $==$ (igual), $\sim=$ (diferente), \leq (menor ou igual) e \geq (maior ou igual).

Operações lógicas são realizadas por:
“ $A \& B$ ” (A e B): vale T se A e B forem verdadeiros (ou, no caso de A e B serem números, forem não nulos);

“A|B” (A ou B): vale T se pelo menos um entre A e B for verdadeiro;
 “~A” (não A): vale T se A for falso;

20.- Recorrências (loops): Para repetir N vezes um conjunto de operações usa se:

```
for n=1:N;
    conjunto de operações;
end.
```

Obs: A mudança de linha entre operações é opcional, podendo-se escrever várias operações, separadas por , ou ; na mesma linha.

Exemplo:

```
s=0; for n=1:500; x=rand(1,2); s=s+(x(1)<x(2)); end; s
```

O resultado deste “loop” deve ser um número próximo de 250.

No “for” acima a sequência de valores de n tem incremento 1; para incremento Δn , diferente de 1, se usa $n_{inicial} : \Delta n : n_{final}$. Por exemplo,
 “for n=200: -5: 0”.

Outra forma de “loop” que se repete “enquanto” uma condição está satisfeita :

```
while( condição )
    conjunto de operações;
end
```

Exemplo: fatorial de N

```
fat=1; N=input('deh inteiro N \n');
while(N>1);
    fat=fat*N; N=N-1;
end;
disp(fat, 'fatorial de N = ')
```

21.- Controle e decisão: Quando diferentes operações devem ser ou não realizadas, dependendo de condições, usa-se “if”, “elseif”,

“else”, na seguinte forma:

```

if(condição A);
    conjunto A de operações;
elseif(condição B);
    conjunto B de operações;
elseif(condição C);
    conjunto C de operações;
.
.
.
else
    operações alternativas;
end

```

Exemplo: “classifica.sce”

```

// gera N números aleatórios Gaussianos
// e conta quantos estão em cada um dos seguintes intervalos:
// a= (-∞, -1], b= (-1, 0], c= (0, 1], d= (1, ∞)
a=0; b=0; c=0; d=0;
N=input('deh numero N \n')
x=rand(1,N,'normal'); // gera N números aleatórios Gaussianos
for j=1:N;
    if(x(j) <=-1)
        a=a+1;
    elseif(x(j) <= 0) // note que a condição
        // x(j) >-1 já está satisfeita devido ao if anterior
        b=b+1;
    elseif(x(j) <= 1)
        c=c+1;
    else
        d=d+1;
    endif
end
disp([d c b a])

```

22.– Seleção Entre Opções: Uma forma alternativa para selecionar o procedimento a seguir, dependendo do valor de uma variável, que pode ser numérica ou uma "string" (palavra), quando o número de opções não é muito grande, é como segue:

Seja *var* a variável em questão e sejam *valor1*, *valor2*, *valor3*, ... seus valores possíveis; o procedimento de seleção é como segue:

```
"select var"
case valor1 then
    operações para o caso de var ter valor1
case valor2 then
    operações para o caso de var ter valor2
case valor3 then
    operações para o caso de var ser valor3
.
.
.
else
    operações alternativas
end
```

A instrução "else", para o caso de var não ter nenhum dos valores previstos, é opcional, podendo ser usada, por exemplo, para indicar que houve erro.

23.– Funções Programadas: O programador pode criar suas próprias funções em programas que diferem em alguns aspectos dos "programas" descritos no item 14. A primeira linha da função programada deve ser:

"function variável=nome_função(argumentos)".

A seguir podem ser colocadas linhas de comentários. A seguir, o conjunto de instruções para calcular "variável", que pode ser um escalar, vetor ou matriz ou mesmo vazia, "[]", se, por exemplo, a "function" apenas apresenta um gráfico. Termina-se a função com "endfunction". Num mesmo arquivo podem ser escritas várias funções. É conveniente dar a terminação ".sci" para os arquivos que contêm funções. Uma prática interessante é fazer um arquivo para cada função, de-

nominando este arquivo de “nome_função.sci”.

Para usar qualquer função do arquivo deve-se, anteriormente, comandar “exec nome_arquivo.sci”, ou “getf nome_arquivo.sci”, que compilam as funções do arquivo e as disponibilizam para uso.

Exemplo:

```
function G=histo(r,n);
// O argumento r é um vetor de N números reais e n é o número //
de classes em que pretendo subdividir as componentes de r.
// “histo” irá contar quantos elementos de r pertencem cada uma
// das classes; como um histograma, mas sem desenhar o gráfico.
N=length(r);
mi=min(r); ma=max(r);
D=(ma-mi)/n; d=D/2;
X=mi+d:D:ma-d;
R=ceil((r-mi)*n/(ma-mi));
k=find(R==0); R(k)=1;
H=zeros(1,n);
for j=1:N
    J=R(j);
    H(J)=H(J)+1;
end
G=[X;H];
endfunction
```

Após digitado e salvo o arquivo acima, com o nome “histo.sci”, digite no prompt de SCILAB “getf histo.sci”; agora crie um vetor com um milhão de números aleatórios Gaussianos digitando “r=rand(1,1.e6,’normal’); a seguir digite F=histo(r,100); a seguir comande “plot(F(:,1),F(:,2))” e veja a bela gaussiana de números aleatórios que foi produzida.

Observações: 1) As variáveis das funções são “locais”, isto é, não são reconhecidas no “workspace” ou pelas demais funções do arquivo; entretanto, quando o programa da função encontra um nome de variável que não lhe foi passada nem nela definida, mas uma variável de mesmo nome se encontra no workspace, ela usa esta, o que pode resultar em erro.

2) Há um grande número de funções definidas em SCILAB à disposição do usuário, como, por exemplo, “roots”, que encontra as raízes de polinômios. Uma lista completa delas pode ser encontrada digitando “help function”.

24.– Gráficos em duas dimensões; SCILAB possui excelentes recursos gráficos. Para complementar as informações apresentadas abaixo, veja “help plot”, “help plot2d”, “help polar”. O mais usado dos comandos de gráficos é “plot”, que foi importado de MATLAB com sintaxe quase idêntica; “plot” é usado para traçar curvas em duas dimensões, podendo traçar qualquer número de curvas, com diferentes formas de linha e cores, no mesmo gráfico. A sintaxe é:

`plot(x1,y1,'s1',x2,y2,'s2',...)`

onde x_j são vetores cujas componentes definem os pontos sobre a abscissa, y_j o mesmo para a ordenada e s_j (chamado “LineSpec”) é a forma e cor da linha e de marcadores (para possíveis formas, cores, e marcadores comande “help LineSpec”). Os s_j são opcionais, a forma “default” é a linha cheia, azul, sem marcador. Os vetores x_j e y_j devem ter a mesma dimensão, que pode ser diferente da dimensão de x_k e y_k . Não deve ser deixado espaço entre os símbolos de cor, linha e marcador, mas todos são opcionais.

Um comando gráfico específico de SCILAB é “plot2d(X,Y)”. Neste caso X e Y são matrizes $m \times n$. Este comando traça n linhas com m pontos cada, sendo cada coluna de X a abscissa para a correspondente coluna de Y. No caso de as abscissas serem todas iguais pode-se escrever “plot2d(x,Y)”, sendo x um vetor coluna, que serve de abscissa para todas as colunas de Y.

25.– Histograma Outro gráfico bi-dimensional muito usado é “histplot(n,r)”, onde r é um vetor de números reais e n é o número de classes em que se pretende dividir as componentes de r. “histplot(n,r)” conta quantas componentes de r pertencem a cada uma das n classes e traça o gráfico de barras (histograma) correspondente. Exemplo: `r=rand(1,1.e6,'normal');` `histplot(50,r);` o operador pode escolher o intervalo de valores de cada classe, pelo seguinte procedimento: define o vetor $x = [x_1, x_2, \dots, x_n]$; escrevendo “histplot(x,r)” as barras do histograma corresponderão ao número de componentes

de r com valor em cada intervalo das componentes de x, ou seja, a primeira classe corresponde aos valores entre x1 e x2, a segunda aos valores entre x2 e x3, etc.; para se obterem os valores do histograma, isto é, os números de componentes de r em cada classe, sem traçar o histograma, existe a função dsearch, usada assim:

[cls,h]=dsearch(x,r);

o vetor cls, com o mesmo número de componentes que r, diz a que classe pertence cada componente de r; o vetor h, com "length(x)-1" componentes, diz quantas componentes de r pertencem a cada classe;

26.-Gráficos em três dimensões; Há uma variedade de possibilidades para gráficos em 3 dimensões. Aqui apresentamos apenas a mais simples. Para outras possibilidades veja "help surf", "help mesh", "help plot3d".

"surf(x,y,Z)", onde x e y são vetores reais de dimensões n e m, respectivamente, e Z uma matriz real m × n, faz um gráfico tridimensional da superfície Z sobre as coordenadas x e y, usando cores "default". "mesh(x,y,Z)" faz o mesmo gráfico em preto e branco. Note que a primeira coordenada (x) tem a mesma dimensão que a segunda dimensão de Z e a segunda coordenada (y) tem a mesma dimensão que a primeira de Z .

Exemplo: x=linspace(0,%pi,101); y=-1:0.1:1;
 $Z=(y.^*y)^*\sin(x)$; surf(x,y,Z); Nota: em "tools", "2D/3D rotation" pode-se girar o gráfico.

27.-Memória RAND disponível. A memória RAND disponibilizada por "default" para uso por SCILAB depende do sistema, podendo ser menor do que a que o programa necessita. Neste caso haverá uma mensagem na forma "stacksize exceeded". Use o comando "stacksize()" para ser informado sobre o tamanho da memória disponível e o quanto desta está usado. Para aumentar a memória disponível para n reais de 8 bytes comande "stacksize(n)".

28.-**Tempo de CPU:** Uma das características importantes do bom programador é usar os recursos que agilizam a execução do programa. Há várias maneiras de se saber quanto tempo de CPU foi gasto em partes escolhidas do programa. A mais simples delas consiste em introduzir:

```
tic;  
conjunto de comandos;  
toc
```

O tempo de CPU gasto pelo conjunto de comandos será registrado na tela. Note que a primeira vez, em uma seção SCILAB, em que uma rotina é chamada há uma demora um pouco maior do que quando a mesma rotina for usada em vezes subsequentes. Por isso é aconselhável sempre repetir a medida de tempo com tic-toc.

Uma sugestão para a eficiência de programação e execução: Sempre que possível execute as operações em forma vetorial ou matricial, em vez de executá-las explicitamente com os elementos dos vetores ou matrizes; em outras palavras, sempre que possível evite a recorrência "for". Por exemplo, se x e y são vetores fila de N componentes, a operação $s=x*y'$ é ordens de grandeza mais rápida que a forma equivalente,

```
s=0;for j=1:N; s=s+x(j)*y(j); end.
```

Faça o teste com $x=rand(1,100000)$; $y=rand(1,100000)$; use tic-toc.

Apêndice B: TRADUÇÃO DE SCILAB A MATLAB

Os exemplos de programas que constam deste livro estão escritos para serem executados em SCILAB. Para o leitor que preferir usar a linguagem MATLAB, apresentamos aqui um conjunto de sugestões de modificações dos programas para executá-los com MATLAB. A grande maioria dos comandos são iguais para essas duas linguagens. As diferenças que ocorrem mais frequentemente são apresentadas a seguir:

SCILAB	MATLAB	SIGNIFICADO
exec "nomeprog.sce"	"nomeprog"	chama e executa "nomeprog";
//	%	comentário
sum(A,1)	sum(A)	soma cada coluna de A;
sum(A)	sum(sum(A))	soma todos elementos de A;
ones(A)./A	1./A	divisão elemento a elemento;
rand()	rand	gera um número aleatório uniforme em [0,1];
rand(n,n)	rand(n)	gera matriz n × n de números aleatórios uniformes em [0,1];
rand(n,m,'normal')	randn(n,m)	gera matriz n × m aleatória normal;
select	switch	seleciona uma das opções;
xset('window',n)	figure(n)	abre janela gráfica nº n;
clf	"automático"	apaga gráfico existente antes de traçar um novo;

Referências Bibliográficas

- [1] Márcia A. Gomes Ruggiero e Vera Lúcia da Rocha Lopes, **Cálculo Numérico, Aspectos Teóricos e Computacionais**, 2^a edição, Makron Books do Brasil Editora Ltda, São Paulo, 1997;
- [2] John H. Mathews, **Numerical Methods for Computer Science, Engineering and Mathematics**, Prentice-Hall Inc., New Jersey, 1987;
- [3] G. J. Borse, **Numerical Methods with MATLAB**, PWS Publishing Company, Boston, MA, 1997;
- [4] Paul L. DeVries, **A First Course in Computational Physics**, John Wiley & Sons, New York, 1994;
- [5] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, **Numerical Recipes, the Art of Scientific Computing**, Cambridge University Press, New York, 1992 (Há versões em FORTRAN, C e PASCAL)
- [6] Alejandro L. Garcia, **Numerical Methods for Physics**, Prentice Hall, New Jersey, 1994.
- [7] J.W.Cooley and J.W.Tukey, Math. Comput. 19, 297 (1965);
- [8] W.Feller, **An Introduction to Probability Theory and its Applications**, 2^a edição, Vol.II, New York, 1974.
- [9] Giuseppe Milone, **Estatística Geral e Aplicada**, Pioneira Thomson Learning, São Paulo, 2004;

- [10] C.W.Gardiner, **Handbook of Stochastic Methods for Physics, Chemistry and the Natural Sciences**, Springer-Verlag, Berlin, 1985.
- [11] R.Car e M.Parrinello, **Phys. Rev. Lett.** **55**, 2471 (1985);
- [12] Kaoru Ohno, Keivan Esfarjani e Yoshiyuki Kawazoe, **Computational Materials Science**, vol 129 de “Springer Series in Solid-State Sciences”, Springer-Verlag, Berlin, 1999;
- [13] Joseph Marie Thijssen, **Computational Physics** Cambridge University Press, Cambridge, 1999;
- [14] Daan Frenkel e Berend Smit, **Understanding Molecular Simulation**, Academic Press, San Diego, 1996;
- [15] M.P.Allen e D.J.Tildesley, **Computer Simulation of Liquids** Oxford University Press, Oxford, 1989;
- [16] L.E.Reichl, **A Modern Course in Statistical Physics**, Edward Arnold (Publishers) Ltd, 1980;
- [17] M.Toda, R.Kubo, N.Saitô, **Statistical Physics I: Equilibrium Statistical Mechanics**, Springer Verlag 1983;
- [18] R.Kubo, M.Toda, Hashitsume, **Statistical Physics II: Non-Equilibrium Statistical Mechanics**, Springer Verlag 1983;
- [19] C. Garrod, **Statistical Mechanics and Thermodynamics**, Oxford University Press, 1995;
- [20] S.R.A.Salinas, **Introdução à Física Estatística**, EDUSP, São Paulo, 1997;
- [21] M.E.J.Newmann e G.T.Barkema, **Monte Carlo Methods in Statistical Physics**, Clarendon Press, Oxford, 1999;
- [22] N.Metropolis, A.W.Rosenbluth, M.N.Rosenbluth, A.H.Teller e E.Teller, **J.Chem.Phys.** **21**, 1087 (1953);
- [23] J.P.Valleau e D.N.Card, **J.Chem.Phys.** **57**, 5457 (1972);

- [24] A.M.Ferrenberg e R.H.Swendsen, *Phys.Rev.Lett.* **61**, 2635 (1988);
- [25] R.M.White, **Quantum Theory of Magnetism**, Springer Series in Solid-State Sciences, vol.32, Springer-Verlag, Berlin, Heidelberg, New York, 1983.
- [26] P.M.C. de Oliveira, T.J.P.Penna e H.J.Herrmann, *Braz.J.Phys.* **26**, 677 (1996).
- [27] L.Arnold, **Stochastic Differential Equations: Theory and Applications**, Willey, New York, 1974.
- [28] R.L.Stratonovich, **Introduction to the Theory of Random Noise**, Gordon and Breach, New York, 1963.
- [29] T.F.Ricci, **Um Modelo Estocástico para a Dinâmica do Momento Magnético de Partículas Superparamagnéticas**, Tese de Doutorado, defendida no Instituto de Física da UFRGS, Porto Alegre, 1996.

ÍNDICE

- ajuda (help) em SCILAB, 271
- ajuste, 25-33
- ajuste linear, 27
- ajuste a uma curva qualquer, 29
- algoritmo "cinco meios", Eq. Langevin, 249
- arquivos, controle em SCILAB, 277
- aproximações numéricas de funções, 16
- balanço detalhado, princípio, 189
- Boltzmann, distribuição de, 187
- cálculo de Ito, 229
- cálculo de Stratonovich, 240
- calor específico, 188
- células em DM, 159
- células enumeradas, Monte Carlo, 210
- células vizinhas, identificação, 161
- clear, em SCILAB, 273
- coeficiente de difusão dependente de posição, 90
- condições de contorno, 88
- condições de contorno periódicas, 171
- conjunto completo de eventos, 120
- contínuos, processos estocásticos 139
- controle de arquivos, em SCILAB, 277
- controle e decisão em SCILAB, 300
- convergência em média quadrática, 229
- convolução, 41
- correlação, 42, 124, 136
- correlação, tempo de, 137
- correspondência SCILAB-MATLAB, 287
- covariância, 123
- Cramer, regra de, 10
- Crank-Nicholson, procedimento, 97-100
- cubico.sci, "function em SCILAB", 23
- defeitos por Crank-Nicholson, 101
- defeitos por radiação, 92
- delta de Dirac, 43
- densidade de probabilidade, 125

- densidade espectral, 140
- deriva, equação de, 91, 99
- derivada, 5, 6
 - derivada centrada, 65
 - derivada de Processo estocástico, 222
 - derivada numérica, 5, 61
 - derivada segunda, 6
- diferenças finitas, 5
 - diferença finita de primeira ordem, 5
 - diferença finita de segunda ordem, 6
- dimensões de matrizes, em SCILAB, 274
- diferencial de Ito, 235
- dinâmica estocástica, 219
- dinâmica molecular, 145
 - dinâmica molecular, abordagem nível 0, 148
 - dinâmica molecular, abordagem nível 1, 157
 - dinâmica molecular, abordagem nível 2, 171
- Dirac, delta de, 43
- discretos, processos estocásticos, 135-139
- disjuntos, eventos, 120
- distribuição angular de dipolo, 130
- distribuição canônica, 187
- distribuição, função, 125
- distribuição Gaussiana, 127
- distribuição uniforme, 126
- distribuições especiais, 126
- distribuições consagradas, 128
- distribuição lognormal, 131
- dois pontos em SCILAB, 276
- equação diferencial estocástica de Ito, 235
- equação diferencial estocástica de Stratonovich, 242
- Einstein, relação de, 262
- elementares, eventos, 120
- eliminação de Gauss, 10
- energia interna, 188
- ensemble, 184
- equação de deriva, 91

equação da difusão, 88
equação de Fokker-Planck, 259
equação de Fokker-Planck vetorial, 263
equação de Langevin, 239
equação de Langevin até ordem 5/2, 249
equação master, 141
equações diferenciais de primeira ordem, 61
equações diferenciais de segunda ordem, 67
equações diferenciais estocásticas (EDE), 224, 228
equações diferenciais ordinárias (EDO), 61
equações diferenciais parciais (EDP), 87
equação diferencial estocástica de Ito, 235
equação diferencial estocástica de Stratonovich, 242
ergódico, sistema, 192
erro na derivação numérica, 64
erro, local ou global, 65
espaço de fases, 183
estabilidade numérica, condição de, 89, 92
estacionária, solução, 95
estacionário, processo estocástico, 137
estocásticos, processos, 135
Euler, método de, 62
eventos independentes, 122
explícito, procedimento, 97
Fokker-Planck, equação de, 259
Fourier, transformada de, 39
flutuação-dissipação, teorema da, 262
forças dependentes de velocidade, 70, 77
forças, cálculo em DM, 165
formatos das variáveis na tela em SCILAB, 272
FTCS, método, 88
função partição, 188
função resposta, 266
funções elementares em SCILAB, 274
funções especiais de SCILAB, 274
funções matriciais em SCILAB, 275
funções vetoriais não lineares, 15

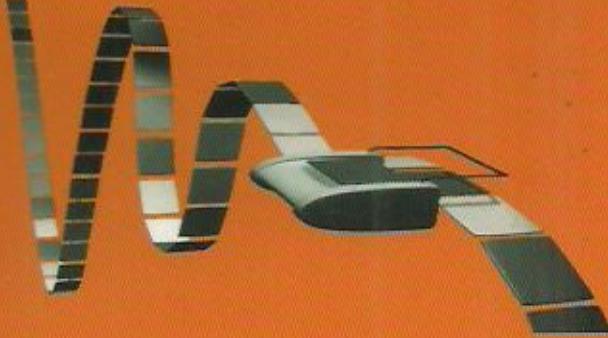
"function"(função programada) em SCILAB, 281
gás bi-dimensional, modelo, 147
gás bi-dimensional, Monte Carlo, 205
Gauss, eliminação de, 10
Gaussiana, 33
Gaussiano, processo estocástico, 140
Gauss-Seidel, algoritmo, 115
gráficos em SCILAB, 283-284
Heisenberg, Hamiltoniano de, 211
histograma, em SCILAB, 283
histograma, método do, 216
Heun, algoritmo de, 225, 240
implícito, procedimento, 97
incremento de Wiener, 221
incrementos adaptativos, 80
independentes, eventos, 122
inicialização de DM, 150, 160
integração em DM, 154, 168
integração numérica, 34-38
integrais impróprias, 37
integral de Ito, 230
integral de Stratonovich, 240
intensidade espectral, 139
interpolação, 17
interpolação por segmentos, 20
interpolação linear, 20
Ising, modelo de, 196
iteração, 2
Ito, cálculo de, 229
Jacobi, algoritmo, 111, 113
Lagrange, polinômio interpolador de, 17, 18
Langevin, equação de, 222
largura de distribuição, 127
Lax, método de, 92
leap-frog, algoritmo, 73
Lennard-Jones, potencial de, 151
limite em média quadrática, 229

listas de átomos em DM, 159, 164, em MC 230
Lorentziana, 26
lorentz.sci, "function em SCILAB", 31
macro-estado, 185
magnetização no modelo de Ising, 199
manipulação de matrizes, em SCILAB, 278
Markoviano, processo estocástico, 138
master, equação, 141
matrizes especiais em SCILAB, 274
matriz Jacobiana, 15
matriz tridiagonal, 13
mecânica estatística, 181
mecânica estatística, objeto da, 182
mecânica estatística, conceitos fundamentais, 183
média, 122
média de ensemble, 184
média por amostragem, 190
média por amostragem tendenciosa, 191
média sobre realizações, 136
memória disponível, para SCILAB, 284
método de Euler 62
método FTCS, 88
Metropolis, algoritmo de, 192
micro-estado, 183
momentos, 123
monitoração da precisão pela energia, 80
Monte Carlo, método, 132, 181
movimento Browniano, 220
movimento Browniano em coordenadas polares, 255
Newton, método Generalizado, 16
Newton-Raphson, método de, 3
números aleatórios, geração de, 129, 132
objeto da Mecânica Estatística, 182
operações em SCILAB, 272, 277
operações lógicas em SCILAB, 278
órbita de cometa, 81
Ornstein-Uhlenbeck, processo de, 261

passo de Monte Carlo, 198
pêndulo rígido, 69
pivô, linha, elemento, 11
poço duplo com ruído aditivo, 226
Poisson, distribuição de, 142
Poisson, processo de, 141
polinômio interpolador de Lagrange, 18
ponto e vírgula em SCILAB, 296
pontos fixos, 2
porque a mecânica estatística funciona, 186
precisão, teste de, 233
primeiros princípios, dinâmica molecular, 145
probabilidade, conceito de, 119
probabilidade condicional, 121
probabilidade de transição, 138, 190
procedimentos explícito, implícito e Crank-Nicholson, 97
processos estocásticos, 135-143
programa de dinâmica molecular, 146
programa em SCILAB, 276
programa principal de DM, 146, 148, 158, 172
quadrado, 26
raiz dupla, 8
raiz múltipla, 8
raiz.sci, "function" em SCILAB, 6
raízes de funções, 1, 15, 16
raízes de polinômios, 8
raízes de sistemas lineares, 9
raízes de sistemas não-lineares, 15, 16
raízes vetoriais, 9
realização de processo estocástico, 136
recorrências em SCILAB, 279
referenciamento a partes de matriz em SCILAB, 278
reflexão nas paredes, em DM, 155
regra de Simpson, 37
regra do trapézio, 34
regressão linear, 27
rejeição, método, 132

relação de Einstein, 262
relações diferenciais de Ito, 232
relaxação, método de, 95
relaxação em várias dimensões, 110
repositionamento atômico, condições periódicas, 171
resposta linear, teoria para PE, 265
roots, rotina SCILAB, 9
ruído aditivo, 223, 224
ruído branco, 219
ruído multiplicativo, 223, 228, 249
SCILAB, linguagem de programação, 271
Runge-Kutta, métodos, 75
Runge-Kutta, 2^a ordem, 75
Runge-Kutta 4^a ordem, 76
Schrödinger, equação, 105
SCILAB, 271
seleção entre opções, em SCILAB, 281
sequências em SCILAB, 277
Simpson, regra de, 37
simulação numérica de PE discreto, 142
simulações numéricas, Ito e Stratonovich, 246
sistema de equações diferenciais, 65
sistemas lineares, 9
smooth, "function" de SCILAB, 26, 27
solução estacionária de EDP, 95-97
spline cúbico, 21-25
splines, 21
Stratonovich, cálculo de, 240
subdiagonais, 13
super-relaxação, algoritmo, 116
susceptibilidade magnética, 199
susceptibilidade dinâmica, 268
temperatura crítica, 198
tempo de correlação, 137
tempo de CPU, em SCILAB, 285
teorema central de limite, 127
teorema de Perseval, 42

teoria da resposta linear, 265
tfd.sci, "function" em SCILAB, 46
tfr.sci, "function" em SCILAB, 55
Thomas, algoritmo de, 97
tipos de variáveis em SCILAB, 273
transformação de coordenadas por Ito, 256
transformação de coordenadas por Stratonovich, 257
transformada de Fourier, 39-59
transformada de Fourier discreta, 44
transformada de Fourier inversa, 39
transformada de Fourier-Laplace, 266
transformada de Fourier, propriedades, 40-42
transformada de Fourier rápida, 50-59
transição, probabilidade de, 138, 190
variância, 123
variância do momento magnético, 199
variáveis dinâmicas, cálculo em DM, 156, 169
variáveis especiais, em SCILAB, 273
variáveis lógicas, em SCILAB, 278
variável aleatória, 122
variável aleatória contínua, 124
vários ruídos, cálculo estocástico, 244
Velocity-Verlet, algoritmo, 74
Verlet, algoritmo de, 68
vetor girante, 237, 243, 264
who, whos, em SCILAB, 293
Wiener-Khintchine, teorema, 140
Wiener, processo de, 219
vizinhas, células, 161
workspace em SCILAB, 272



Este livro foi produzido a partir das notas de aula da disciplina "Métodos Computacionais da Física II", hoje denominada "Métodos Computacionais C" lecionadas durante vários semestres para alunos do curso de Bacharelado em Física da Universidade Federal do Rio Grande do Sul.

O livro inicia com uma introdução ao cálculo numérico por computador, com uma seleção de assuntos de especial interesse para os Físicos, mas também para outros profissionais que usam métodos numéricos.

A seguir, apresentam-se os algoritmos mais usados na solução de Equações Diferenciais, ordinárias e parciais, uma introdução à Teoria de Probabilidade, Dinâmica Molecular, Método de Monte Carlo e Dinâmica Estocástica. Esta seleção de assuntos é arbitrária, mas, no entender do autor, engloba os métodos computacionais numéricos mais usados pelos físicos.

Ele se destina a um curso de um ou dois semestres, dependendo da seleção de assuntos. Se houver interesse em aprofundar algum dos assuntos tratados mais do que o faz o livro, as referências bibliográficas sugerem bons textos a serem usados.

Os exemplos de programas ao longo do livro são apresentados em linguagem SCILAB, a qual é explicada em apêndice, mas os métodos apresentados podem, evidentemente, ser aplicados com qualquer outra linguagem de programação numérica, como FORTRAN, C, C++ ou PASCAL. A ênfase do livro é a na clareza da apresentação, com intenção exclusivamente didática, exceto no capítulo sobre Dinâmica Estocástica, onde contribuições recentes do autor também são apresentadas, não abandonando, entretanto, a preocupação didática.

