



# INTRODUÇÃO À FÍSICA COMPUTACIONAL

**RUDI GAELZER (INSTITUTO DE FÍSICA - UFRGS)**

Apostila preparada para as disciplinas de Métodos Computacionais da Física, ministradas para os Cursos de Bacharelado em Física e Engenharia Física do Instituto de Física da Universidade Federal do Rio Grande do Sul, Porto Alegre - RS.



Apostila escrita usando:

**PROCESSADOR DE DOCUMENTOS  $\text{L}\text{\kern-0.1em}\text{\kern-0.1em}\text{X}$**

<http://www.lyx.org/>

<http://wiki.lyx.org/LyX/LyX>

# SUMÁRIO

<b>1</b>	<b>Representação de Números e Erros</b>	<b>1</b>
1.1	Representação de números em diferentes bases . . . . .	1
1.1.1	Representação de números inteiros e conversões de base . . . . .	1
1.1.2	Representação de números reais e conversões de base . . . . .	4
1.1.3	Conversão de números inteiros da base $b$ para a base decimal . . . . .	5
1.1.3.1	Algoritmo de Horner. . . . .	5
1.1.3.2	Divisão de Ruffini. . . . .	5
1.1.4	Conversão de números fracionários da base $b$ para a base decimal . . . . .	6
1.1.4.1	Algoritmo de Horner. . . . .	6
1.1.4.2	Divisão de Ruffini. . . . .	6
1.2	Operações com números binários . . . . .	8
1.2.1	Adição binária . . . . .	8
1.2.2	Subtração binária . . . . .	8
1.2.3	Multiplicação binária . . . . .	9
1.3	Representação de números em computadores digitais . . . . .	9
1.3.1	Representação de números inteiros . . . . .	9
1.3.2	Representação de números reais . . . . .	10
1.4	Fontes de erros e incertezas . . . . .	13
1.5	Erros na representação e na álgebra de ponto flutuante . . . . .	13
1.5.1	Precisão e acurácia . . . . .	13
1.5.2	Erros absoluto e relativo . . . . .	14
1.5.2.1	Erro absoluto . . . . .	14
1.5.2.2	Erro relativo . . . . .	15
1.5.3	Erros na representação: arredondamento e truncamento . . . . .	15
1.5.4	Número de dígitos significativos corretos . . . . .	17
1.5.5	Erros na álgebra de ponto flutuante . . . . .	17
1.5.5.1	Erros de arredondamento . . . . .	17
1.5.5.2	Erros de truncamento . . . . .	22
1.5.5.3	Análise de erros de ponto flutuante . . . . .	26
<b>2</b>	<b>Cálculo de Funções</b>	<b>27</b>
2.1	Casos básicos . . . . .	27
2.1.1	Equação quadrática . . . . .	27
2.1.2	Equação cúbica . . . . .	29
2.2	Polinômios e funções racionais . . . . .	29
2.3	Séries convergentes . . . . .	32
2.3.1	Séries de Taylor e de potências . . . . .	32
2.3.2	Aceleração da convergência de séries . . . . .	37
2.3.2.1	Transformação de Euler . . . . .	37
2.3.2.2	O método $\Delta^2$ de Aitken . . . . .	37
2.4	Relações de recorrência . . . . .	38
2.4.1	O método de Miller para a solução mínima . . . . .	40
2.5	Séries de Fourier-Chebyshev . . . . .	42
2.6	Frações continuadas . . . . .	44
2.6.1	Frações continuadas a partir de relações de recorrências ou séries de potências . . . . .	44

2.6.2	Método de Viskovakov . . . . .	46
2.6.3	Transformações de equivalência e contrações . . . . .	46
2.6.4	Computação de frações continuadas . . . . .	47
<b>3</b>	<b>Derivação Numérica</b>	<b>49</b>
3.1	Introdução . . . . .	49
3.2	Fórmulas clássicas de diferença finita . . . . .	49
3.2.1	Fórmula de diferença adiantada ( <i>forward difference</i> ) . . . . .	49
3.2.2	Fórmula de diferença atrasada ( <i>backward difference</i> ) . . . . .	50
3.2.3	Fórmula de diferença centrada ( <i>centered difference</i> ) . . . . .	51
3.2.4	Fórmula de 5 pontos . . . . .	51
3.3	Fórmulas de diferenças finitas para a derivada segunda . . . . .	52
3.3.1	Fórmula de três pontos . . . . .	52
3.3.2	Fórmula de cinco pontos . . . . .	52
3.4	Fórmulas para o cálculo de derivadas em pontos fora da rede . . . . .	52
3.4.1	Derivada de três pontos . . . . .	53
3.4.2	Derivada de quatro pontos . . . . .	53
3.4.3	Derivada de cinco pontos . . . . .	53
3.5	Extrapolação de Richardson e estimativa de erro . . . . .	53
<b>4</b>	<b>Integração Numérica</b>	<b>61</b>
4.1	Introdução . . . . .	61
4.2	Fórmulas de Newton-Cotes . . . . .	62
4.2.1	Fórmulas fechadas de Newton-Cotes . . . . .	62
4.2.1.1	Regra trapezoidal ( $N = 1$ ) . . . . .	64
4.2.1.2	Regra de Simpson ( $N = 2$ ) . . . . .	65
4.2.1.3	Regra de Simpson dos 3/8 ( $N = 3$ ) . . . . .	66
4.2.1.4	Regra de Bode ( $N = 4$ ) . . . . .	66
4.2.1.5	Regras em ordens mais altas ( $N \geq 5$ ) . . . . .	66
4.2.2	Fórmulas abertas de Newton-Cotes . . . . .	67
4.2.3	Fórmulas fechadas estendidas . . . . .	67
4.2.3.1	Regra trapezoidal estendida . . . . .	68
4.2.3.2	Regra de Simpson estendida . . . . .	68
4.2.4	Fórmulas abertas estendidas . . . . .	69
4.2.5	Estimativas de erro nas fórmulas de Newton-Cotes . . . . .	70
4.3	Quadratura gaussiana . . . . .	71
4.3.1	Idéia básica na quadratura gaussiana . . . . .	72
4.3.2	Fórmulas gaussianas clássicas . . . . .	73
4.3.2.1	Fórmula de Gauss-Legendre . . . . .	74
4.3.2.2	Fórmula de Gauss-Chebyshev . . . . .	76
4.3.2.3	Fórmula de Gauss-Laguerre . . . . .	78
4.3.2.4	Fórmula de Gauss-Hermite . . . . .	78
4.4	Integração automática e adaptativa . . . . .	79
4.4.1	Integração de Romberg . . . . .	80
4.4.1.1	Integrais definidas de Romberg . . . . .	80
4.4.1.2	Integrais impróprias de Romberg . . . . .	86
4.4.2	Integração automática usando quadraturas gaussianas . . . . .	94
4.4.3	Integração adaptativa . . . . .	94
<b>5</b>	<b>Soluções de Equações Não Lineares</b>	<b>97</b>
5.1	Introdução . . . . .	97
5.2	Métodos iterativos para o cálculo de raízes reais . . . . .	97
5.2.1	Método da biseção . . . . .	98
5.2.2	Método da falsa posição . . . . .	101
5.2.3	Método da secante . . . . .	104
5.2.4	Método de Newton-Raphson . . . . .	107
5.3	Raízes complexas de funções analíticas . . . . .	110
5.3.1	O método de Müller . . . . .	111
<b>6</b>	<b>Interpolação, Extrapolação e Ajuste de Funções</b>	<b>119</b>

<b>7 Problemas de Valor Inicial</b>	<b>121</b>
7.1 Introdução . . . . .	121
7.2 Equações de diferenças finitas lineares . . . . .	121
7.3 Integração numérica por série de Taylor . . . . .	123
7.3.1 O método de Euler . . . . .	124
7.4 O Método de Runge-Kutta . . . . .	124
7.4.1 O Método de Runge-Kutta de segunda ordem ou o Método do ponto médio . . . . .	125
7.4.2 O Método de Runge-Kutta de quarta ordem . . . . .	125
7.5 Sistemas de equações diferenciais . . . . .	126
<b>8 Outros Métodos</b>	<b>131</b>
<b>Referências Bibliográficas</b>	<b>133</b>



# REPRESENTAÇÃO DE NÚMEROS E ERROS

*There are 10 kinds of people in the world: Those who understand binary and those who don't...*

Neste capítulo serão considerados aspectos básicos a respeito do cálculo numérico: a representação de números inteiros e de ponto flutuante em código binário e as fontes de erros que invariavelmente ocorrem quando se faz necessário usar uma representação finita para representar um número ou uma função matemática que, em geral é transcendental e/ou necessita de uma soma ou produto infinito de números para ser exatamente representado.

## 1.1 REPRESENTAÇÃO DE NÚMEROS EM DIFERENTES BASES

Nesta seção serão discutidos alguns métodos para a mudança de base na representação de números, tanto inteiros quanto reais. É fato comum para grande parte dos computadores atualmente empregados na modelagem computacional o emprego de uma base numérica distinta da base decimal, à qual os seres humanos tendem a se apegar. Em geral, os números são armazenados na base 2 (binária), existindo ainda plataformas que os armazenam na base 8 (octal) ou na base 16 (hexadecimal). A representação de números inteiros é ligeiramente distinta da representação de números reais.

### 1.1.1 REPRESENTAÇÃO DE NÚMEROS INTEIROS E CONVERSÕES DE BASE

De uma forma geral, um número inteiro  $N$  é representado, na base  $b$ , por um conjunto de dígitos  $a_i$ , ( $i = 0, 1, 2, \dots$ ), sendo que  $a_i$  assume um intervalo de valores determinado pela base em uso. A tabela 1.1 indica estes valores para as bases mais utilizadas, inclusive para a base decimal.

Há no mínimo duas maneiras de se representar o número  $N$ . O **sistema posicional** agrupa os dígitos na forma de uma sequência, na qual a magnitude da contribuição de cada dígito ao número depende da posição relativa que este ocupa. Neste sistema, o número  $N$  é escrito como:

$$N = (a_n a_{n-1} \dots a_1 a_0)_b.$$

**Tabela 1.1:** Intervalos de valores para os dígitos  $a_i$  da base  $b$ .

$b$	$a_i$
2	0, 1
8	0, 1, 2, 3, 4, 5, 6, 7
10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

A contribuição de cada dígito para o valor de  $N$  fica explicitada na **forma polinomial**, onde  $N$  é escrito como:

$$N = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b + a_0. \quad (1.1)$$

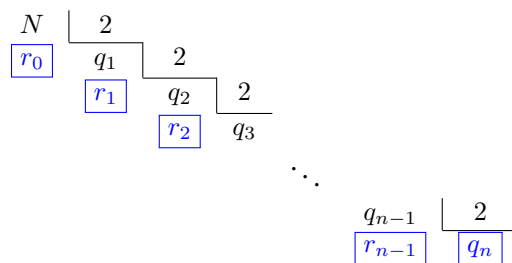
Até este momento,  $N$  tem sido tratado de uma forma abstrata. Por uma questão evolutiva,  $N$  tende a ser visto como um número na base 10 (decimal),

$$N = (a_n a_{n-1} \dots a_1 a_0)_{10} \equiv a_n a_{n-1} \dots a_1 a_0.$$

Caso se passe a representar  $N$  sempre na base decimal, então deve-se abordar as outras representações do ponto de vista de conversões de ou para a base 10.

## MÉTODO DAS DIVISÕES SUCESSIVAS

Considera-se inicialmente a conversão de um inteiro da base decimal para a base binária, uma vez que esta será a representação mais provável em um computador. Para realizar-se esta conversão de uma maneira prática, pode-se usar o método das divisões sucessivas, no qual  $N$  e os sucessivos quocientes  $q_i$  são divididos por 2, sendo coletados os restos  $r_i = 0, 1$  até que o último quociente seja  $q_n = 0, 1$ :



O último quociente ( $q_n$ ) somente será 0 se  $N = 0$ . Então,

$$N = (q_n r_{n-1} \dots r_2 r_1 r_0)_2$$

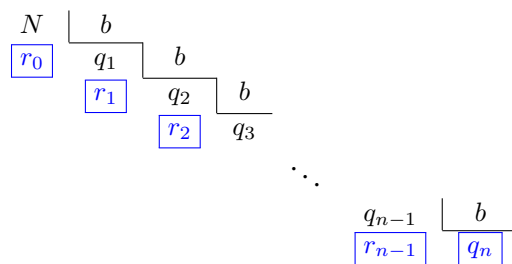
ou

$$N = q_n 2^n + r_{n-1} 2^{n-1} + \dots + r_2 2^2 + r_1 2^1 + r_0 2^0.$$

Como exemplos, temos

$$\begin{aligned} 12 &= (1100)_2 \\ 25 &= (11001)_2 \\ 315 &= (100111011)_2. \end{aligned}$$

O mesmo método pode ser utilizado para converter  $N$  para qualquer base  $b$ ; divide-se  $N$  e os sucessivos quocientes  $q_i$  por  $b$  até que o último quociente seja um inteiro :



Desta forma,

$$\begin{aligned} N &= (q_n r_{n-1} \dots r_2 r_1 r_0)_b \\ &= q_n b^n + r_{n-1} b^{n-1} + \dots + r_2 b^2 + r_1 b^1 + r_0 b^0. \end{aligned}$$

Assim,

$$\begin{aligned} 12 &= (14)_8 = (C)_{16} \\ 25 &= (31)_8 = (19)_{16} \\ 315 &= (473)_8 = (13B)_{16}. \end{aligned}$$

O programa 1.1 implementa o método das divisões sucessivas para a conversão de qualquer número inteiro da base 10 para a base 2.



**Listing 1.1:** Converte um número  $N$  da base 10 para a base 2. Caso  $N < 0$ , o bit de sinal é utilizado.

```

program conversor
implicit none
integer, parameter :: base= 2
integer :: i, j, qn, rn, num, num_abs
real, parameter :: log2= 0.69314718055994530942
integer, DIMENSION(:), ALLOCATABLE :: b
!
write(*,fmt= '(a)',advance= 'no') 'Número na base 10: '
read*, num
select case (num)
case (0)
allocate(b(1))
b= 0
case (:-1)
!
!Neste caso, o vetor irá alocar um dígito a mais e atribuir o valor 1
!ao último dígito, como convenção para sinal negativo.
!
num_abs= abs(num)
j= log(real(num_abs))/log2
allocate(b(0:j + 1))
qn= num_abs
do i= 0, j
rn= mod(qn,base)
qn= qn/base
b(j - i + 1)= rn
end do
b(0)= 1
case (1:)
j= log(real(num))/log2
allocate(b(0:j))
qn= num
do i= 0, j
rn= mod(qn,base)
qn= qn/base
b(j - i)= rn
end do
end select
j= size(b)
write(*, fmt= '("A forma binária é: ")', advance= 'no')
do i= 1, j - 1
write(*,fmt= '(i1)', advance= 'no')b(i - 1)
end do
write(*,fmt= '(i1)')b(j - 1)

end program conversor

```

### 1.1.2 REPRESENTAÇÃO DE NÚMEROS REAIS E CONVERSÕES DE BASE

Dado agora um número real  $X$ , o qual possui uma parte inteira  $X_i$  e uma parte fracionária  $X_f = X - X_i$ , utiliza-se novamente o método das divisões sucessivas para  $X_i$ , enquanto que para  $X_f$  usa-se o **Método das Multiplicações Sucessivas**: multiplica-se  $X_f$  por 2, extraindo-se a parte inteira do resultado (a qual pode ser 0); o restante é novamente multiplicado por 2, repetindo-se o processo até que o resto fracionário seja 0 ou que se obtenha um padrão repetitivo, em cujo caso o número fracionário será periódico. Este método será ilustrado com dois exemplos.

**Exemplo 1.1.** Seja  $X_f = 0,8125$ , então

$$\begin{array}{r} 0,8125 \\ \times 2 \\ \hline 1,6250 \end{array} \quad \begin{array}{r} 0,6250 \\ \times 2 \\ \hline 1,2500 \end{array} \quad \begin{array}{r} 0,2500 \\ \times 2 \\ \hline 0,5000 \end{array} \quad \begin{array}{r} 0,5000 \\ \times 2 \\ \hline 1,0000 \end{array} .$$

Ou seja,

$$0,8125 = (0,1101)_2 .$$

O exemplos a seguir mostram a dificuldade de se obter a representação de um número fracionário em outra base.

**Exemplo 1.2.** Um exemplo interessante é o número  $X_f = 0,1$ . Neste caso,

$$\begin{array}{r} 0,1 \\ \times 2 \\ \hline 0,2 \end{array} \quad \begin{array}{r} 0,2 \\ \times 2 \\ \hline 0,4 \end{array} \quad \begin{array}{r} 0,4 \\ \times 2 \\ \hline 0,8 \end{array} \quad \begin{array}{r} 0,8 \\ \times 2 \\ \hline 1,6 \end{array} \quad \begin{array}{r} 0,6 \\ \times 2 \\ \hline 1,2 \end{array} \quad \begin{array}{r} 0,2 \\ \times 2 \\ \hline 0,4 \end{array} \quad \dots$$

e o processo de multiplicações sucessivas repete a sequência de dígitos 0011 *ad infinitum*. Portanto,

$$0,1 = (0,0001100110011\dots)_2 .$$

**Exemplo 1.3.** Seja  $X_f = 0,5225$ , então

$$\begin{array}{r} 0,5225 \\ \times 2 \\ \hline 1,0450 \end{array} \quad \begin{array}{r} 0,0450 \\ \times 2 \\ \hline 0,0900 \end{array} \quad \begin{array}{r} 0,0900 \\ \times 2 \\ \hline 0,1800 \end{array} \quad \begin{array}{r} 0,1800 \\ \times 2 \\ \hline 0,3600 \end{array} \quad \begin{array}{r} 0,3600 \\ \times 2 \\ \hline 0,7200 \end{array} \quad \begin{array}{r} 0,7200 \\ \times 2 \\ \hline 1,4400 \end{array} \quad \begin{array}{r} 0,4400 \\ \times 2 \\ \hline 0,8800 \end{array} \quad \begin{array}{r} 0,8800 \\ \times 2 \\ \hline 1,7600 \end{array} \quad \begin{array}{r} 0,7600 \\ \times 2 \\ \hline 1,5200 \end{array} \quad \begin{array}{r} 0,5200 \\ \times 2 \\ \hline 1,0400 \end{array} \\ \begin{array}{r} 0,0400 \\ \times 2 \\ \hline 0,0800 \end{array} \quad \begin{array}{r} 0,0800 \\ \times 2 \\ \hline 0,1600 \end{array} \quad \begin{array}{r} 0,1600 \\ \times 2 \\ \hline 0,3200 \end{array} \quad \begin{array}{r} 0,3200 \\ \times 2 \\ \hline 0,6400 \end{array} \quad \begin{array}{r} 0,6400 \\ \times 2 \\ \hline 1,2800 \end{array} \quad \begin{array}{r} 0,2800 \\ \times 2 \\ \hline 0,5600 \end{array} \quad \begin{array}{r} 0,5600 \\ \times 2 \\ \hline 1,1200 \end{array} \quad \begin{array}{r} 0,1200 \\ \times 2 \\ \hline 0,2400 \end{array} \quad \begin{array}{r} 0,2400 \\ \times 2 \\ \hline 0,4800 \end{array} \quad \begin{array}{r} 0,4800 \\ \times 2 \\ \hline 0,9600 \end{array} .$$

Ou seja,

$$0,5225 = (0,10000101110000101000\dots)_2 .$$

Este exemplo mostra que em um computador, onde o espaço para representação de um número é finito, este número terá que ser arredondado.

A **forma polinomial** de um número fracionário é dada por:

$$X_f = \alpha_1 2^{-1} + \alpha_2 2^{-2} + \alpha_3 2^{-3} + \dots .$$

Portanto, um número real  $X = X_i + X_f$  pode ser representado na base 2 por

$$\begin{aligned} X &= a_n 2^n + a_{n-1} 2^{n-1} + \dots + a_0 2^0 + \alpha_1 2^{-1} + \alpha_2 2^{-2} + \alpha_3 2^{-3} + \dots . \\ &= (a_n a_{n-1} \dots a_0, \alpha_1 \alpha_2 \alpha_3 \dots)_2 . \end{aligned}$$

**Exemplo 1.4.** Seja  $X = 75,8$ , temos

$$X_i = 75 = (1001011)_2$$

e

$$X_f = 0,8 = (0,110011001100\dots)_2 .$$

Portanto,

$$75,8 = (1001011,110011001100\dots)_2 .$$

Para converter um número fracionário da base decimal para uma base  $b$ , também aplica-se o método das multiplicações sucessivas, que, neste caso, consiste em multiplicar o número por  $b$  e extrair a parte inteira (podendo ser 0). O resto fracionário é multiplicado novamente por  $b$  e a parte inteira é extraída. Este processo deve ser repetido até sobrar o resto igual a 0 ou até se observar um padrão repetitivo.

### 1.1.3 CONVERSÃO DE NÚMEROS INTEIROS DA BASE $b$ PARA A BASE DECIMAL

Para introduzir a conversão para a base decimal, será usada novamente a base binária como um primeiro exemplo. Seja o número  $N$ , representado por

$$N = (a_m \cdots a_2 a_1 a_0)_2,$$

a sua representação na base decimal pode ser obtida simplesmente pela soma do polinômio

$$N = a_m 2^m + \cdots + a_2 2^2 + a_1 2^1 + a_0.$$

A operacionalização desta soma pode ser obtida pelos seguintes algoritmos:

#### 1.1.3.1 ALGORITMO DE HORNER.

O número  $N$  pode ser obtido na base decimal através do cálculo da sequência

$$\begin{aligned} b_m &= a_m, \\ b_{m-1} &= a_{m-1} + 2b_m, \\ b_{m-2} &= a_{m-2} + 2b_{m-1}, \\ &\vdots \\ b_1 &= a_1 + 2b_2, \\ b_0 &= a_0 + 2b_1. \end{aligned}$$

E então,

$$N = b_0.$$

#### 1.1.3.2 DIVISÃO DE RUFFINI.

Equivalente ao método anterior, diferindo somente na disposição dos coeficientes  $a_i$  e  $b_i$ :

2	$a_m$	$a_{m-1}$	$\cdots$	$a_2$	$a_1$	$a_0$
		$2b_m$	$\cdots$	$2b_3$	$2b_2$	$2b_1$
	$b_m$	$b_{m-1}$	$\cdots$	$b_2$	$b_1$	$b_0$

e, novamente,

$$N = b_0.$$

**Exemplo 1.5.** Seja o número  $(11101)_2$ . Então, a partir da sequência de Horner,

$$\begin{aligned} b_4 &= a_4 = 1, \\ b_3 &= a_3 + 2b_4 = 1 + 2 \cdot 1 = 3, \\ b_2 &= a_2 + 2b_3 = 1 + 2 \cdot 3 = 7, \\ b_1 &= a_1 + 2b_2 = 0 + 2 \cdot 7 = 14, \\ b_0 &= a_0 + 2b_1 = 1 + 2 \cdot 14 = 29. \end{aligned}$$

A partir da divisão de Ruffini,

2	1	1	1	0	1
	2	6	14	28	
	1	3	7	14	29

Portanto,

$$(11101)_2 = 29.$$

Esta metodologia pode ser generalizada para converter qualquer número inteiro na base  $b$  para a base decimal. Considere o número

$$N = (a_m \dots a_2 a_1 a_0)_b.$$

Usando o Algoritmo de Horner, por exemplo, temos a sequência

$$\begin{aligned} c_m &= a_m, \\ c_{m-1} &= a_{m-1} + bc_m, \\ c_{m-2} &= a_{m-1} + bc_{m-1}, \\ &\vdots \\ c_1 &= a_1 + bc_2, \\ c_0 &= a_0 + bc_1 \end{aligned}$$

e, novamente,

$$N = c_0.$$

#### 1.1.4 CONVERSÃO DE NÚMEROS FRACIONÁRIOS DA BASE $b$ PARA A BASE DECIMAL

Considere um número fracionário com representação finita na base binária:

$$X_f = (0, \alpha_1 \alpha_2 \dots \alpha_n)_2.$$

O seu valor na base decimal será dado por

$$X_f = \alpha_1 2^{-1} + \alpha_2 2^{-2} + \dots + \alpha_n 2^{-n}.$$

Esta soma pode ser calculada diretamente ou utilizando qualquer um dos dois métodos enunciados na seção 1.1.3, com algumas modificações.

##### 1.1.4.1 ALGORITMO DE HORNER.

No caso de um número fracionário, o algoritmo fica

$$\begin{aligned} b_n &= \alpha_n, \\ b_{n-1} &= \alpha_{n-1} + \frac{1}{2}b_n, \\ b_{n-2} &= \alpha_{n-2} + \frac{1}{2}b_{n-1}, \\ &\vdots \\ b_2 &= \alpha_2 + \frac{1}{2}b_3, \\ b_1 &= \alpha_1 + \frac{1}{2}b_2, \\ b_0 &= \frac{1}{2}b_1. \end{aligned}$$

Então,

$$N = b_0.$$

##### 1.1.4.2 DIVISÃO DE RUFFINI.

No caso de um número fracionário,

$$\begin{array}{c|cccccc}
\frac{1}{2} & a_m & a_{m-1} & \cdots & a_2 & a_1 & 0 \\
& \frac{1}{2}b_m & \frac{1}{2}b_{m-1} & \cdots & \frac{1}{2}b_3 & \frac{1}{2}b_2 & \frac{1}{2}b_1 \\
\hline
& b_m & b_{m-1} & \cdots & b_2 & b_1 & \boxed{b_0}
\end{array}$$

**Exemplo 1.6.** O número  $(0, 10111)_2$ , pelo Algoritmo de Horner, fica

$$\begin{aligned}
b_5 &= \alpha_5 = 1, \\
b_4 &= \alpha_4 + \frac{1}{2}b_5 = 1 + \frac{1}{2} \cdot 1 = \frac{3}{2}, \\
b_3 &= \alpha_3 + \frac{1}{2}b_4 = 1 + \frac{1}{2} \cdot \frac{3}{2} = \frac{7}{4}, \\
b_2 &= \alpha_2 + \frac{1}{2}b_3 = 0 + \frac{1}{2} \cdot \frac{7}{4} = \frac{7}{8}, \\
b_1 &= \alpha_1 + \frac{1}{2}b_2 = 1 + \frac{1}{2} \cdot \frac{7}{8} = \frac{23}{16}, \\
b_0 &= \frac{1}{2}b_1 = \frac{23}{32}.
\end{aligned}$$

Portanto,

$$(0, 10111)_2 = \frac{23}{32} = 0,71875.$$

Uma outra situação que pode ocorrer é quando o número binário for infinito, por exemplo, através de uma sequência de dígitos periódicos:

$$X_f = (0, \alpha_1 \alpha_2 \dots \alpha_n \overline{\beta_1 \beta_2 \dots \beta_m})_2,$$

onde  $\overline{\beta_1 \beta_2 \dots \beta_m}$  indica que a sequência de dígitos  $\beta_1 \beta_2 \dots \beta_m$  se repete *ad infinitum*. Na base decimal, tal número é dado por

$$\begin{aligned}
X_f &= \alpha_1 2^{-1} + \alpha_2 2^{-2} + \cdots + \alpha_n 2^{-n} + \beta_1 2^{-n-1} + \beta_2 2^{-n-2} + \cdots + \beta_m 2^{-n-m} \\
&+ \beta_1 2^{-n-m-1} + \beta_2 2^{-n-m-2} + \cdots + \beta_m 2^{-n-2m} \\
&+ \beta_1 2^{-n-2m-1} + \beta_2 2^{-n-2m-2} + \cdots + \beta_m 2^{-n-3m} \\
&+ \cdots.
\end{aligned}$$

Observa-se que este número pode ser escrito como

$$\begin{aligned}
X_f &= \alpha_1 2^{-1} + \alpha_2 2^{-2} + \cdots + \alpha_n 2^{-n} + (\beta_1 2^{-1} + \beta_2 2^{-2} + \cdots + \beta_m 2^{-m}) 2^{-n} \\
&+ (\beta_1 2^{-1} + \beta_2 2^{-2} + \cdots + \beta_m 2^{-m}) 2^{-n-m} \\
&+ (\beta_1 2^{-1} + \beta_2 2^{-2} + \cdots + \beta_m 2^{-m}) 2^{-n-2m} \\
&+ (\beta_1 2^{-1} + \beta_2 2^{-2} + \cdots + \beta_m 2^{-m}) 2^{-n-3m} \\
&+ \cdots,
\end{aligned}$$

$$X_f = \alpha_1 2^{-1} + \alpha_2 2^{-2} + \cdots + \alpha_n 2^{-n} + (\beta_1 2^{-1} + \beta_2 2^{-2} + \cdots + \beta_m 2^{-m}) 2^{-n} (1 + 2^{-m} + 2^{-2m} + 2^{-3m} + \cdots).$$

Usando agora a identidade,

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots, \text{ (para } |x| < 1),$$

temos

$$1 + 2^{-m} + 2^{-2m} + 2^{-3m} + \cdots = \frac{1}{1 - 2^{-m}} = \frac{2^m}{2^m - 1},$$

obtem-se

$$X_f = \alpha_1 2^{-1} + \alpha_2 2^{-2} + \cdots + \alpha_n 2^{-n} + (\beta_1 2^{-1} + \beta_2 2^{-2} + \cdots + \beta_m 2^{-m}) \frac{2^{m-n}}{2^m - 1}.$$

As duas expressões entre parênteses têm a mesma forma e podem ser calculadas diretamente usando qualquer um dos métodos descritos anteriormente.

**Exemplo 1.7.** O número fracionário

$$X_f = (0, 11\overline{010})_2 = (0, 11010010010 \dots)_2$$

tem o seu valor na base decimal dado por

$$\begin{aligned} X_f &= (1 \cdot 2^{-1} + 1 \cdot 2^{-2}) + (0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3}) \frac{2}{2^3 - 1} = \left(\frac{1}{2} + \frac{1}{4}\right) + \frac{1}{4} \frac{2}{7} = \frac{23}{28} \\ &= 0,8214285714285 \dots = 0,821428\overline{57142}. \end{aligned}$$

Em geral, se o número fracionário tem representação infinita periódica na base  $b$ ,

$$X_f = (0, \alpha_1 \alpha_2 \dots \alpha_n \overline{\beta_1 \beta_2 \dots \beta_m})_b,$$

então o seu valor decimal será dado por

$$X_f = \alpha_1 b^{-1} + \alpha_2 b^{-2} + \dots + \alpha_n b^{-n} + (\beta_1 b^{-1} + \beta_2 b^{-2} + \dots + \beta_m b^{-m}) \frac{b^{m-n}}{b^m - 1},$$

onde as expressões entre parênteses podem ser calculadas diretamente ou utilizando quaisquer um dos métodos descritos anteriormente.

## 1.2 OPERAÇÕES COM NÚMEROS BINÁRIOS

Como a maioria dos computadores usa a base  $b = 2$ , estes executam operações aritméticas em números que estão na representação binária. Para tanto, as seguintes tabelas de operações são automaticamente satisfeitas.

### 1.2.1 ADIÇÃO BINÁRIA

Uma adição no sistema binário é realizada da mesma forma que a adição no sistema decimal, lembrando que, no sistema binário, há apenas 2 dígitos. Esta operação é realizada de acordo com a seguinte tabela de adição:

+	0	1
0	<b>0</b>	<b>1</b>
1	<b>1</b>	<b>10</b>

Para somar números com mais de 2 algarismos, o mesmo processo de transporte para a coluna posterior, utilizado na adição decimal, é empregado. Por exemplo, se  $1 = (01)_2$  e  $3 = (11)_2$ , então

$$1 + 3 = (01)_2 + (11)_2 = (100)_2 = 4.$$

Outro exemplo, se  $10 = (1010)_2$  e  $15 = (1111)_2$ , então

$$10 + 15 = (1010)_2 + (1111)_2 = (11001)_2 = 25.$$

### 1.2.2 SUBTRAÇÃO BINÁRIA

A subtração é análoga à adição, sendo realizada de acordo com a tabela:

-	0	1
0	<b>0</b>	<b>1</b>
1	<b>1</b>	<b>0</b>

Deve-se ressaltar que a operação **0 - 1** resulta em **1**, porém com o transporte de **1** para a coluna à esquerda, que deve ser acumulado ao subtraendo e, por consequência, subtraído do minuendo. Por exemplo, se  $7 = (111)_2$  e  $4 = (100)_2$ , então

$$7 - 4 = (111)_2 - (100)_2 = (11)_2 = 3.$$

Outro exemplo, se  $10 = (1010)_2$  e  $8 = (1000)_2$ , então

$$10 - 8 = (1010)_2 - (1000)_2 = (10)_2 = 2.$$

### 1.2.3 MULTIPLICAÇÃO BINÁRIA

Procede-se como em uma multiplicação no sistema decimal, de acordo com a tabela de multiplicação:

×	0	1
0	0	0
1	0	1

Por exemplo, se  $26 = (11010)_2$  e  $2 = (10)_2$ , então

$$26 \times 2 = (11010)_2 \times (10)_2 = (110100)_2 = 52.$$

A divisão binária é um procedimento um tanto mais complicado e não será abordado aqui.

## 1.3 REPRESENTAÇÃO DE NÚMEROS EM COMPUTADORES DIGITAIS

Nesta seção serão apresentadas algumas das representações usadas para armazenar números inteiros ou reais na memória de um computador. As representações de números inteiros ou reais apresentadas na seção 1.1 não são suficientes; é necessário distinguir-se, por exemplo, o sinal do número. Como não existe a representação de um sinal  $+$  ou  $-$  na memória de um computador, o recurso utilizado é acrescentar um bit adicional, para computadores binários, ao número para representar o sinal. Este bit é denominado *bit de sinal*.

### 1.3.1 REPRESENTAÇÃO DE NÚMEROS INTEIROS

A representação mais direta de números inteiros é denominada *Sinal-Módulo*. Nesta representação, o valor absoluto do número inteiro é obtido diretamente a partir dos algoritmos discutidos na seção 1.1, enquanto que o sinal é representado por um dígito adicional colocado à esquerda do número. Quando a representação é binária, o bit de sinal é dito ocupar a posição do bit mais significativo.

Então, supondo que a memória do computador disponha de  $q$  dígitos para sua representação, um número inteiro na base  $b$  será representado pelo computador através da sequência de dígitos

$$a_{q-1}a_{q-2} \dots a_1a_0, \quad (1.2)$$

sendo  $\{a_0, a_1, \dots, a_{q-1}\} \in \{0, 1, \dots, b-1\}$ , com  $a_{q-1}$  representando o sinal do número. Esta sequência de dígitos é denominada **palavra**. Por exemplo, no sistema binário convencionam-se usar  $a_{q-1} = 0$  para “+” e  $a_{q-1} = 1$  para “-”.

A conversão do número internamente representado por (1.2) para o sistema decimal é realizado através de uma fórmula semelhante à forma polinomial (1.1):

$$N = s \times \sum_{k=0}^{q-2} a_k \times b^k, \quad (1.3)$$

sendo

- $N$  o número inteiro na base decimal.
- $s$  o sinal (ou  $+1$  ou  $-1$ ).
- $q-1$  o número de dígitos disponível para representar o valor absoluto de  $N$ .
- $b$  a base, às vezes denominada de *radix* (um inteiro maior que 1).
- $a_k$  um dígito válido na representação ( $0 \leq a_k < b$ ),  $k = 0, 1, \dots, q-1$ .

Os valores em questão para as quantidades em (1.3) dependem da arquitetura e do compilador em uso.

**Exemplo 1.8.** O compilador Intel Fortran [1] possui 4 modelos de representação de inteiros com 1, 2, 4 e 8 bytes, também denominados de *espécies*. Sendo para todos os casos  $b = 2$ , o valor absoluto do maior número inteiro que pode ser representado internamente para cada espécie  $N_{\max}^p$ , ( $p = 1, 2, 4, 8$ ) é, a partir de (1.3),

$$N_{\max}^p = \sum_{k=0}^{8p-2} 2^k = 1 + 2 + 2^2 + \dots + 2^{8p-2} = 2^{8p-1} - 1 = \begin{cases} 127, & \text{para } p = 1; \\ 32.767, & \text{para } p = 2; \\ 2.147.483.647, & \text{para } p = 4; \\ 9.223.372.036.854.775.807, & \text{para } p = 8. \end{cases}$$

Outras representações de números inteiros em computadores existem, como por exemplo as representações *complemento de 1* ou *complemento de 2* [2]; porém, estas não serão discutidas aqui.

A representação de um número inteiro em um computador é exata. Operações aritméticas entre números inteiros também é exata, sob as seguintes condições:

1. o resultado não pode se encontrar fora do intervalo de números inteiros que podem ser representados;
2. a divisão somente pode ser realizada entre números exatamente divisíveis, isto é, a parte fracionária deve ser nula.

### 1.3.2 REPRESENTAÇÃO DE NÚMEROS REAIS

A representação de números reais em computadores é também denominada *representação de ponto flutuante*. Em uma representação de ponto flutuante, um número é representado internamente através de uma notação científica, isto é, por um bit de sinal  $s$  (interpretado como positivo ou negativo), um expoente inteiro exato  $e$  e uma mantissa inteira positiva  $M$ , sendo que um número limitado de dígitos é permitido para  $e$  e  $M$ . Tomando todas estas quantidades juntas, estas representam o número

$$x = s \times (0, d_1 d_2 \dots d_n) \times b^e, \quad (1.4)$$

o qual está escrito em uma forma legível para seres humanos. Além das quantidades já definidas, em (1.4) os dígitos  $d_1, d_2, \dots, d_n$  são limitados pela base  $b$  e o expoente é limitado ao intervalo  $e_{\min} \leq e \leq e_{\max}$ . Adicionalmente,  $n \geq 1$  é denominado de *número de dígitos do sistema* e define o tamanho da mantissa  $M = 0, d_1 d_2 \dots d_n$ .

Contudo, um computador somente pode representar os valores de  $e$  e  $E$  através de dígitos na base  $b$ . Um computador digital ( $b = 2$ ), por exemplo, dispõe sempre de um *tamanho de palavra* finito, isto é, o número total de bits que podem ser utilizados para representar  $s$  (1 bit), a parte exponencial e a mantissa é sempre fixo, para uma dada espécie de variável real. Um número real de precisão simples, por exemplo, é tipicamente representado por uma palavra de 4 bytes ou 32 bits, sendo que 1 bit é utilizado para representar o sinal, enquanto que 8 bits são utilizados para representar a parte exponencial, restando 23 bits para representar a mantissa. Desta forma, tal número será representado na memória do computador como

$$x = s \underbrace{e_7 e_6 e_5 e_4 e_3 e_2 e_1 e_0}_{\text{expoente}} \underbrace{d_{23} d_{22} \dots d_2 d_1}_{\text{mantissa}},$$

onde  $\{s, e_0, \dots, e_7, d_1, \dots, d_{23}\} = \{0, 1\}$ . A figura 1.1 ilustra representações em 4 bytes de alguns números. Uma descrição mais aprofundada acerca da representação binária de números em computadores digitais pode ser obtida em [2, seção 2.5].

A conversão do número  $x$  representado em (1.4) para a base decimal pode ser realizada pela fórmula polinomial

$$x = s \times b^e \times \sum_{k=1}^n d_k \times b^{-k}.$$

Como exemplo, a tabela 1.2 mostra os valores de  $n$ ,  $e_{\min}$  e  $e_{\max}$  para o compilador Intel Fortran.

Para uma base  $b$  qualquer, denotando este sistema pelo símbolo

$$x[b, n, e_{\min}, e_{\max}],$$

observam-se as seguintes características:



	$s$	expoente de 8 bits	mantissa de 23 bits
$\frac{1}{2}$	$\underbrace{0}_{s}$	10000000	10000000000000000000000
$\frac{3}{4}$	0	10000010	11000000000000000000000
$\frac{1}{4}$	0	01111111	10000000000000000000000
$10^{-7}$	0	01101001	11010110101111111001010

Figura 1.1: Representações de ponto flutuante para alguns números em uma palavra típica de 32 bits (4 bytes).

- O menor número positivo que pode ser representado neste sistema é

$$x_{\min} = 0,1 \times b^{e_{\min}} = b^{e_{\min}-1}.$$

Valores para  $x_{\min}$  válidos para o compilador Intel Fortran são apresentados na tabela 1.2. Isto significa que qualquer número  $x$  tal que

$$-x_{\min} < x < x_{\min}$$

não poderá ser representado pelo computador. Esta ocorrência é denominada *underflow*. Os compiladores podem ser instruídos ou a parar o processamento neste ponto, disparando uma mensagem de erro, ou a seguir o processamento arredondando  $x = 0$ .

- O maior número positivo que pode ser representado neste sistema é

$$x_{\max} = 0, \underbrace{(b-1)(b-1)\dots(b-1)}_{n \text{ vezes}} \times b^{e_{\max}} = (b-1) \times \left( \sum_{k=1}^n b^{-k} \right) \times b^{e_{\max}} = (1 - b^{-n}) b^{e_{\max}}.$$

Valores para  $x_{\max}$  válidos para o compilador Intel Fortran são apresentados na tabela 1.2. Isto significa que qualquer número  $x$  tal que

$$x < -x_{\max} \text{ ou } x > x_{\max}$$

não poderá ser representado pelo computador. Esta ocorrência é denominada *overflow*. Os compiladores usualmente tomam duas possíveis providências quando detectam um overflow; ou páram o processamento do programa emitindo uma mensagem de erro, ou continuam o processamento atribuindo a  $x$  o valor simbólico  $x = -\text{Infinito}$  ou  $x = \text{Infinito}$ .

- O maior número que pode ser somado ou subtraído a 1,0, com o resultado permanecendo indistinguível de 1,0 é

$$x_{\text{eps}} = b^{1-n}. \quad (1.5)$$

Os valores de  $x_{\text{eps}}$  para o compilador Intel Fortran são também apresentados na tabela 1.2. A quantidade  $x_{\text{eps}}$  também é denominada de *épsilon da máquina* ( $\epsilon_m$ ) ou de *precisão da máquina*.

- Somente um conjunto finito  $\mathcal{F}$  de números racionais podem ser representados na forma (1.4). Os números neste conjunto são denominados **números de ponto flutuante**. Para uma representação normalizada ( $d_1 \neq 0$ ), este conjunto contém precisamente

$$2(b-1)(e_{\max} - e_{\min} + 1)b^{n-1} + 1$$

números racionais.

Tabela 1.2: Valores de  $n$ ,  $e_{\min}$ ,  $e_{\max}$ ,  $x_{\min}$ ,  $x_{\max}$  e  $x_{\text{eps}}$  para o compilador Intel Fortran.

Espécie	REAL(4)	REAL(8)	REAL(16)
$n$	24	53	113
$e_{\min}$	-125	-1021	-16381
$e_{\max}$	128	1024	16384
$x_{\min}$	$1,1754944 \times 10^{-38}$	$2,225073858507201 \times 10^{-308}$	$3,362103143112093506 \dots \times 10^{-4932}$
$x_{\max}$	$3,4028235 \times 10^{38}$	$1,797693134862316 \times 10^{308}$	$1,189731495357231765 \dots \times 10^{4932}$
$x_{\text{eps}}$	$1,1920929 \times 10^{-7}$	$2,220446049250313 \times 10^{-16}$	$1,925929944387235853 \dots \times 10^{-34}$

**Exemplo 1.9.** Considere um modelo simplificado de representação numérica de ponto flutuante dado por  $x [2, 4, -5, 6]$ . Para este sistema:

- o menor número positivo possível é:

$$x_{\min} = (0, 1000)_2 \times 2^{-5} = 2^{-5-1} = \frac{1}{64};$$

ou seja, a região de *underflow* consiste no intervalo

$$-\frac{1}{64} < x < \frac{1}{64}.$$

- O maior número positivo possível é:

$$x_{\max} = (0, 1111)_2 \times 2^6 = (1 - 2^{-4}) 2^6 = 60;$$

ou seja, as regiões de *overflow* consistem nos intervalos

$$x < -60, \quad x > 60.$$

- O maior número que pode ser somado ou subtraído de 1,0 e que mantém o resultado inalterado é:

$$x_{\text{eps}} = 2^{1-4} = \frac{1}{8}.$$

- O número de elementos em  $\mathcal{F}$  é:

$$2.1. (6 + 5 + 1) 2^{4-1} + 1 = 193.$$

**Exemplo 1.10.** Considere o sistema de números de ponto flutuante  $x [2, 3, -1, 2]$ . Para este sistema:

- o menor número positivo possível é:

$$x_{\min} = 2^{-1-1} = \frac{1}{4}.$$

ou seja, a região de *underflow* consiste no intervalo

$$-\frac{1}{4} < x < \frac{1}{4}.$$

- O maior número positivo possível é:

$$x_{\max} = (1 - 2^{-3}) 2^2 = \left(1 - \frac{1}{8}\right) 4 = \frac{7}{2};$$

ou seja, as regiões de *overflow* consistem nos intervalos

$$x < -\frac{7}{2}, \quad x > \frac{7}{2}.$$

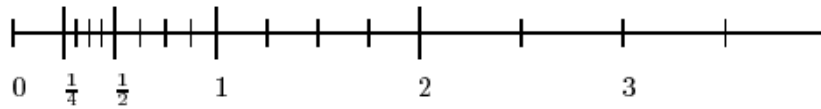
- O maior número que pode ser somado ou subtraído de 1,0 e que mantém o resultado inalterado é:

$$x_{\text{eps}} = 2^{1-3} = \frac{1}{4}.$$

- O número de elementos em  $\mathcal{F}$  é:

$$2.1. (2 + 1 + 1) 2^{3-1} + 1 = 33.$$

A fração positiva dos números possíveis em  $x [2, 3, -1, 2]$  está indicada na figura 1.2. Cada risco vertical posiciona um número representável neste sistema.



**Figura 1.2:** Números normalizados positivos representáveis em  $x[2, 3, -1, 2]$ . Os riscos verticais posicionam os números. Para cada número positivo, existe um correspondente número negativo.

## 1.4 FONTES DE ERROS E INCERTEZAS

Embora sempre se busque soluções “exatas” aos problemas que enfrentamos, raramente atingimos o nosso objetivo. Erros e incertezas podem ser introduzidos em cada etapa da formulação e solução de problemas. A natureza das incertezas que surgem quando se busca a solução de um problema será abordada neste capítulo. Simultaneamente, os erros introduzidos pela computação numérica, destinada a buscar a solução desejada, serão examinados com um certo grau de detalhe.

O processo de solução de um problema é dividido em três fases:

1. Formulação precisa de um modelo matemático e o seu modelo numérico relacionado.
2. Construção de um método destinado a resolver o problema numérico.
3. Implementação de um método para calcular a solução.

Na discussão que será feita a respeito das fontes de erro em cálculo numérico, não serão considerados erros triviais que podem ser evitados, tais como copiar uma fórmula erroneamente ou realizar um erro de sintaxe na programação, muito embora tais erros ocorram e perfaçam uma fração considerável do esforço e do tempo dispendidos ao se trilhar as três fases mencionadas acima.

Neste capítulo estaremos somente interessados nos erros que resultam ser inevitáveis, dada a própria natureza da representação finita de números em um computador e/ou da implementação numérica de um determinado cálculo. As incertezas introduzidas contaminam a solução e é importante tentar-se *balancear as incertezas*. Se a incerteza no modelo matemático é de 1%, então não faz sentido a implementação de um modelo numérico e de um método que atinja 6 dígitos de precisão, por exemplo.

O diagrama da figura 1.3 ilustra o processo usualmente percorrido quando se busca uma solução para um problema físico real a partir de uma modelagem, inicialmente matemática, seguida por uma modelagem computacional e, finalmente, passando pela implementação do método numérico a partir da modelagem computacional, seguida pela obtenção dos resultados. As incertezas ocorrem desde a fase de modelagem matemática até a solução numérica. Neste capítulo, serão abordadas algumas fontes de incertezas nas etapas de modelagem computacional e implementação do método numérico.

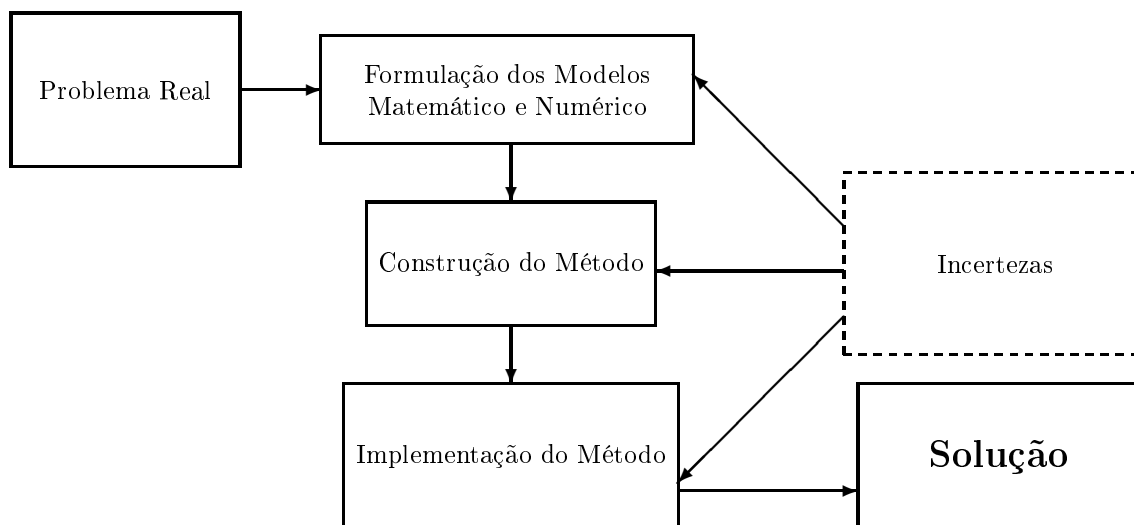
## 1.5 ERROS NA REPRESENTAÇÃO E NA ÁLGEBRA DE PONTO FLUTUANTE

Nesta seção será feita uma breve descrição dos principais erros envolvidos na representação de pontos flutuantes e nas operações algébricas entre os mesmos.

### 1.5.1 PRECISÃO E ACURÁCIA

Os conceitos de precisão e acurácia são amiúde confundidos entre si. A diferença entre ambos é oriunda da diferença entre o *hardware* e o *software* à disposição do programador.

**Precisão (*precision*):** refere-se ao quão próximo um número representado pelo computador representa o número que ele almeja representar. A precisão de um número é governada pelo número de dígitos empregados na representação e na álgebra. Assim, a constante  $\pi$  será representada com maior precisão utilizando 8 bytes do que utilizando 4 bytes para armazenar o número (ver tabela 1.2).



**Figura 1.3:** Diagrama que representa o processo de solução numérica de um problema físico real, indicando em que etapas entram as incertezas.

**Acurácia (accuracy) ou Exatidão:** refere-se a quão próximo um número representado pelo computador (como resultado de uma série de operações, por exemplo) está do valor correto do número que ele almeja representar. A acurácia é governada pelos erros (de truncamento e arredondamento) no método numérico empregado. Assim, se os números  $\pi_1 = 3,1416304958$  e  $\pi_2 = 3,1415809485$  almejam ambos a representar o número  $\pi = 3,141592654\dots$ , o número  $\pi_2$  possui maior acurácia que  $\pi_1$ ,<sup>1</sup> embora ambos possuam a mesma precisão.

Com frequência, em linguagem coloquial refere-se à precisão quando na verdade o correto seria referir-se à acurácia de um resultado. As seções a seguir indicam como se pode medir a acurácia de um número através do cálculo dos erros absoluto e relativo do mesmo.

## 1.5.2 ERROS ABSOLUTO E RELATIVO

São duas medidas relacionadas entre si, largamente empregadas na análise de erro numérico.

### 1.5.2.1 ERRO ABSOLUTO

Seja  $X$  o valor exato (não conhecido) de um número e  $fl(X)$  o seu valor aproximado (conhecido) por uma representação de ponto flutuante. O *erro absoluto* ( $EA_X$ ) é definido como o valor absoluto da diferença entre o valor exato e o valor aproximado:

$$EA_x = |X - fl(X)|.$$

Ou seja, conhecendo-se  $fl(X)$  e  $EA_X$ , pode-se afirmar que

$$X = fl(X) \pm EA_x.$$

Em geral, somente é possível *estimar-se* o valor do erro absoluto.

Por exemplo, Arquimedes estimou o valor de  $\pi$  através da média do perímetro de polígonos que estavam contidos em uma circunferência de raio unitário e de polígonos que continham a circunferência. Fazendo uso deste método, Arquimedes foi capaz de estimar

$$\pi_{min} = 3,1409\dots = 3 + \frac{1137}{8069} < \pi < \pi_{max} = 3 + \frac{1335}{9347} = 3,1428\dots$$

<sup>1</sup>Ou, simplesmente, é mais acurado que  $\pi_1$ .

Desta forma, Arquimedes obteve erros absolutos iguais a  $EA_\pi = 6,830 \times 10^{-4}$  para  $\pi_{min}$ ,  $EA_\pi = 1,2339 \times 10^{-4}$  para  $\pi_{max}$  e  $EA_\pi = 2,7546 \times 10^{-4}$  para a média entre  $\pi_{min}$  e  $\pi_{max}$ . Portanto, Arquimedes concluiu que

$$\pi = \frac{1}{2}(\pi_{min} + \pi_{max}) \pm \frac{1}{2}(\pi_{max} - \pi_{min}) = 3,14187 \pm 0,00096.$$

### 1.5.2.2 ERRO RELATIVO

Seja  $X$  o valor exato de um número e  $fl(X)$  o seu valor aproximado, o *erro relativo* ( $ER_X$ ) é definido como o erro absoluto dividido por  $|fl(X)|$ :

$$ER_X = \frac{EA_X}{|fl(X)|} = \left| \frac{X - fl(X)}{fl(X)} \right|.$$

Voltando ao exemplo anterior, os erros relativos das estimativas de Arquimedes foram:  $ER_\pi = 2,1741 \times 10^{-4}$  para  $\pi_{min}$ ,  $ER_\pi = 3,9262 \times 10^{-4}$  para  $\pi_{max}$  e  $ER_\pi = 8,7674 \times 10^{-5}$  para a média.

Em geral, a melhor medida para se estimar a precisão de uma aproximação é o erro relativo, pois este indica diretamente o número de dígitos significativos corretos na aproximação.

### 1.5.3 ERROS NA REPRESENTAÇÃO: ARREDONDAMENTO E TRUNCAMENTO

O tamanho finito da palavra utilizada em um computador digital para a representação de números de ponto flutuante provoca o surgimento de diversos tipos de erros, tanto na representação destes números quanto na álgebra que os envolve. Uma estratégia que reduz estes erros, empregada na maior parte dos computadores, consiste em empregar números de ponto flutuante **normalizados**, isto é, números cuja mantissa  $M$  está sempre dentro do intervalo

$$\frac{1}{b} \leq M < 1,$$

ou seja,  $0,5 \leq M < 1$  para computadores de base  $b = 2$ . Esta providência diminui o número de zeros à direita da vírgula e maximiza o número de dígitos não nulos utilizados para representar um dado número.

Entretanto, mesmo em um sistema com representação normalizada, nem todos os números reais podem ser representados. Utilizando novamente o exemplo do sistema  $x[2, 4, -5, 6]$ , o número racional

$$y = 0,12345999\dots$$

não pode ser exatamente representado. A forma de  $y$  em base 2 é:

$$y = 0,12345999\dots = (0,000111111001101\dots)_2.$$

Para escrever  $y$  de acordo com o sistema  $x[2, 4, -5, 6]$ , deve-se primeiro normalizar de acordo com as operações:

$$\begin{aligned} (y)_2 &= 2^{-4} + 2^{-5} + 2^{-6} + 2^{-7} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-15} + \dots \\ &= 2^{-3} \times (2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-6} + 2^{-9} + 2^{-10} + 2^{-12} + \dots) \\ &= (0,11111001101\dots) \times 2^{-3}, \end{aligned}$$

o qual está agora na forma normalizada. De acordo com (1.4), podemos identificar então

$$M = 0,11111001101\dots, \quad e = -3.$$

Contudo, para o sistema  $x[2, 4, -5, 6]$  pode-se usar somente 4 dígitos na mantissa. Desta forma, uma aproximação possível para  $(y)_2$  fica:

$$fl((y)_2) = (0,1111) \times 2^{-3},$$

o qual corresponde ao seguinte número na base 10:  $fl(y) = 0,1171875\dots$ , resultando em erros absoluto e relativo:

$$EA_y = 6,272 \times 10^{-3}, \quad ER_y = 5,3525 \times 10^{-2} = 5,35\%.$$

Este procedimento de aproximação é denominado **truncamento**.<sup>2</sup> Uma definição mais rigorosa do método de truncamento será apresentada a seguir.

Dado um número  $X$  já na forma normalizada que não possua representação exata no sistema  $x[b, n, e_{\min}, e_{\max}]$ . Sejam agora  $\underline{X}$  o maior número representável no sistema e que seja menor que  $X$  e  $\overline{X}$  o menor número representável no sistema e que seja maior que  $X$ . Então,

$$\underline{X} \leq X \leq \overline{X}.$$

Pode-se escrever  $X$  como

$$X = (0, d_1 d_2 \dots d_n) \times b^e + g_X \times b^{e-n},$$

onde

$$0 \leq g_X < 1$$

é a parcela de  $X$  que não pode ser incluída na sua representação. Existem então 2 maneiras de se realizar a aproximação:

**Truncamento.** O truncamento consiste em simplesmente ignorar  $g_X$ . Assim,

$$fl(X) = (0, d_1 d_2 \dots d_n) \times b^e,$$

o qual é representável no sistema. Neste caso, os erros absoluto e relativo são

$$\begin{aligned} EA_X &= |X - fl(X)| = |g_X| \times b^{e-n} < b^{e-n}, \\ ER_X &= \frac{EA_X}{fl(X)} = \frac{g_X \times b^{e-n}}{(0, d_1 d_2 \dots d_n) \times b^e} < \frac{b^{-n}}{(0, 1)_b} = b^{-n+1}, \end{aligned}$$

pois  $(0, d_1 d_2 \dots d_n) \geq (0, 1)_b = b^{-1}$ . Desta forma, obtém-se limites superiores para ambos os erros. No exemplo acima, pode-se escrever:

$$(y)_2 = (0, 1111) \times 2^{-3} + g_y \times 2^{-3-4}, \text{ sendo } g_y = 0, 11001101 \dots$$

Realizando então o truncamento, obtém-se  $fl((y)_2)$ .

**Arredondamento.** No arredondamento,<sup>3</sup> executa-se a seguinte operação:

$$fl(X) = \begin{cases} (0, d_1 d_2 \dots d_n) \times b^e, & \text{se } |g_X| < \frac{1}{2} \\ (0, d_1 d_2 \dots (d_n + 1)) \times b^e, & \text{se } |g_X| \geq \frac{1}{2} \end{cases} \quad (1.6)$$

Neste caso, o erro absoluto da operação será

$$EA_X = |X - fl(X)| = \begin{cases} |g_X| \times b^{e-n}, & \text{se } |g_X| < \frac{1}{2} \\ |g_X - 1| \times b^{e-n}, & \text{se } |g_X| \geq \frac{1}{2} \end{cases} \left( < \frac{1}{2} b^{e-n} \right),$$

de onde se obtém uma estimativa superior e o erro relativo será

$$ER_X < \begin{cases} \frac{\frac{1}{2} b^{e-n}}{(0, d_1 d_2 \dots d_n) \times b^e}, & \text{se } |g_X| < \frac{1}{2} \\ \frac{\frac{1}{2} b^{e-n}}{(0, d_1 d_2 \dots (d_n + 1)) \times b^e}, & \text{se } |g_X| \geq \frac{1}{2} \end{cases} \left( < \frac{\frac{1}{2} b^{e-n}}{(0, 1)_b \times b^e} = \frac{1}{2} b^{-n+1} \right),$$

o qual fornece uma estimativa superior para o erro relativo. No exemplo acima, como  $g_y > 1/2$ , deve-se somar 1 ao dígito  $d_4$  resultando, com o auxílio da tabela de adição de binários apresentada na seção 1.2.1,

$$fl((y)_2) = (0, 1111) \times 2^{-3} + (0, 0001) \times 2^{-3} = (1, 0000) \times 2^{-3} = (0, 1000) \times 2^{-2}.$$

Neste caso, obtém-se

$$fl(y) = 0, 125,$$

o qual possui um erro relativo de 1,2% de  $y$ , bem menor que o erro obtido com o truncamento, que foi de 5,35%.

<sup>2</sup>Tradução livre do termo em inglês *chopping*.

<sup>3</sup>Tradução do termo inglês *rounding*.

Computadores mais recentes modificam ligeiramente o arredondamento em relação à fórmula apresentada em (1.6). Nesta, o último dígito significativo ( $d_n$ ) não será alterado se  $|g_X| < 1/2$  e este será alterado se  $|g_X| \geq 1/2$ . Há, portanto, uma ligeira preferência para a alteração de  $d_n$  no processo de arredondamento, o que insere um erro sistemático no processo. Atualmente, se  $|g_X| = 1/2$ , o valor de  $d_n$  será alterado somente em metade das situações, com base em algum critério. Este critério pode ser a paridade de  $d_n$ , por exemplo. Assim, para  $b = 10$  o número 12,5 seria arredondado para 12, enquanto que 13,5 seria arredondado para 14. Este critério é também denominado *arredondamento par* [3].

### 1.5.4 NÚMERO DE DÍGITOS SIGNIFICATIVOS CORRETOS

Quando se conta o *número de dígitos* em um valor numérico, não se deve incluir os zeros no início do número, uma vez que estes zeros somente auxiliam a localizar a posição ideal do ponto decimal. Caso se queira contar o *número de decimais*, então os zeros à direita do ponto decimal devem ser incluídos. Por exemplo, o número 0,00147 é dado com três dígitos mas possui cinco decimais. O número 12,34 é dado com quatro dígitos, mas possui apenas dois decimais.

Quando se trabalha com uma representação de um número obtida por meio de um processo de arredondamento ou truncamento, uma maneira alternativa para se estimar a qualidade da aproximação, ou seja a exatidão do número, consiste em computar o *número de dígitos significativos corretos* da representação. Se  $fl(X)$  é uma aproximação de  $X$  em uma representação de base  $b$ , então

$$b^{-k-1} < \left| \frac{X - fl(X)}{X} \right| = |\epsilon_X| \leq b^{-k}, \quad (1.7)$$

onde  $k$  é o maior número inteiro positivo para o qual as desigualdades acima são verificadas. Diz-se então que  $fl(X)$  é uma aproximação para  $X$  na base  $b$  com no mínimo  $k$  e no máximo  $k + 1$  dígitos significativos corretos.

**Exemplo 1.11.** Sejam  $b = 10$ ,  $X = 1/6$  e  $fl(X) = 0,16667$ ; então

$$|\epsilon_X| = \left| \frac{1/6 - 0,16667}{1/6} \right| = 2 \times 10^{-5}.$$

Mas  $10^{-4-1} < 2 \times 10^{-5} < 10^{-4}$ ; ou seja,  $fl(X)$  é uma aproximação para  $X$  com no mínimo 4 e no máximo 5 dígitos corretos.

### 1.5.5 ERROS NA ÁLGEBRA DE PONTO FLUTUANTE

Adicionalmente aos erros resultantes do truncamento ou do arredondamento na representação de números de ponto flutuante por computadores, as operações algébricas que necessariamente são realizadas pelo computador introduzem dois outros tipos de erros no resultado destas operações e que tendem a se acumular à medida que uma grande quantidade de operações de ponto flutuante são realizadas. Estes erros são os **erros de arredondamento**<sup>4</sup> e os **erros de truncamento**.<sup>5</sup> Estes tipos de erros adicionais serão brevemente discutidos nesta seção.

#### 1.5.5.1 ERROS DE ARREDONDAMENTO

A origem deste tipo de erro está também relacionada com a representação finita das palavras em um computador e surge com a realização de operações de ponto flutuante pelo mesmo.

Um exemplo simples ilustra o surgimento deste tipo de erro. Suponha-se que se esteja usando um sistema numérico de base 10 com 5 dígitos na mantissa, semelhante à representação (1.4). Deseja-se agora calcular o valor da função

$$f(x) = \frac{1 - \cos x}{\sin x} = \frac{\sin x}{1 + \cos x}$$

<sup>4</sup>Neste caso, o termo “erros de arredondamento” possui um significado distinto do processo de arredondamento utilizado na representação de números reais, discutida na seção 1.5.3. O termo, neste contexto, consiste na tradução usualmente empregada para o termo em inglês *round-off errors*.

<sup>5</sup>Aqui, também, o termo “erros de truncamento” não se refere ao processo de truncamento discutido na seção 1.5.3, mas sim ao tipo de erro que em inglês é denominado *truncation errors*.



para  $x = 0,007$ . Rotinas intrínsecas fornecidas pelo fabricante do compilador utilizado encarregam-se de calcular o valor das funções trigonométricas dentro da precisão disponível, por meio de um processo de arredondamento. Assim,

$$\begin{aligned}\operatorname{sen}(0,007) &= 0,69999 \times 10^{-2} \\ \cos(0,007) &= 0,99998.\end{aligned}$$

A primeira expressão para  $f(x)$  fornece:

$$f(x) = \frac{1 - \cos x}{\operatorname{sen} x} = \frac{1 - 0,99998}{0,69999 \times 10^{-2}} = \frac{0,2 \times 10^{-4}}{0,69999 \times 10^{-2}} = 0,28572 \times 10^{-2},$$

enquanto que a segunda expressão fornece:

$$f(x) = \frac{\operatorname{sen} x}{1 + \cos x} = \frac{0,69999 \times 10^{-2}}{1 + 0,99998} = 0,35000 \times 10^{-2},$$

sendo que este último resultado é o correto, dentro da precisão de 5 dígitos disponível. O erro relativo entre o primeiro valor (errado) e o segundo (correto) é de 22,5%. Na primeira expressão, devido à escolha feita na precisão, restou somente um dígito relevante no numerador após a subtração. Isto levou a uma perda de exatidão e a um resultado errôneo devido ao cancelamento de dois números muito próximos entre si. A perda deletéria de exatidão devida à subtração em precisão finita de duas quantidades muito próximas entre si é denominada **cancelamento catastrófico**.

A ocorrência do cancelamento catastrófico seria evitada caso o sistema de representação dispusesse de, pelo menos, mais um dígito significativo na mantissa; porém, o ponto a ser frisado aqui é que muito facilmente este tipo de erro de arredondamento ocorre, devido ao tamanho finito da palavra no computador. Por outro lado, caso fosse solicitado o valor de  $f(x)$  para  $x \approx \pi$ , seria a segunda expressão que forneceria um valor incorreto, enquanto que a primeira forneceria um valor correto.

Este exemplo simples demonstra a perda de precisão numérica devida a erros de arredondamento, onde o número de dígitos significativos é reduzido na subtração de dois números próximos entre si. Isto mostra que não é possível confiar cegamente no cálculo realizado; deve-se sempre analisar cuidadosamente o algoritmo empregado na procura de possíveis fontes de erros.

Considera-se então um número real  $X$ , o qual possui uma representação de máquina  $fl(X)$  que pode ser escrita como

$$fl(X) = X(1 + \epsilon_X),$$

onde  $\epsilon_X$  é o erro associado com a representação de  $X$ . De forma equivalente, pode-se escrever

$$fl(X) = X + \delta_X,$$

sendo  $\delta_X = X\epsilon_X$ . Então,  $|\delta_X| = EA_X$ . Pode-se ver que

$$\begin{aligned}\epsilon_X &= \frac{fl(X) - X}{X} = \frac{fl(X) - X}{fl(X) - \delta_X} = \frac{fl(X) - X}{fl(X)} \frac{1}{1 - \delta_X/fl(X)}, \\ \epsilon_X &= \frac{fl(X) - X}{fl(X)} \left( 1 + \frac{\delta_X}{fl(X)} + \frac{\delta_X^2}{fl(X)^2} + \dots \right) \approx \frac{fl(X) - X}{fl(X)} \left( 1 + \frac{X}{fl(X)} \epsilon_X \right), \\ \epsilon_X &= \frac{fl(X) - X}{fl(X)} \left( 1 - \frac{fl(X) - X}{fl(X)} \frac{X}{fl(X)} \right)^{-1} \approx \frac{fl(X) - X}{fl(X)} + \left( \frac{fl(X) - X}{fl(X)} \right)^2 \frac{X}{fl(X)},\end{aligned}$$

ou seja,  $|\epsilon_X| \approx ER_X \leq \epsilon_{\max}$ , onde  $\epsilon_{\max}$  é denominado de **unidade na última posição**<sup>6</sup>, ou **uup**. Isto é, com  $k$  dígitos na mantissa e com a base  $b$ ,

$$\text{uup} \simeq b^{-k}.$$

Dados agora dois números reais positivos  $X$  e  $Y$ , deseja-se estimar os erros relativos das operações algébricas entre ambos:

$$X \text{ op } Y,$$

<sup>6</sup>Do termo em inglês *unit in the last place*, ou *ulp*.



sendo  $op$  um das operações: “+”, “−”, “×” ou “÷”, conhecendo-se os erros relativos  $\epsilon_X$  e  $\epsilon_Y$  correspondentes:

$$fl(X) = X(1 + \epsilon_X) \text{ e } fl(Y) = Y(1 + \epsilon_Y).$$

Os resultados destas operações de ponto flutuante são escritos:

$$fl(X + Y), fl(X - Y), fl(X \times Y), fl(X/Y).$$

Assumindo que não ocorra overflow nem underflow, supõe-se que seja possível escrever

$$fl(X op Y) = (X op Y)(1 + \epsilon_{op}),$$

sendo

$$\epsilon_{op} = \frac{fl(X op Y) - (X op Y)}{(X op Y)},$$

com  $|\epsilon_{op}| = ER_{op}$ , o erro relativo da operação.

Grande parte dos computadores atualmente empregados utilizam o padrão IEEE para operações aritméticas de ponto flutuante no sistema binário. Entre outros recursos, este padrão especifica que todas as operações aritméticas devem ser idealmente realizadas como se o computador dispusesse de precisão infinita e somente após obtido o resultado este deve ser transformado para o sistema de ponto flutuante em uso através de um processo de arredondamento [3]. Este procedimento pode ser implementado fazendo-se uso de **dígitos de guarda** (*guard digits*) [3] e ele permite estimar o erro em cada operação de ponto flutuante como

$$|\epsilon_{op}| \simeq \max(|\epsilon_X|, |\epsilon_Y|).$$

Desta forma, obtém-se o menor erro relativo possível na operação algébrica e este irá se propagar lentamente com o aumento do número de operações. Grande parte dos computadores em uso atualmente seguem o padrão IEEE, o qual exige o emprego dos dígitos de guarda. Caso este padrão não seja empregado, os erros decorrentes de operações de ponto flutuante aumentam de forma extremamente rápida com o número de operações. Neste último caso, para cada operação algébrica, obtém-se:

**Adição.** Resulta:

$$fl(X + Y) = fl(X) + fl(Y) = (X + Y) + (\delta_X + \delta_Y),$$

a qual pode ser escrita:

$$fl(X + Y) = (X + Y) \left( 1 + \frac{\delta_X + \delta_Y}{X + Y} \right) = (X + Y) \left( 1 + \frac{X\epsilon_X + Y\epsilon_Y}{X + Y} \right) \approx (X + Y)(1 + \epsilon_+),$$

sendo

$$\epsilon_+ \equiv \frac{\delta_X + \delta_Y}{fl(X) + fl(Y)}.$$

Ou seja, os erros absoluto e relativo do processo de soma de ponto flutuante são:

$$\begin{aligned} EA_+ &= |\delta_X + \delta_Y| \approx |fl(X)\epsilon_X + fl(Y)\epsilon_Y|, \\ ER_+ &= |\epsilon_+| = \left| \frac{\delta_X + \delta_Y}{fl(X) + fl(Y)} \right| = \left| \frac{\epsilon_X}{1 + fl(Y)/fl(X)} + \frac{\epsilon_Y}{1 + fl(X)/fl(Y)} \right|. \end{aligned}$$

Há três situações possíveis na última expressão acima:

1.  $fl(X) \gg fl(Y)$ . Neste caso, obtém-se

$$ER_+ \approx |\epsilon_X|.$$

2.  $fl(X) \ll fl(Y)$ . Neste caso,

$$ER_+ \approx |\epsilon_Y|.$$

3.  $\mathcal{O}[fl(X)] = \mathcal{O}[fl(Y)]$ .<sup>7</sup> Agora,

$$ER_+ \approx \frac{1}{2} |\epsilon_X + \epsilon_Y|.$$

<sup>7</sup>A notação  $\mathcal{O}(x)$  indica a ordem de grandeza de  $x$ . Uma definição rigorosa é apresentada na página 23.

Conclui-se, portanto, que  $ER_+ \sim \mathcal{O}[\max(\epsilon_X, \epsilon_Y)]$ .

**Subtração.** De forma similar ao caso anterior, resulta:

$$fl(X - Y) = fl(X) - fl(Y) = (X - Y) + (\delta_X - \delta_Y),$$

a qual pode ser escrita:

$$fl(X - Y) = (X - Y) \left( 1 + \frac{\delta_X - \delta_Y}{X - Y} \right) = (X - Y) \left( 1 + \frac{X\epsilon_X - Y\epsilon_Y}{X - Y} \right) \approx (X - Y)(1 + \epsilon_-),$$

sendo

$$\epsilon_- \equiv \frac{\delta_X - \delta_Y}{fl(X) - fl(Y)}.$$

Ou seja, os erros absoluto e relativo do processo de soma de ponto flutuante são:

$$\begin{aligned} EA_- &= |\delta_X - \delta_Y| = |fl(X)\epsilon_X - fl(Y)\epsilon_Y|, \\ ER_- &= |\epsilon_-| = \left| \frac{\delta_X - \delta_Y}{fl(X) - fl(Y)} \right| = \left| \frac{\epsilon_X}{1 - fl(Y)/fl(X)} + \frac{\epsilon_Y}{1 - fl(X)/fl(Y)} \right|. \end{aligned}$$

Considerando os mesmos casos anteriores,

1.  $fl(X) \gg fl(Y)$ . Neste caso,

$$ER_- \approx |\epsilon_X|.$$

2.  $fl(X) \ll fl(Y)$ . Neste caso,

$$ER_- \approx |\epsilon_Y|.$$

3.  $\mathcal{O}[fl(X)] = \mathcal{O}[fl(Y)]$ . Agora,  $1 - fl(Y)/fl(X) \ll 1$  e  $1 - fl(X)/fl(Y) \ll 1$ , resultando

$$ER_- \gg |\epsilon_X + \epsilon_Y|.$$

Este resultado mostra claramente como o erro relativo pode se tornar muito grande quando  $X \approx Y$ . Isto ocorre porque a subtração de dois números muito próximos entre si resulta em um número cuja representação ocorre nos últimos dígitos da mantissa, resultando em um grande erro de arredondamento.

**Multiplicação.** Neste caso,

$$fl(X \times Y) = fl(X) \times fl(Y) = (X + \delta_X) \times (Y + \delta_Y) = X \times Y + X \times \delta_Y + Y \times \delta_X + \delta_X \times \delta_Y.$$

Supondo que  $|\delta_X \times \delta_Y| \ll (|fl(X) \times \delta_Y|, |fl(Y) \times \delta_X|)$ , obtém-se

$$fl(X \times Y) \approx (X \times Y) + (fl(X) \times \delta_Y + fl(Y) \times \delta_X).$$

Por outro lado,

$$\begin{aligned} fl(X \times Y) &\approx (X \times Y) \left( 1 + \frac{fl(X) \times \delta_Y + fl(Y) \times \delta_X}{fl(X) \times fl(Y)} \right) \\ &= (X \times Y) \left( 1 + \frac{\delta_Y}{fl(Y)} + \frac{\delta_X}{fl(X)} \right) \equiv (X \times Y)(1 + \epsilon_\times). \end{aligned}$$

Assim,

$$\begin{aligned} EA_\times &= |fl(X) \times \delta_Y + fl(Y) \times \delta_X|, \\ ER_\times &= |\epsilon_\times| = \left| \frac{\delta_Y}{fl(Y)} + \frac{\delta_X}{fl(X)} \right| \approx |\epsilon_Y + \epsilon_X|. \end{aligned}$$

Portanto,  $ER_\times \sim \mathcal{O}[\max(\epsilon_X, \epsilon_Y)]$ , da mesma forma que a adição.

**Divisão.** Neste caso,

$$fl(X) = X + \delta_X,$$

$$\begin{aligned} fl(X \div Y) &= \frac{fl(X)}{fl(Y)} = \frac{X + \delta_X}{Y + \delta_Y} = \frac{X + \delta_X}{Y} \frac{1}{1 + \delta_Y/Y} \approx \frac{X + \delta_X}{Y} \left(1 - \frac{\delta_Y}{fl(Y)}\right) \\ &\approx \frac{X + \delta_X}{Y} - \frac{X\delta_Y + \delta_X\delta_Y}{fl(Y)^2} \approx \frac{X}{Y} + \frac{\delta_X}{fl(Y)} - \frac{fl(X)}{fl(Y)^2} \delta_Y, \\ fl(X \div Y) &\approx (X \div Y) + \left( \frac{fl(Y)\delta_X - fl(X)\delta_Y}{fl(Y)^2} \right). \end{aligned}$$

Ao passo que

$$\begin{aligned} fl(X \div Y) &= (X \div Y) \left(1 + \frac{Y}{X} \frac{fl(Y)\delta_X - fl(X)\delta_Y}{fl(Y)^2}\right) \approx (X \div Y) \left(1 + \frac{\delta_X}{fl(X)} - \frac{\delta_Y}{fl(Y)}\right) \\ &\equiv (X \div Y)(1 + \epsilon_{\div}). \end{aligned}$$

Então,

$$\begin{aligned} EA_{\div} &= \left| \frac{fl(Y)\delta_X - fl(X)\delta_Y}{fl(Y)^2} \right|, \\ ER_{\div} &= |\epsilon_{\div}| = \left| \frac{\delta_X}{fl(X)} - \frac{\delta_Y}{fl(Y)} \right| \approx |\epsilon_X - \epsilon_Y|. \end{aligned}$$

Ou seja,  $ER_{\div}$  é da mesma ordem de grandeza que  $ER_{\times}$ .

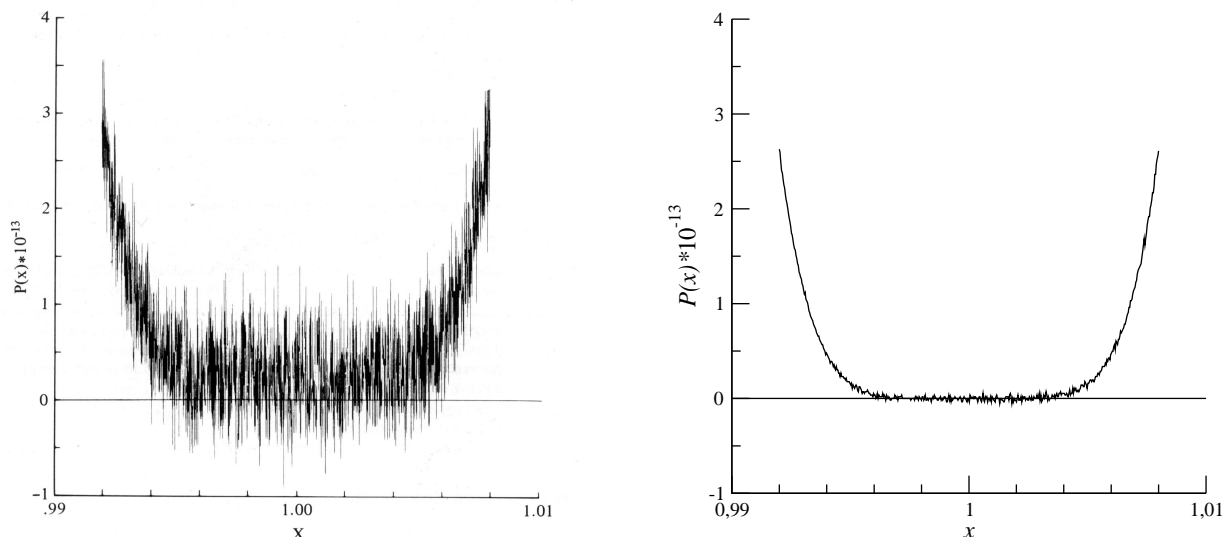
Para exemplificar o efeito deletério que os erros de arredondamento podem apresentar em um cálculos, em princípio, completamente exatos, *Rice (1983)* [4, Capítulo 3] mostra o gráfico do polinômio de sexto grau  $P(x) = (x - 1)^6$ , calculado na sua forma expandida

$$P(x) = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1.$$

A figura mostra  $P(x)$  traçado em torno de  $x = 1$ , em cujo ponto  $P(1) = 0$ . Entretanto, o cálculo realizado sem o emprego de dígitos de guarda (figura 1.4 esquerda) evidencia um efeito muito mais significativo dos erros de arredondamento nas operações básicas realizadas em números de ponto flutuante que o efeito observado quando se faz uso de um computador e um compilador [1] que seguem as normas IEEE 754, as quais prevêem o uso dos dígitos de guarda (figura 1.4 direita). O programa em Fortran que gerou figura 1.4 direita pode ser visto no programa 1.2.

**Listing 1.2:** Programa em Fortran que gerou a figura 1.4 direita.

```
program polynomial
implicit none
integer :: i
integer, parameter :: dp= 8
real(kind=dp) :: x,dx,px
!
open(unit=10, file='pol.dat')
dx= 0.016_dp/real(500-1,dp)
x= 0.992_dp
do i= 1, 500
    px= x**6 - 6*x**5 + 15*x**4 - 20*x**3 + 15*x**2 - 6*x + 1.0_dp
    write(10,*)x,px,px*1.0e13_dp
    x= x + dx
end do
end program polynomial
```



**Figura 1.4:** Esquerda: Cálculo do polinômio  $P(x)$  sem o uso de dígitos de guarda [4]. Direita: Cálculo de  $P(x)$  usando um compilador que segue as normas IEEE 754.

### 1.5.5.2 ERROS DE TRUNCAMENTO

Este erro ocorre quando se realiza um truncamento em um processo infinito. Um exemplo usualmente empregado consiste no cálculo do valor de uma função transcendental usando séries de McLaurin. Suponha-se que se deseja calcular o valor da função  $f(x) = e^x$  em  $x = 1$ , por exemplo. Neste caso,

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}.$$

Como não é possível ao computador a realização da soma infinita, a série deverá ser truncada para algum  $n > N$ . Neste caso, o resultado obtido irá diferir de  $e = 2,7182818284590452353602874\dots$  por um certo valor que dependerá do valor de  $N$ , isto é, do número de termos considerados na soma. Este tipo de erro soma-se ao erro de arredondamento considerado na seção 1.5.5.1. Em muitos casos, o erro de truncamento é exatamente a diferença entre o modelo matemático e o modelo numérico empregados. Uma das principais tarefas da análise numérica (e uma das mais difíceis) é a determinação de um valor máximo para o erro de truncamento.

Em muitos modelos numéricos existe um parâmetro que governa o erro de truncamento. Por exemplo, o número  $N$  de termos usados vindos de uma série infinita, ou o tamanho  $\Delta x$  usado numa fórmula de derivação numérica. Uma maneira comum e prática de estimar o erro de truncamento consiste em variar este parâmetro (tornando  $N$  maior ou  $\Delta x$  menor) e observar os resultados numéricos. Se os resultados computados convergirem a um certo número de dígitos significativos, então pode-se decidir que o erro de truncamento (juntamente com os demais tipos de erros) são pequenos o suficiente para produzir um resultado aceitável. Assim, muitas rotinas numéricas incluem um **teste de convergência** para decidir quando os resultados são aceitáveis. Infelizmente, não existe um teste de convergência padrão para qualquer problema numérico, uma vez que se trata de um problema matematicamente insolúvel. Em um nível intuitivo, isto significa que a convergência nunca pode ser testada de forma totalmente confiável; do ponto de vista matemático, para um dado teste de convergência que funciona em diversas situações, sempre existe um problema para o qual o teste irá falhar.

**ORDEM DE CONVERGÊNCIA.** Trata-se de uma maneira de medir o quanto o erro de truncamento vai a zero à medida que o parâmetro do método varia. Desta maneira, pode-se comparar a eficácia de distintos métodos. Em função do cálculo da ordem de convergência para diferentes métodos, pode-se obter diversos resultados, para distintos parâmetros, tais como:

- O método converge tal como  $1/N$ .
- O método converge tal como  $1/k^{3,5}$ .

- O método converge como  $h^2$ .
- O método converge exponencialmente, como  $e^{-N}$ , por exemplo.
- O erro de truncamento é da ordem  $1/N^5$ .
- A ordem do erro é  $h^4$ .
- A taxa de convergência é  $\log N/N$ .

O termo **ordem de convergência**, às vezes também denominado **taxa de convergência**, pode ter distintos significados. Em métodos iterativos, a ordem de convergência é calculada através de uma fórmula específica. Se o resultado é 2, por exemplo, então se diz que o método é de segunda ordem. Já um método de segunda ordem para resolver equações diferenciais possui outro significado. O termo **convergência linear** implica que o erro é reduzido (aproximadamente) por um fator constante em cada passo. A notação matemática para ordem de convergência, se um dado método converge tal como  $1/N^2$ , por exemplo, é:  $\mathcal{O}(1/N^2)$ . A notação  $\mathcal{O}$  é definida com segue:

Uma função  $f(x)$  é dita ser  $\mathcal{O}(g(x))$  à medida que  $x$  tende a  $L$  se

$$\lim_{x \rightarrow L} \left| \frac{f(x)}{g(x)} \right| < \infty.$$

A ordem de convergência pode ser complicada (por exemplo,  $h^{1,5}/\log h$ ) mas em alguns casos simples denominações especiais são empregadas. Se a ordem de convergência é uma potência inteira (por exemplo,  $h^2$ ,  $N^{-3}$ ,  $x^5$ ), então diz-se que a ordem de convergência é desta potência (2, 3 ou 5), ou que a convergência é de segunda, terceira ou quinta ordens. Por outro lado, diz-se *convergência logarítmica* ou *exponencial* se a ordem envolve uma função exponencial (como  $e^{-N}$ ) ou logarítmica (como  $1/\log N$ ).

**Exemplo 1.12.** As ordens de convergência de dois métodos de derivação numérica por diferença finita serão calculadas:

1. Diferença “adiantada” (*forward difference*):

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

2. Diferença “centrada” (*centered difference*):

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$

Em ambos os métodos, para um parâmetro  $h$  suficientemente pequeno e para um função  $f(x)$  bem comportada em torno de  $x$ , pode-se desenvolver a mesma em série de McLaurin:

$$f(x \pm h) = f(x) \pm f'(x)h + \frac{1}{2}f''(x)h^2 \pm \frac{1}{6}f'''(x)h^3 + \dots$$

Neste caso, para o método 1:

$$\frac{f(x+h) - f(x)}{h} = f'(x) + \frac{1}{2}f''(x)h + \dots$$

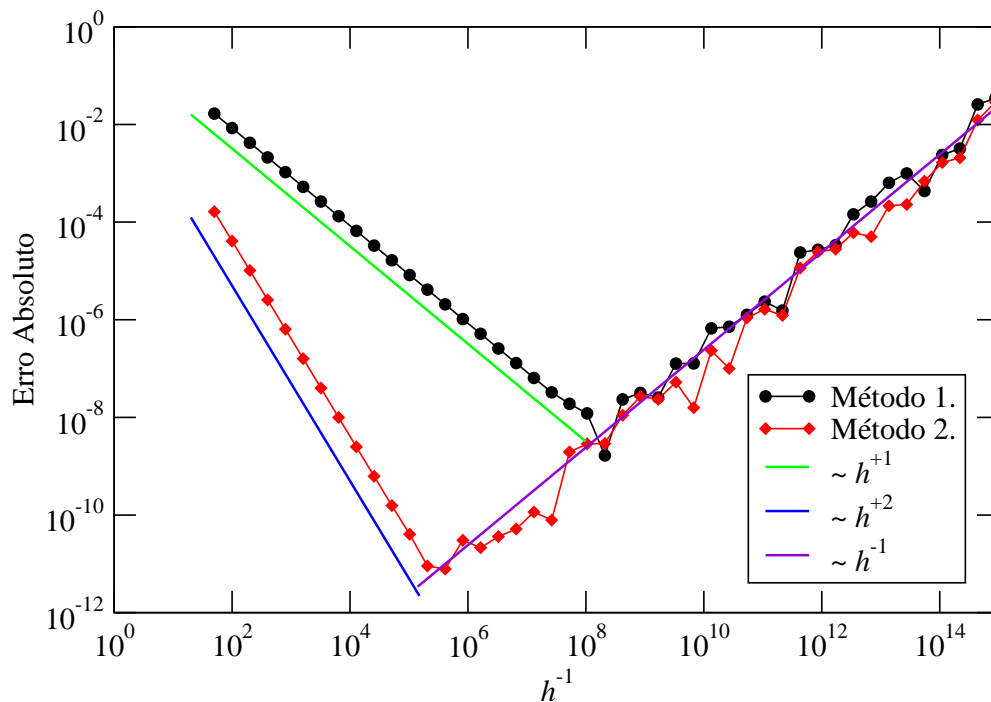
Ou seja, como o termo predominante é proporcional a  $h$ , a ordem de convergência deste método é  $\mathcal{O}(h)$ .

Para o método 2:

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + \frac{1}{6}f'''(x)h^2 + \dots,$$

ou seja, este método é da ordem  $\mathcal{O}(h^2)$ .

Como exemplo prático da aplicação destes métodos, deseja-se comparar os cálculos da derivada da função  $f(x) = \sin x^2$  no ponto  $x = 0,5$  pelos métodos 1 e 2, comparando-os com o valor correto da derivada:  $f'(x) = 2x \cos x^2$ , para  $x = 0,5$ . O erro absoluto *versus* o parâmetro  $h$  está traçado na figura 1.5. Os dados usados na figura foram calculados pelo programa 1.3.



**Figura 1.5:** Gráfico log-log do erro absoluto nos cálculos de derivadas usando métodos de diferenças finitas. As ordens de convergência são  $h$  e  $h^2$  para os métodos 1. e 2., respectivamente. Nota-se que erros de arredondamento acabam por arruinar totalmente a computação antes de a precisão de 15 dígitos ser atingida.

**Listing 1.3:** Programa em Fortran 95 que calculou as diferenças finitas da figura 1.5.

```

program derivadas
implicit none
integer, parameter :: dp= 8
integer :: i
real(kind= dp) :: h= 0.02_dp           ! h inicializado a 1/50.
real(kind= dp), parameter :: x= 0.5_dp ! Valor de x fixo.
real(kind= dp) :: df1, df2, f1
!
f1= 2.0_dp*x*cos(x*x) ! Valor correto da derivada em x.
open(unit=10, file='derivs.dat')
do i= 1, 45
  df1= (f(x+h) - f(x))/h           ! Cálculo método 1.
  df2= 0.5_dp*(f(x+h) - f(x-h))/h ! Cálculo método 2.
  write(10,*) 1.0_dp/h, abs(df1-f1), abs(df2-f1)
  h= 0.5_dp*h                     ! h é dividido por 2.
end do
CONTAINS
function f(x)
real(kind= dp) :: f
real(kind= dp), intent(in) :: x
f= sin(x*x)
return
end function f
end program derivadas

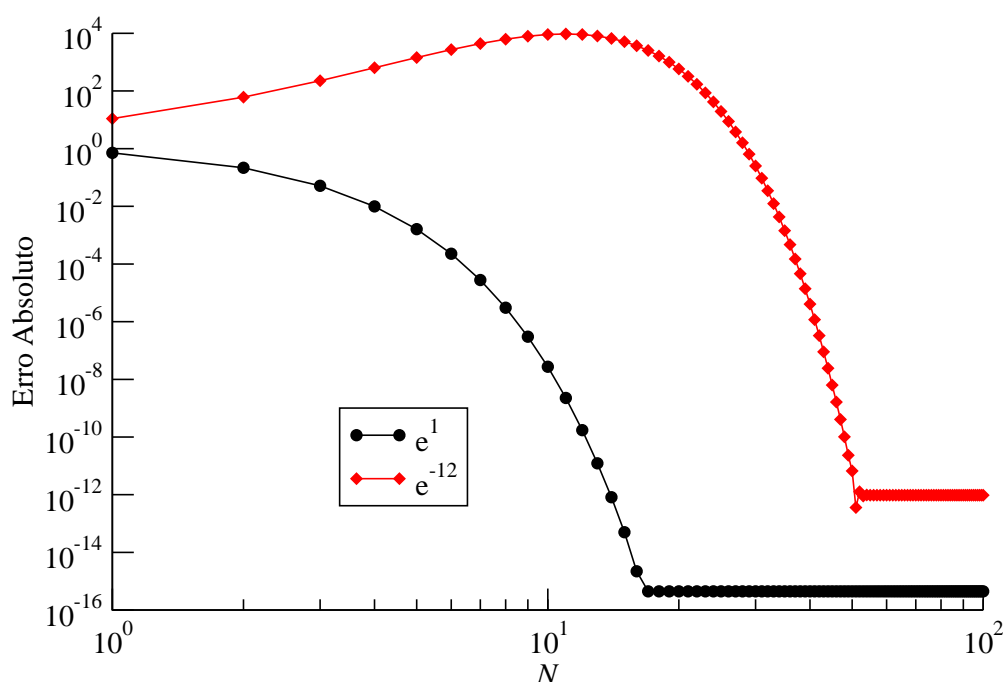
```

Os gráficos foram traçados em escala log-log para se verificar diretamente o expoente da taxa de convergência. Isto é, se  $\text{erro} = \alpha^k$ , então a inclinação da reta é  $k$ . Pode-se ver claramente que no início do processo iterativo, as taxas de convergência dos métodos 1 e 2 concordam com o valor previsto ( $h^1$  e  $h^2$ , respectivamente). Contudo, a partir de um determinado ponto os erros de arredondamento começam a se tornar mais importantes e o erro absoluto passa a variar a uma taxa proporcional a  $h^{-1}$ .

**Exemplo 1.13.** Deseja-se calcular o erro absoluto decorrente do truncamento no cálculo da série de McLaurin para a função  $e^x$ :

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \approx \sum_{n=0}^N \frac{x^n}{n!}.$$

O erro, calculado para os pontos  $x = 1$  e  $x = -12$ , em função do parâmetro  $N$  é apresentado na figura 1.6, enquanto que o correspondente programa em Fortran está no programa 1.4. Observa-se claramente como o erro absoluto inicia diminuindo rapidamente com o aumento de  $N$  (para  $x = -12$  isto ocorre para  $N > 11$ ), de uma forma não linear na escala log-log. Porém, eventualmente os erros de arredondamento que surgem na representação finita de termos cada vez menores na série impõe um limite inferior ao erro absoluto. Isto ocorre para  $x = 1$  antes que para  $x = -12$ .



**Figura 1.6:** Gráfico log-log do erro absoluto no cálculo da série de McLaurin para  $e^x$  quando a série é truncada na potência  $N$ . Erros de arredondamento limitam a precisão do resultado para  $x = 1$  antes que para  $x = -12$ .

**Listing 1.4:** Programa em Fortran 95 que calculou os pontos na figura 1.6.

```

program expo
implicit none
integer, parameter :: dp= 8
integer :: j, n
real(kind= dp), dimension(2), parameter :: x= (/1.0_dp,-12.0_dp/), &
ex= (/2.7182818284590452353602874713527_dp, &
6.1442123533282097586823081788055e-6_dp/)
real(kind= dp), dimension(2) :: soma, fator
!
open(unit=10,file='expo.dat')
do n= 1, 100
soma= 1.0_dp
fator= soma
do j= 1, n
fator= fator*x/real(j,dp)
soma= soma + fator
end do
write(10,*)n,abs(soma-ex)

```

```
end do
end program expo
```

### 1.5.5.3 ANÁLISE DE ERROS DE PONTO FLUTUANTE

Os exemplos da seção 1.5.5.2 mostraram como um cálculo relativamente simples pode ser completamente arruinado por erros de arredondamento. Isto mostra que um determinado método numérico sempre terá a sua utilidade limitada a um determinado valor do parâmetro de controle, de tal forma que uma posterior alteração no valor deste parâmetro terá sempre um efeito deletério na computação desejada. Caso o programador deseje uma precisão maior que o método pode oferecer, resta a ele buscar um método alternativo para atingir este objetivo.

Os erros de arredondamento tendem sempre a crescer com o número de operações de ponto flutuante realizadas. Como será este crescimento não se pode prever de antemão. Existem processos particularmente desafortunados, nos quais o erro de arredondamento cresce de forma linear ou através de uma lei de potência do tipo  $N^k$  ( $k > 1$ ), principalmente quando as operações realizadas são sempre do mesmo tipo, resultando em erros que sempre se somam. O resultado apresentado na figura 1.5 é um exemplo deste tipo de situação. A mesma tendência de erro sistemático ocorre quando o processo de representação finita é realizado por truncamento (ver seção 1.5.3).

Em um caso mais geral, os tipos de operações de ponto flutuante envolvidas são distintos, de forma a sempre haver a possibilidade que um erro será parcialmente compensado por outro, resultando em um aumento mais lento no erro total. Além disso, se for utilizado o processo de arredondamento par descrito na seção 1.5.3, os erros resultantes flutuarão de forma aleatória em valores positivos ou negativos, resultando num crescimento mais lento. Desta forma, a teoria de probabilidades indica que o erro deve variar:

$$\delta_{\text{arr}} \sim \sqrt{N} x_{\text{eps}},$$

onde  $x_{\text{eps}}$  é dado por (1.5) e  $N$  é um parâmetro que mede o número de termos considerados no método ou o número de operações de ponto flutuante realizadas.

O erro total será então a soma do erro de arredondamento e do erro de truncamento, decorrente da aproximação feita no algoritmo. De acordo com os exemplos e argumentos apresentados, uma forma comum de se encontrar os erros de truncamento é:

$$\delta_{\text{trunc}} \sim \frac{\alpha}{N^\beta}, \quad (\beta > 0).$$

Ou seja, teoricamente,  $\lim_{N \rightarrow \infty} \delta_{\text{trunc}} = 0$ . Então, o erro total será

$$\delta_{\text{total}} \sim \frac{\alpha}{N^\beta} + \sqrt{N} x_{\text{eps}}.$$

De acordo com este modelo,  $\delta_{\text{total}}$  deve começar diminuindo para  $N$  relativamente pequeno, porém, à medida que  $N$  aumenta, os erros de arredondamento tornam-se mais importantes e  $\delta_{\text{total}}$  começa a aumentar. O ponto onde este aumento se inicia pode ser estimado como

$$\ln N \sim -\frac{1}{\beta + 1/2} \ln \left( \frac{x_{\text{eps}}}{2\alpha\beta} \right).$$

Este comportamento pode ser claramente visto na figura 1.5.



# CÁLCULO DE FUNÇÕES

Neste capítulo serão apresentadas algumas técnicas para o cálculo de funções matemáticas. Em particular, de funções especiais, polinômios ou séries que não são usualmente suportados por rotinas intrínsecas fornecidas por compiladores comerciais. As técnicas aqui apresentadas tratam de implementar numericamente certas propriedades matemáticas conhecidas de funções especiais, tais como representações em séries de potências, representações integrais, representações em frações continuadas, entre outras.

## 2.1 CASOS BÁSICOS

É interessante iniciar a discussão com alguns casos simples de implementações de processos de cálculos que são necessários em diversos problemas distintos e que, portanto, podem ser implementados em rotinas. Os exemplos abordados envolvem as soluções de equações quadráticas ou cúbicas.

### 2.1.1 EQUAÇÃO QUADRÁTICA

Uma equação quadrática é uma equação na forma

$$ax^2 + bx + c = 0, \quad (2.1)$$

na qual os coeficientes  $a$ ,  $b$  e  $c$  são conhecidos e deseja-se encontrar os valores de  $x$  que satisfazem a equação.

A solução de (2.1) é bem conhecida.<sup>1</sup> Existem até duas soluções, denotadas por  $x_{\pm}$ , as quais são dadas por

$$x_{\pm} = \frac{-b \pm \sqrt{\Delta}}{2a} \quad (2.2a)$$

$$= \frac{2c}{-b \mp \sqrt{\Delta}}, \quad (2.2b)$$

onde  $\Delta = b^2 - 4ac$  é o *discriminante* da equação quadrática. As soluções (2.2a) e (2.2b) são *matematicamente* equivalentes (*i.e.*, ambas geram o mesmo número), mas não são *numericamente* equivalentes. Isto porque as fórmulas são sujeitas a cancelamentos catastróficos (seção 1.5.5.1).

**Exemplo 2.1.** Considere a implementação da fórmula (2.2a) em um modelo  $x[10, 2]$  de representação de números reais.<sup>2</sup> Suponha que

$$a = 2.00,$$

$$b = 20.02,$$

$$c = 0.20.$$

Em primeiro lugar, na forma normalizada,  $b = 2.00 \times 10^1$ . Então, computando a fórmula sem dígitos de guarda e mantendo os resultados na forma normalizada, a sequência de operações é:

$$\left\{ \begin{array}{l} b^2 = 4.00 \times 10^2 \\ 4ac = 1.6 \\ 2a = 4.0 \end{array} \right. \rightsquigarrow \begin{array}{l} b^2 - 4ac = 4.00 \times 10^2 - 1.6 = 4.00 \times 10^2 - 0.02 \times 10^2 = 3.98 \times 10^2 \\ \rightsquigarrow \sqrt{b^2 - 4ac} = 1.99 \times 10^1 \end{array}$$

<sup>1</sup>Embora a origem histórica da fórmula seja controversa.

<sup>2</sup>Ver seção 1.3.2. Para este exemplo não é necessário explicitar as quantidades  $e_{\min}$  e  $e_{\max}$ .

$$\sim \begin{cases} x_+ = \frac{-2.00 \times 10^1 + 1.99 \times 10^1}{4.0} = -3.00 \times 10^{-2} \\ x_- = \frac{-2.00 \times 10^1 - 1.99 \times 10^1}{4.0} = -9.98. \end{cases}$$

Ou seja, o modelo previu  $x_+ \approx -0.03$  e  $x_- \approx -9.98$ , quando o resultado correto é  $x_+ = -0.01$  e  $x_- = -10.00$ . O resultado obtido para  $x_-$  difere do exato por 0.2%, mas  $x_+$  difere do valor correto por 200%!

Por outro lado, empregando a fórmula (2.2b), obtém-se

$$x_+ = \frac{0.4}{-2.00 \times 10^1 - 1.99 \times 10^1} = -1.00 \times 10^{-2}, \quad x_- = \frac{0.4}{-2.00 \times 10^1 + 1.99 \times 10^1} = -4.0.$$

Ou seja, agora  $x_+$  foi estimado corretamente, mas  $x_-$  está completamente errado.

O exemplo acima demonstra um caso patológico de cancelamento catastrófico. No caso da fórmula (2.2), isto ocorre principalmente quando  $\{a, b, c\} \in \mathbb{R}$ ,  $4ac < 0$  e  $|b| \gg |4ac|$ , em cuja situação  $\sqrt{b^2 - 4ac} \approx |b|$ .

Uma maneira de implementar o cálculo das raízes de (2.1) de forma a minimizar o efeito dos erros de arredondamento é primeiro calculando

$$q = -\frac{1}{2} \left( b + s_q \sqrt{\Delta} \right), \text{ onde } s_q = \text{sgn}(b),$$

sendo  $\text{sgn}(x)$  a função sinal de  $x$ . Com isso, calcula-se

$$x_+ = \frac{c}{q}, \quad x_- = \frac{q}{a}.$$

Contudo, se  $\Delta < 0$ , o resultado de  $\sqrt{\Delta}$  é complexo. Esta situação deve ser levada em conta em um algoritmo. Além disso, se  $\{a, b, c\} \in \mathbb{C}$ , então as raízes em geral serão complexas. Neste último caso, a quantidade  $s_q$  no cálculo de  $q$  deve ser determinada de tal forma que

$$\text{Re} \left( b^* \sqrt{\Delta} \right) \geq 0,$$

onde  $\text{Re}$  denota a parte real de um número complexo e  $b^*$  denota a conjugação complexa de  $b$ .

O algoritmo 2.1 implementa o cálculo das raízes de (2.1) para o caso em que  $\{a, b, c\} \in \mathbb{R}$ . Este algoritmo é implementado em Fortran no programa 2.1, empregando precisão dupla.

---

**Algoritmo 2.1** Cálculo das soluções da equação quadrática (2.1) para coeficientes reais.

---

**Dados:**  $a, b$  e  $c$ , variáveis reais com  $a \neq 0$

Calcule  $\Delta = b^2 - 4ac$

**Se**  $\Delta \geq 0$  **então**

▷ (2 raízes reais)

Calcule, com álgebra real,

$$q = -\frac{1}{2} \left( b + \text{sgn}(b) \sqrt{\Delta} \right)$$

Calcule

$$x_1 = \frac{q}{a}, \quad x_2 = \frac{c}{q}$$

**Senão**

▷ (2 raízes complexas conjugadas)

Calcule, com álgebra complexa,

$$q = -\frac{1}{2} \left( b + \sqrt{\Delta} \right)$$

Calcule

$$x_1 = \frac{q}{a}, \quad x_2 = \frac{c}{q}$$

**Fim Se**

---

Listing 2.1: Implementa algoritmo 2.1 em Fortran, usando precisão dupla.

```

! ***** PROGRAMA QUADRATIC_EQ *****
! Calcula as raízes da equação quadrática
!       $ax^2 + bx + c = 0$ 
! Para o caso de coeficientes reais (precisão dupla).
!
! Autor: Rudi Gaelzer, IF-UFRGS
! Data: Agosto/2019.
!
program quadratic_eq
use, intrinsic :: iso_fortran_env, only: dp => real64
implicit none
real(dp), parameter :: zero= 0.0_dp, half= 0.5_dp, one= 1.0_dp, four= 4.0_dp
real(dp) :: a, b, c, disc, qr, x1, x2
complex(dp) :: qc, z1, z2

print*, 'Entre com os coeficientes da equação: '
write(*, fmt= '(a)', advance= 'no') 'a= ' ; read*, a
write(*, fmt= '(a)', advance= 'no') 'b= ' ; read*, b
write(*, fmt= '(a)', advance= 'no') 'c= ' ; read*, c
disc= b*b - four*a*c
print*, 'As raízes são: '
if(disc >= zero)then ! Raízes reais.
    qr= -half*(b + sign(one, b)*sqrt(disc))
    x1= qr/a ; x2= c/qr
    print '(2(a,g0))', 'x1= ', x1, ' x2= ', x2
else
    qc= -half*(b + sqrt(cmplx(disc, kind= dp)))
    z1= qc/a ; z2= c/qc
    print '(2(a,g0),a)', 'z1= (', real(z1,dp), ', ', aimag(z1), ')'
    print '(2(a,g0),a)', 'z2= (', real(z2,dp), ', ', aimag(z2), ')'
end if
end program quadratic_eq

```

### 2.1.2 EQUAÇÃO CÚBICA

Dadas as constantes  $c_0$ ,  $c_1$ ,  $c_2$  e  $c_3$  e uma incógnita  $x$ , a equação cúbica busca determinar os valores de  $x$  que satisfazem

$$a_3x^3 + a_2x^2 + a_1x + a_0 = 0. \quad (2.3)$$

A solução de (2.3) é conhecida e a sua derivação é atribuída ao matemático italiano Gerolamo Cardano (1501 – 1576). As fórmulas envolvidas no cálculo das raízes podem ser vistas em [8, sec. 1.11(iii)] e [9, sec. 4.43].

Se os coeficientes forem todos reais, esta equação ou possui três raízes reais ou uma raiz real e duas complexas conjugadas. O procedimento para o cálculo das raízes neste caso é fornecido pelo algoritmo 2.2.

## 2.2 POLINÔMIOS E FUNÇÕES RACIONAIS

Um polinômio de grau  $N$ , tal como

$$P_N(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{N-1}x^{N-1} + a_Nx^N, \quad (2.4)$$

é representado numericamente a partir de um vetor que armazena os seus coeficientes:  $a(i)$ , com  $i = 0, 1, \dots, N$ . A regra básica para o cálculo de um polinômio de grau  $N$  consiste em *nunca* implementar o cálculo de  $P_N(x)$  pela expressão em (2.4), pois (i) ocorrerá uma contaminação

**Algoritmo 2.2** Cálculo das soluções da equação cúbica (2.3) para coeficientes reais.

**Dados:**  $a_0, a_1, a_2$  e  $a_3$ , variáveis reais com  $a_3 \neq 0$

Calcule  $a, b$  e  $c$  tais que

$$a = \frac{a_2}{a_3}, \quad b = \frac{a_1}{a_3}, \quad c = \frac{a_0}{a_3}$$

Calcule  $q$  e  $r$  tais que

$$q = \frac{a^2 - 3b}{9}, \quad r = \frac{2a^3 - 9ab + 27c}{54}$$

**Se**  $r^2 < q^3$  **então**

▷ (3 raízes reais)

Calcule

$$\theta = \cos^{-1} \left( \frac{r}{\sqrt{q^3}} \right)$$

Calcule

$$x_1 = -2\sqrt{q} \cos \left( \frac{\theta}{3} \right) - \frac{a}{3} \quad x_1 = -2\sqrt{q} \cos \left( \frac{\theta + 2\pi}{3} \right) - \frac{a}{3} \quad x_1 = -2\sqrt{q} \cos \left( \frac{\theta - 2\pi}{3} \right) - \frac{a}{3}$$

**Senão**

▷ (1 raiz real e 2 complexas conjugadas)

Calcule

$$p = -\operatorname{sgn}(r) \left[ |r| + \sqrt{r^2 - q^3} \right]^{1/3}, \quad s = \begin{cases} q/p, & (p \neq 0) \\ 0, & (p = 0) \end{cases}$$

Calcule

$$\begin{aligned} x_1 &= p + s - \frac{a}{3} \text{ (raiz real)} \\ x_2 &= -\frac{1}{2}(p + s) - \frac{a}{3} + i\frac{\sqrt{3}}{2}(p - s) \text{ (raiz complexa 1)} \\ x_3 &= -\frac{1}{2}(p + s) - \frac{a}{3} - i\frac{\sqrt{3}}{2}(p - s) \text{ (raiz complexa 2)} \end{aligned}$$

**Fim Se**

acentuada de erros de arredondamento e, principalmente, (ii) computações de  $x^i$  para valores crescentes de  $0 \leq i \leq N$  rapidamente resultarão em overflow ou underflow mesmo que  $P_N(x)$  seja representável.

A maneira numericamente correta de calcular  $P_N(x)$  pode ser obtida pela sequência de passos:

$$\begin{aligned} P_N(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{N-1}x^{N-1} + a_Nx^N \\ &= a_0 + x(a_1 + a_2x + a_3x^2 + \cdots + a_{N-1}x^{N-2} + a_Nx^{N-1}) \\ &= a_0 + x[a_1 + x(a_2 + a_3x + \cdots + a_{N-1}x^{N-3} + a_Nx^{N-2})] \\ &\vdots \\ &= a_0 + x[a_1 + x\{a_2 + x[a_3 + \cdots + x(a_{N-1} + a_Nx)]\}], \end{aligned}$$

onde então a expressão é calculada a partir dos parênteses mais internos para os mais externos. Esta expressão é facilmente implementada por um laço, onde em cada iteração é adicionado o termo  $a_{i-1} + a_ix$  a um acumulante, variando-se o índice  $i$  de  $N$  a zero. Este é o **algoritmo de Horner**. O algoritmo 2.3 implementa esta sequência de passos.

O algoritmo de Horner pode ser facilmente modificado para calcular simultaneamente a de-

**Algoritmo 2.3** Algoritmo de Horner para o cálculo do polinômio  $P_N(x)$ .

**Dados:**  $N \geq 0$ ,  $\{a_i, i = 0, \dots, N\}$  e  $x$

$p = a_N$

**Para**  $i = N - 1, N - 2, \dots, 0$  **faça**

$p = px + a_i$

▷ (Se  $N = 0$ , não executa laço)

**Fim Laço**

derivada de  $P_N(x)$ , pois

$$P'_N(x) = a_1 + 2a_2x + 3a_3x^2 + \dots + Na_Nx^{N-1}.$$

Na verdade, o algoritmo modificado permite calcular simultaneamente as  $0 \leq m \leq N$  derivadas de  $P_N(x)$ .

O programa 2.2 implementa em Fortran o cálculo de  $P_N(x)$  e suas derivadas pelo algoritmo de Horner. A unidade que chama a sub-rotina `poly_dn` deve fornecer o vetor `c` de forma `[0:N]` contendo os coeficientes do polinômio e o valor do argumento  $x$ . A unidade deve também alocar o vetor `pd` com forma `[0:m]` ( $0 \leq m \leq N$ ), o qual irá retornar o valor de  $P_N(x)$  em `pd(0)` e suas primeiras  $m$  derivadas em `pd(1:m)`. Observa-se que o código fornece o resultado correto mesmo se inadvertidamente for alocado um vetor `pd` com  $m > N$ .

**Listing 2.2:** Implementa o algoritmo de Horner para o cálculo de  $P_N(x)$  e suas derivadas.

```
! ***** SUBROTINA POLY_DN *****
! Calcula o polinômio P_N(x) e suas nd primeiras derivadas.
! Método: algoritmo de Horner.
!
! Argumentos:
!   c: Vetor de forma [0:N] com os coeficientes.           (Entrada)
!   x: Argumento do polinômio.                             (Entrada)
!   pd: Vetor de forma [0:nd] contendo P_N(x) e suas nd derivadas. (Saída)
!
! Autor: Rudi Gaelzer, IF-UFRGS.
! Data: Setembro/2019.
! Obs: baseado na rotina ddpoly do Numerical Recipes.
subroutine poly_dn(c, x, pd)
use, intrinsic :: iso_fortran_env, only: dp => real64
implicit none
real(dp), intent(in) :: x
real(dp), intent(in), dimension(0:) :: c
real(dp), intent(out), dimension(0:) :: pd
integer :: i, j, nc, nd
real(dp), parameter :: zero= 0.0_dp, one= 1.0_dp
real(dp) :: co
nc= size(c) - 1 ; nd= size(pd) - 1
pd(0)= c(nc) ; pd(1:nd)= zero
do i= nc - 1, 0, -1
    do j= min(nd, nc - i), 1, -1
        pd(j)= pd(j)*x + pd(j - 1)
    end do
    pd(0)= pd(0)*x + c(i)
end do
co= one
do i= 2, nd ! Correção para derivadas de ordem nd > 1.
    co= co*i
    pd(i)= co*pd(i)
end do
return
end subroutine poly_dn
```

## 2.3 SÉRIES CONVERGENTES

O cálculo de séries é talvez uma das técnicas mais empregadas para a representação de funções. Séries são também com frequência empregadas para representar soluções de equações diferenciais.

Uma série é definida como o limite de uma sequência de somas parciais. Seja a sequência de valores  $\{S_N\}_0^\infty = S_0, S_1, S_2, \dots, S_N, \dots$ , sendo que cada termo  $S_N$  da sequência é o resultado da soma parcial

$$S_N = \sum_{n=0}^N u_n, \quad (N = 0, 1, 2, \dots),$$

onde as quantidades  $\{u_n\}$  são supostas conhecidas. Se a sequência das somas parciais converge para um valor finito no limite  $N \rightarrow \infty$ , então a **série**

$$S \doteq \lim_{N \rightarrow \infty} S_N \equiv \sum_{n=1}^{\infty} u_n$$

é denominada uma **série convergente**. Caso isto não aconteça,  $S$  será uma **série divergente**.

Existem diversas técnicas para testar se uma série é convergente. Uma condição necessária (mas não suficiente) para tanto é que  $\lim_{n \rightarrow \infty} u_n = 0$ ; ou seja, cada termo da sequência de somas parciais irá diferir do anterior por uma quantidade progressivamente menor à medida que  $N \rightarrow \infty$ . Esta condição não é suficiente e a notória *série harmônica*  $\sum_{n=1}^{\infty} 1/n$  é um exemplo de série divergente, embora  $u_n = 1/n$  satisfaça o critério.

Um teste absoluto de convergência de uma série é o **critério de Cauchy**. Dada uma série  $S$ , se para qualquer quantidade  $\varepsilon > 0$  existir um número inteiro positivo  $N$  tal que

$$|S_i - S_j| < \varepsilon, \text{ para todos } i, j > N,$$

então a série é convergente. Ou seja, a série converge porque o critério é satisfeito mesmo para  $\varepsilon$  arbitrariamente próximo de zero.

Outros testes de convergência de séries podem ser vistos em *Arfken (2013)* [10, sec. 1.1]. Aqui será destacado o **teste da razão**

$$\lim_{n \rightarrow \infty} \left| \frac{u_{n+1}}{u_n} \right| = \begin{cases} < 1, & \text{(série convergente)} \\ > 1, & \text{(série divergente)} \\ = 1, & \text{(indeterminado)}. \end{cases} \quad (2.5)$$

O teste da razão é usualmente o primeiro (e o mais simples) a ser empregado para verificar a convergência de uma série. Se o seu resultado for indeterminado, outro teste deve ser empregado.

Nesta seção serão abordados brevemente alguns métodos relacionados ao cálculo de séries convergentes. De particular importância são as *séries de potências*, pois estas fornecem uma ferramenta prática para o cálculo do valor de uma função. Os métodos apresentados aqui estarão restritos a funções de uma variável (real ou complexa).

### 2.3.1 SÉRIES DE TAYLOR E DE POTÊNCIAS

A série de Taylor de uma função  $f(x)$  é uma ferramenta útil para o cálculo da função, desde que a mesma possua derivadas em todas as ordem em  $x$ .

Considera-se primeiro o caso de funções reais de argumentos reais. Sejam os pontos  $\underline{x}$  e  $\bar{x}$  ( $\underline{x} < \bar{x}$ ), os quais definem o intervalo  $L = (\underline{x}, \bar{x}) \subset \mathbb{R}$  e um ponto  $x = a$  contido em  $L$ ; isto é,  $\underline{x} \leq a \leq \bar{x}$ . Se uma determinada função  $f(x)$  possuir derivadas no mínimo até a ordem  $n > 0$  em qualquer ponto do intervalo  $L$ , a função pode ser calculada em qualquer ponto  $x \in L$  por

$$f(x) = S_N(x) + R_N(x), \quad S_N(x) = \sum_{n=0}^N \frac{f^{(n)}(a)}{n!} (x-a)^n, \quad (2.6)$$

onde  $f^{(n)}(a)$  denota a  $n$ -ésima derivada da função em  $x = a$  e

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} (x-a)^{n+1}, \text{ para algum } \underline{x} \leq \xi \leq \bar{x}.$$

A quantidade  $R_N(x)$  é denominada o *resto* da soma  $S_N(x)$ , o qual é necessário para o cálculo exato de  $f(x)$ .

Se a função  $f(x)$  possuir derivadas em qualquer ordem em  $L^3$  e  $\lim_{n \rightarrow \infty} R_n = 0$ , então o limite da sequência de somas parciais  $S_N(x)$  ( $N = 0, 1, \dots$ ) resulta na **série de Taylor de  $f(x)$  em torno de  $a$**

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x-a)^n,$$

a qual será uma série convergente porque satisfaz o critério de Cauchy. Quando a série de Taylor é desenvolvida em torno de  $a = 0$ , a série

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} x^n$$

é também conhecida como a **série de Maclaurin de  $f(x)$** . Exemplos bem conhecidos de séries de Maclaurin de funções elementares são

$$\begin{aligned} e^x &= \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \dots \\ \sin x &= \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \dots \\ \cos x &= \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \dots \end{aligned}$$

Uma função  $f(x)$  que pode ser desenvolvida por uma série de Taylor em um intervalo  $L \subseteq \mathbb{R}$  é dita ser uma **função analítica em  $L$** . Pode-se demonstrar que se uma função é analítica no intervalo  $L$ , sua série de Taylor em  $L$  é única.

Uma série de Taylor ou Maclaurin é um caso particular de uma **série de potências**

$$f(x) = \sum_{n=0}^{\infty} c_n (x-a)^n, \quad (2.7)$$

onde as quantidades  $\{c_n; n = 0, 1, \dots\}$ , supostas conhecidas, são os *coeficientes* da série. Ou seja, para uma série de Taylor,  $c_n = f^{(n)}(a)/n!$ . Em alguns casos, é possível determinar-se a série de potências de uma função  $f(x)$  conhecendo-se os seus coeficientes, sem que se conheça de antemão os valores das derivadas  $f^{(n)}(a)$ . Isto ocorre, por exemplo, quando se emprega o método de séries de potências para a solução de equações diferenciais [10, sec. 7.5].

Uma propriedade de uma série de potências é o seu **raio de convergência  $R$** , definido a partir do teste da razão (2.5). Dada a série de potências (2.7) de  $f(x)$  desenvolvida em torno de  $x = a$ . Se existir uma quantidade  $R$  não nula, obtida por

$$\frac{1}{R} = \lim_{n \rightarrow \infty} \left| \frac{c_{n+1}}{c_n} \right|, \quad (2.8)$$

então a série garantidamente converge no intervalo  $-R + a < x < R + a$ . Para verificar se a série converge nos pontos  $x = -R + a$  ou  $x = R + a$ , outros testes podem ser necessários.

Os raios de convergência das séries das funções  $e^x$ ,  $\sin x$  e  $\cos x$  são todos infinitos ( $R \rightarrow \infty$ ); portanto, suas séries convergem para qualquer  $x \in \mathbb{R}$ . Já a **série geométrica**

$$\sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \dots \stackrel{|x| < 1}{=} \frac{1}{1-x} \quad (2.9)$$

possui raio de convergência  $R = 1$ ; isto é, a série somente converge para  $f(x) = (1-x)^{-1}$  no intervalo  $|x| < 1$ .

Considerando agora funções complexas, isto é, funções  $f(z) \in \mathbb{C}$  para  $z \in \mathbb{C}$ , as suas propriedades usualmente generalizam as propriedades de funções reais. Discussões a respeito de funções complexas podem ser vistas em [10, cap. 11] ou [14, cap. 2].

<sup>3</sup>Sendo então denominada uma *função suave*.

Em particular, se uma função  $f(z)$  possui derivada e é unívoca em uma região  $D \subseteq \mathbb{C}$  centrada em  $z = z_0$ , então ela é dita ser **analítica em  $D$**  (ou **holomórfica** ou **regular**). Se a função  $f(z)$  é analítica em  $D$ , então ela sempre pode ser desenvolvida em uma série de Taylor em torno de  $z_0$ ,

$$f(z) \stackrel{z \in D}{=} S_{z_0}(z) = \sum_{n=0}^{\infty} \frac{f^{(n)}(z_0)}{n!} (z - z_0)^n \equiv \sum_{n=0}^{\infty} c_n (z - z_0)^n, \quad (2.10)$$

a qual é única em  $D$ , sendo que agora  $c_n \in \mathbb{C}$ .

O raio de convergência da série (2.10) pode ser também determinado por (2.8). Uma regra prática para se determinar o raio de convergência de  $S_{z_0}(z)$  consiste em “medir” a distância de  $z_0$  ao ponto singular de  $f(z)$  mais próximo. Chamando de  $z_s$  este ponto singular, o raio de convergência será  $R = |z_0 - z_s|$ . **Funções inteiras** possuem singularidade somente em  $|z_s| \rightarrow \infty$ ; isto é, suas séries de Taylor convergem sobre todo o plano complexo.

Retornando agora ao problema prático de empregar uma série de potências para o cálculo de uma função

$$f(x) = S_{x_0}(x) = \sum_{n=0}^{\infty} c_n (x - x_0)^n \quad (x \text{ real ou complexo}),$$

está claro, a partir da discussão realizada na seção 1.5.5.2, que nunca será computacionalmente possível atingir-se o limite da sequência das somas parciais (ou a soma dos infinitos termos de  $S_{x_0}$ ), porque sempre ocorrerá um erro de truncamento. Em outras palavras, um computador somente será capaz de fornecer uma aproximação a  $f(x)$  a partir da série  $S_{x_0}(x)$ , a qual será uma das somas parciais apresentadas em (2.6).

Supondo que a função  $f(x)$  possa ser representada por uma série de potências  $S_{x_0}(x)$  em torno de  $x_0$  e denotando-se por

$$S_{N,x_0}(x) = \sum_{n=0}^N c_n (x - x_0)^n$$

a  $N$ -ésima soma parcial na sequência que leva a  $S_{x_0}(x)$  quando  $N \rightarrow \infty$ , do ponto de vista de métodos numéricos, a questão que se coloca é: *para qual valor  $N$  a soma  $S_{N,x_0}(x)$  irá satisfazer um determinado critério de tolerância para uma aproximação aceitável da função  $f(x)$* ? O critério de tolerância em questão pode ser um dos seguintes:<sup>4</sup>

- $|f(x) - S_{N,x_0}(x)| \leq EA_f(N, x_0)$ , sendo  $EA_f(N, x_0)$  o erro absoluto máximo permitido à aproximação.
- $|1 - S_{N,x_0}(x)/f(x)| \leq ER_f(N, x_0)$ , sendo  $ER_f(N, x_0)$  o erro relativo máximo permitido.
- $S_{N,x_0}(x)$  possui os  $n_{dc}$  primeiros dígitos corretos.

Observa-se que os dois últimos critérios acima estão relacionados entre si, porque, de acordo com (1.7), na base  $b = 10$  temos

$$-1 - \log(ER_f) < n_{dc} \leq -\log(ER_f).$$

Por exemplo, se  $ER_f = 10^{-5}$ , então  $S_{N,x_0}(x)$  será uma aproximação para  $f(x)$  com no mínimo 4 e no máximo 5 dígitos corretos. Contudo, a validade destes critérios deve ser analisada cuidadosamente quando as somas parciais são calculadas em valores de  $x$  próximos às raízes de  $f(x)$ .

Se a série for **uniformemente convergente** no domínio  $D$ , então para qualquer  $\varepsilon > 0$  arbitrariamente pequeno sempre existirá um número  $N$  a partir do qual

$$|S_{x_0}(x) - S_{n,x_0}(x)| = |R_{n,x_0}(x)| < \varepsilon, \text{ para todo } x \in D \text{ e } n \geq N.$$

Neste caso, a quantidade  $\varepsilon$  será o erro absoluto  $EA_f(n, x_0)$  e esta pode ser tomada como o critério para interromper o cálculo das somas parciais.

Por outro lado, assumindo que a tolerância para a aproximação de  $f(x)$  possa ser medida com o erro relativo, observa-se de (2.6) que

$$ER_f(N, x_0) \geq \left| \frac{S_{x_0}(x) - S_{N,x_0}(x)}{S_{x_0}(x)} \right| = \left| \frac{R_{N,x_0}(x)}{S_{x_0}(x)} \right|.$$

<sup>4</sup>Ver seções 1.5.2 e 1.5.4.



Ou seja, é necessário conhecer a razão entre o resto da  $N$ -ésima soma parcial e o valor da série. Novamente neste caso, existe um caso especial de série convergente para o qual o seguinte teorema se aplica.

**Teorema.** *Seja a série*

$$S = \sum_{j=0}^{\infty} u_j = S_n + R_n, \quad S_n = \sum_{j=0}^n u_j.$$

*Se existir um número  $n$  tal que para todo  $j \geq n$ ,  $|u_{j+1}| \leq \varepsilon |u_j|$  para  $0 < \varepsilon < 1$ , então*

$$|R_n| \leq \frac{|u_{n+1}|}{1 - \varepsilon} \leq \frac{\varepsilon |u_n|}{1 - \varepsilon}.$$

*Em particular, se  $\varepsilon < 1/2$ , então  $|R_n| < |u_n|$ .*

**Demonstração.** O resto da série  $R_n = S - S_n = \sum_{j=n+1}^{\infty} u_j$  é dado pela soma dos termos  $u_{n+1}, u_{n+2}, \dots$ . Se  $|u_{j+1}| \leq \varepsilon |u_j|$  para  $j \geq n$ , então, em  $R_n$ ,  $|u_{n+2}| \leq \varepsilon |u_{n+1}|$ ,  $|u_{n+3}| \leq \varepsilon |u_{n+2}| \leq \varepsilon^2 |u_{n+1}|$ , etc. Por indução, conclui-se que  $|u_{n+m+1}| \leq \varepsilon^m |u_{n+1}|$  ( $m \geq 0$ ). Ou seja, em  $R_n$ ,  $|u_j| \leq \varepsilon^{j-n-1} |u_{n+1}|$  ( $j \geq n+1$ ). Portanto,

$$|R_n| = \left| \sum_{j=n+1}^{\infty} u_j \right| \leq \sum_{j=n+1}^{\infty} |u_j| \leq |u_{n+1}| \sum_{j=n+1}^{\infty} \varepsilon^{j-n-1} \stackrel{(2.9)}{=} \frac{|u_{n+1}|}{1 - \varepsilon} \leq \frac{\varepsilon |u_n|}{1 - \varepsilon}.$$

Por fim, se  $\varepsilon < 1/2$ , então  $\varepsilon/(1 - \varepsilon) < 1$ , de onde segue que  $|R_n| < |u_n|$ .  $\square$

Como a diferença entre duas somas parciais consecutivas é  $S_{n+1} - S_n = u_{n+1}$ , resulta que para as séries onde este teorema se aplica para algum  $n \geq N$ , o valor absoluto do resto irá necessariamente diminuir de forma monotônica com o aumento de  $n$ . Neste caso, empregando como critério de tolerância o cálculo do erro relativo  $ER_f(N, x_0)$ , pode-se assumir, com um grande grau de confiança, o critério de convergência

$$\left| \frac{u_{n,x_0}(x)}{S_{n,x_0}(x)} \right| \leq ER_f(n, x_0), \quad (n \geq N).$$

O algoritmo 2.4 implementa o cálculo de  $f(x)$  quando a sua série de potências converge de maneira uniforme.

---

**Algoritmo 2.4** Cálculo de  $f(x)$  por meio de uma série uniformemente convergente.

---

**Dados:**  $\{c_n, n = 0, 1, 2, \dots\}$ , os coeficientes da série  $S(x) = \sum_{n=0}^{\infty} c_n x^n / n!$

$\varepsilon$ , a tolerância máxima para a aproximação  $S_N(x)$  ( $0 < \varepsilon \ll 1$ )

$x$ , o argumento de  $S(x)$

**Inicialize:**  $p = 1, s = c_0, i = 1$

**Repita**

$p = p * x / i$

$t = c_i * p$

$s = s + t$

$i = i + 1$

**Até que**  $|t/s| \leq \varepsilon$

**Retorne:**  $S(x) \approx s$

---

Observa-se que o algoritmo supõe que os coeficientes  $\{c_n\}$  da série são conhecidos ou que podem ser calculados a partir de alguma relação de recorrência. Neste último caso, linhas adicionais dentro do laço devem ser adicionadas para o cálculo de  $c_i$  a cada iteração. Observa-se também que o algoritmo escreve os coeficientes da série como  $u_n = c_n x^n / n!$  para realizar o cálculo de  $r_n(x) = x^n / n!$  também através da relação de recorrência  $r_n(x) = x r_{n-1}(x) / n$  (para  $n > 1$ , com  $r_0 = 1$ ). Este método reduz o efeito dos erros de arredondamento de uma maneira semelhante à estratégia adotada no algoritmo 2.3 para o cálculo de um polinômio.

O algoritmo 2.4 garantidamente funciona para uma série que apresenta convergência uniforme e monotônica, o que não é em geral verdade. Se a série convergir de forma lenta ou

alternada, uma análise mais detalhada do comportamento de  $R_n$  pode ser necessária. Uma tentativa de aplicação do algoritmo 2.4 para séries cuja convergência não é conhecida de antemão pode ser realizada se o teste de convergência for realizado, a cada iteração, para um número  $M$  ( $M > 1$ ) de somas parciais prévias. Assim, o algoritmo somente aceita a última soma parcial como uma boa aproximação para  $S(x)$  se o teste for satisfeito para todas as  $M+1$  somas parciais.

O programa 2.3 apresenta um exemplo de aplicação do algoritmo 2.4 para o cálculo da função  $f(x) = \sin x$ . O programa contém a função interna `my_sin(x)` a qual realiza o teste de convergência em três somas parciais consecutivas e somente aceita a última soma como aproximação para  $\sin x$  se todos os testes forem bem sucedidos. Escrevendo a série da função como  $\sin x = x \sum_n u_n(x)$ , observa-se que  $u_n(x)$  satisfaz a relação de recorrência

$$u_n(x) = \frac{(-1)^n x^{2n}}{(2n+1)!} \Rightarrow u_{n+1}(x) = -\frac{x^2 u_n(x)}{(2n+3)(2n+2)}, \quad (n = 0, 1, 2, \dots).$$

Inicialmente calcula-se os dois primeiros termos da série  $u_0(x) = 1$  e  $u_1(x) = -x^2/6$  e os termos para  $n \geq 2$  são então calculados no laço, no qual a relação de recorrência é empregada. O programa imprime também o valor da função intrínseca `sin(x)` para comparação.

**Listing 2.3:** Implementa algoritmo 2.4 em Fortran para o cálculo da função  $\sin x$ .

```
! ***** PROGRAMA MY_SIN *****
! Calcula uma aproximação para a função sen(x)
!
! Autor: Rudi Gaelzer, IF-UFRGS
! Data: Setembro/2019.
!
program my_sin_p
use, intrinsic :: iso_fortran_env, only: dp => real64
implicit none
real(dp) :: x
do
  write(*, '(a)', advance='no') 'x= ' ; read*, x
  print '(a,g0)', 'my_sin(x)= ', my_sin(x)
  print '(a,g0)', 'F_sin(x)= ', sin(x)
end do
CONTAINS
function my_sin(x)
real(dp) :: my_sin
real(dp), intent(in) :: x
real(dp), parameter :: zero= 0.0_dp, one= 1.0_dp, two= 2.0_dp
real(dp), parameter :: tol= epsilon(one)
real(dp) :: x2
real(dp), dimension(2) :: f2np1
real(dp), dimension(0:2) :: vs, vt
x2= x*x
vt= [ one, -x2/6.0_dp, zero ]
vs= [ one, one + vt(1), zero ]
f2np1= [ 5.0_dp, 4.0_dp ]
do ! Inicia laço a partir do termo n = 2
  vt(2)= -vt(1)*x2/product(f2np1)
  vs(2)= vs(1) + vt(2)
  if(all(abs(vt) <= tol*abs(vs))) exit
  f2np1= f2np1 + two
  vt= eoshift(vt, 1) ; vs= eoshift(vs, 1) ! vs(1) -> vs(0), vs(2) -> vs(1)
end do
my_sin= x*vs(2)
return
end function my_sin
end program my_sin_p
```

### 2.3.2 ACELERAÇÃO DA CONVERGÊNCIA DE SÉRIES

O algoritmo 2.4, o qual implementa o cálculo de uma série de potências convergente dada uma certa tolerância no resultado, funciona bem para uma série uniformemente convergente. Quando isto não ocorre os termos da sequência  $\{S_N\}_0^\infty$  podem convergir lentamente para o limite  $S$ . Neste caso, existem alguns métodos que podem acelerar a convergência da série ao substituir a sequência  $\{S_N\}_0^\infty$  por uma outra sequência  $\{S'_N\}_0^\infty$  que irá convergir muito mais rapidamente para  $S$ . Esta nova sequência é obtida a partir de uma determinada transformação aplicada à sequência  $\{S_N\}_0^\infty$  original. Algumas dessas transformações serão apresentadas abaixo.

#### 2.3.2.1 TRANSFORMAÇÃO DE EULER

A transformação de Euler foi criada em 1755 pelo matemático Suíço Leonhard Euler (1707 — 1783) e é particularmente útil para **séries alternantes**, nas quais os termos das somas parciais alternam em sinal; isto é, para uma série do tipo

$$S = \sum_{n=0}^{\infty} (-1)^n u_n(x),$$

com  $u_n(x) > 0$  para um certo  $n \geq N$ .

Com as condições acima, a série pode ser calculada como

$$S = \sum_{n=0}^{N-1} (-1)^n u_n(x) + \sum_{n=0}^{\infty} \frac{(-1)^n}{2^{n+1}} [\Delta^n u_N(x)],$$

para a qual, se  $N = 0$  a primeira soma não é realizada. A quantidade  $\Delta^n u_N(x)$  é o  $n$ -ésimo **operador de diferença adiantada** de  $u_N(x)$ , o qual é definida como

$$\Delta^n u_N = \sum_{k=0}^n \binom{n}{k} (-1)^k u_{N+n-k},$$

$$\Delta^0 u_N = u_N, \quad \Delta u_N = u_{N+1} - u_N, \quad \Delta^2 u_N = u_{N+2} - 2u_{N+1} + u_N, \quad \Delta^3 u_N = u_{N+3} - 3u_{N+2} + 3u_{N+1} - u_N.$$

A operação inversa de  $\Delta^n$  é

$$u_{N+n} = \sum_{k=0}^n \binom{n}{k} \Delta^k u_N, \quad (n = 0, 1, 2, \dots).$$

#### 2.3.2.2 O MÉTODO $\Delta^2$ DE AITKEN

O método  $\Delta^2$  de Aitken ou *aceleração de Aitken* se deve ao matemático escocês Craig Aitken (1896 – 1967). Este método se baseia na hipótese de que a sequência de somas parciais  $\{S_n\}_0^\infty$  se comporta, a partir de um certo  $n \geq N$ , de forma semelhante à série geométrica (2.9).

Uma forma mais geral para a  $n$ -ésima soma parcial de uma sequência geométrica é

$$S_n = S + bk^n,$$

onde  $S = \lim_{n \rightarrow \infty} S_n$ ,  $b$  é uma constante e  $k < 1$ . Para este tipo de sequência, observa-se as seguintes identidades:

$$\frac{S_{n+1} - S}{S_n - S} = k = \frac{S_{n+2} - S}{S_{n+1} - S}, \quad \Delta S_n = bk^n(k-1), \quad \frac{\Delta S_{n+1}}{\Delta S_n} = k,$$

onde o operador de diferença adiantada foi empregado ( $\Delta S_n = S_{n+1} - S_n$ ). Mas dadas as identidades acima, observa-se que

$$bk^n = \frac{\Delta S_n}{k-1} = \frac{\Delta S_n}{\Delta S_{n+1}/\Delta S_n - 1} = \frac{(\Delta S_n)^2}{\Delta S_{n+1} - \Delta S_n} = \frac{(\Delta S_n)^2}{\Delta^2 S_n}.$$

Observa-se também que

$$\lim_{n \rightarrow \infty} S_n = S = S_n - bk^n = S_n - \frac{(\Delta S_n)^2}{\Delta^2 S_n}.$$

Portanto, A nova sequência  $\{S'_n\}_0^\infty$ , definida por

$$S'_n \doteq S_n - \frac{(\Delta S_n)^2}{\Delta^2 S_n} = S_n - \frac{(S_{n+1} - S_n)^2}{S_{n+2} - 2S_{n+1} + S_n}$$

tem o mesmo limite que a sequência original, mas se aproxima do limite muito mais rapidamente. A demonstração da convergência acelerada de  $S'_n$  em comparação com  $S_n$  é fornecida por *Stoer & Burlisch (2013)* [11].

Este método fornece uma aceleração de convergência para somas parciais que se comportam, para um certo  $n \geq N$ , de forma semelhante a uma sequência geométrica, isto é, se

$$\frac{S_{n+1} - S}{S_n - S} \approx \frac{S_{n+2} - S}{S_{n+1} - S}.$$

Uma discussão mais aprofundada destes métodos de aceleração de convergência e de outros métodos é fornecida por *Dahlquist & Björck (2008)* [12] ou *Burden & Faires (2010)* [13].

## 2.4 RELAÇÕES DE RECORRÊNCIA

Muitas funções especiais não dependem somente do seu argumento, mas também de um índice que pode assumir valores inteiros. Os exemplos mais comuns de funções desse tipo são as funções de Bessel  $J_n(x)$ , e os polinômios de Legendre  $P_n(x)$ , os quais dependem do índice  $n = 0, 1, 2, \dots$ . Neste caso, usualmente essas funções satisfazem *relações de recorrência* (RR) entre valores consecutivos do índice  $n$ . Por exemplo,

$$\begin{aligned} J_{n+1}(x) - \frac{2n}{x} J_n(x) + J_{n-1}(x) &= 0, \\ (n+1) P_n(x) - (2n+1)x P_n(x) + n P_{n-1}(x) &= 0. \end{aligned}$$

As relações acima são denominadas *relações de recorrência de três termos* ou *equações homogêneas de diferenças lineares de segunda ordem*. A forma genérica dessas relações pode ser escrita

$$y_{n+1} + b_n y_n + a_n y_{n-1} = 0, \quad (2.11)$$

onde as quantidades  $y_{n-1}$ ,  $y_n$ ,  $y_{n+1}$ ,  $a_n$  e  $b_n$  podem ser funções de uma variável  $x$  (real ou complexa).

A existência das RR oferece uma aparente vantagem no cálculo de funções que as satisfazem. Por exemplo, se for necessário obter uma sequência de funções de Bessel  $J_0(x)$ ,  $J_1(x)$ ,  $J_2(x)$ , etc, em princípio bastaria calcular as duas primeiras na sequência a partir de suas expansões em série de potências (usando, por exemplo, o algoritmo 2.4) e depois calcular as demais usando a relação de recorrência. Infelizmente, esta estratégia somente irá fornecer o resultado correto para  $|x| < 2n$ . Quando esta condição não é satisfeita, os resultados fornecidos pela relação de recorrência para valores cada vez maiores de  $n$  irão rapidamente divergir dos valores corretos.

A razão para esse comportamento vem do fato de que uma RR de três termos como (2.11) estabelece uma equação de diferenças finitas de segunda ordem que possui duas soluções linearmente independentes (LI), tal como ocorre com uma equação diferencial de segunda ordem. Denotando por  $f_n$  e  $g_n$  duas soluções de (2.11), a condição de independência linear entre as mesmas é verificada se o seu *determinante de Casorati* for não nulo, i. e., se

$$D_n = \begin{vmatrix} f_n & g_n \\ f_{n-1} & g_{n-1} \end{vmatrix} \neq 0.$$

Para muitas relações de recorrência, pode ocorrer que

$$\lim_{n \rightarrow \infty} \frac{f_n}{g_n} = 0, \quad (2.12)$$

em cuja circunstância a solução  $f_n$  é denominada **mínima** ou **recessiva**, enquanto que  $g_n$  é denominada a **solução dominante**.

Suponha então que uma RR com soluções mínima/dominante esteja sendo empregada para obter a solução mínima  $f_n$ . Como a representação numérica é sempre finita, em geral os números que representam os valores iniciais  $f_0$  e  $f_1$  no computador não são exatos, mas sim aproximações  $y_0$  e  $\tilde{f}_1$  dos mesmos, os quais sempre podem ser decompostos em uma combinação linear das soluções mínima/dominante (exatas) como

$$\tilde{f}_i = \mu f_i + \gamma r_0 g_i, \quad (i = 0, 1),$$

onde  $r_0 = f_0/g_0$ ,  $\mu = 1 + \epsilon_1$  e  $\gamma = \epsilon_2$ , onde  $|\epsilon_1|, |\epsilon_2| \simeq x_{eps}$ , sendo  $x_{eps}$  o épsilon de máquina (seção 1.3.2). Após sucessivas aplicações da RR (2.11) para  $n$  crescente, a solução numérica para  $n = k$  fica [15, sec. 4.2.1]

$$y_k = f_k (\mu + \gamma r_0 g_k / f_k),$$

a qual possui uma diferença relativa do valor exato  $f_k$  igual a

$$\epsilon_k = \frac{y_k}{f_k} - 1 = \epsilon_1 + \epsilon_2 \frac{r_0}{r_k}, \text{ sendo } r_k = \frac{f_k}{g_k}.$$

Após um número grande de iterações ( $k \gg 1$ ), o limite (2.12) mostra que  $r_k \rightarrow 0$ ; ou seja,  $\epsilon_k \rightarrow \infty$ . De fato, observa-se que [15, sec. 4.2.1]

$$\frac{y_k/y_{k-1}}{g_k/g_{k-1}} = 1 + \frac{\mu(r_k - r_{k-1})}{\gamma r_0 + \mu r_{k-1}} \xrightarrow{k \gg 1} 1;$$

ou seja, a razão de dois valores numéricos consecutivos ( $y_k/y_{k-1}$ ) se aproxima da razão das soluções dominantes, a qual não é a solução desejada. Isto ocorre *independente dos valores iniciais*  $y_0$  e  $\tilde{f}_1$ . Em outras palavras, a solução numérica converge para a solução dominante. Neste caso, diz-se que a relação de recorrência é *instável* ou *mal condicionada* para a solução mínima.

Felizmente, se a relação de recorrência é instável para a obtenção de  $f_n$  no sentido crescente (ou seja, aumentando  $n$ ), ela será estável no sentido decrescente (diminuindo  $n$ ), porque neste caso  $f_n$  se torna a solução dominante e  $g_n$  a solução mínima. Um método que pode ser empregado nesta situação é mencionado mais adiante. A tabela 2.1 apresenta o sentido de estabilidade das relações de recorrência de algumas funções especiais.

**Tabela 2.1:** Sentidos de estabilidade das relações de recorrência de algumas funções especiais [16].

Estabilidade das relações de recorrência	
Sentido crescente de $n$	Sentido decrescente de $n$
$P_n(x), P_n^m(x)$	$P_n(x), P_n^m(x) (x < 1)$
$Q_n(x), Q_n^m(x) (x < 1)$	$Q_n(x), Q_n^m(x)$
$Y_n(x), K_n(x)$	$J_n(x), I_n(x)$
$J_{-n-1/2}(x), I_{-n-1/2}(x)$	$J_{n+1/2}(x), I_{n+1/2}(x)$
$E_n(x) (n < x)$	$E_n(x) (n > x)$

Portanto, se o uso de uma relação de recorrência estiver sendo considerado para o cálculo da função que a satisfaz, é importante verificar se a relação possui um par de soluções mínima/dominante. Um teorema importante que pode fornecer esta informação é apresentado a seguir.

**Teorema (Perron).** Dada a relação de recorrência de três termos

$$y_{n+1} + b_n y_n + a_n y_{n-1} = 0,$$

se existirem as constantes não nulas  $a, b, \alpha$  e  $\beta$  tais que

$$a_n \sim a n^\alpha, \quad b_n \sim b n^\beta, \quad \text{para } n \rightarrow \infty,$$

então:

1. Se  $\beta > \alpha/2$ , existem duas soluções independentes da relação de recorrência,  $f_n$  e  $g_n$ , tais que

$$\frac{f_n}{f_{n-1}} \sim -\frac{a}{b} n^{\alpha-\beta}, \quad \frac{g_n}{g_{n-1}} \sim -b n^\beta, \quad \text{para } n \rightarrow \infty.$$

2. Se  $\beta = \alpha/2$ , sejam  $t_1$  e  $t_2$  as raízes do polinômio característico  $\phi(t) = t^2 + bt + a$ .

(a) Se  $|t_1| < |t_2|$ , existem soluções independentes  $f_n$  e  $g_n$  tais que

$$\frac{f_n}{f_{n-1}} \sim t_1 n^\beta, \quad \frac{g_n}{g_{n-1}} \sim t_2 n^\beta, \quad \text{para } n \rightarrow \infty.$$

(b) Se  $|t_1| = |t_2|$ , então

$$\lim_{n \rightarrow \infty} \sup \left[ \frac{|y_n|}{n!^\beta} \right]^{1/n} = |t_1|,$$

para qualquer solução da RR.

3. Se  $\beta < \alpha/2$ , então

$$\lim_{n \rightarrow \infty} \sup \left[ \frac{|y_n|}{n!^{\alpha/2}} \right]^{1/n} = \sqrt{|a|},$$

para qualquer solução da RR.

Nos casos 1 e 2(a), a solução  $f_n$  é mínima. Nos demais casos, o teorema é inconclusivo.

A demonstração do teorema de Perron pode vista em Gil *et al.* (2007) [15, sec. 4.3]. O mesmo texto mostra a aplicação do teorema de Perron para várias relações de recorrência, bem como diversas outras propriedades matemáticas e generalizações das mesmas.

### 2.4.1 O MÉTODO DE MILLER PARA A SOLUÇÃO MÍNIMA

Na seção anterior, argumentou-se que a relação de recorrência (2.11) é mal condicionada para a obtenção da solução mínima  $f_n$ . Contudo, a mesma RR torna-se *bem condicionada* para  $f_n$  quando a mesma é calculada para  $n$  no sentido decrescente (chamada *recorrência retrógrada*).

Suponha que é desejado obter a sequência  $\{f_n\}_0^k = f_0, f_1, \dots, f_k$  composta pelas soluções mínimas de (2.11) por recorrência retrógrada. Suponha também que sejam tomados valores iniciais  $f_N$  e  $f_{N-1}$ , com  $N \geq k+2$ . Então, para  $n = N-1, N-2, \dots, 0$ , aplica-se de forma iterativa

$$f_{n-1} = -\frac{1}{a_n} (f_{n+1} + b_n f_n). \quad (2.13)$$

Nesta recorrência, se  $y_n$  é o resultado numérico obtido, então [15, sec. 4.4]

$$\frac{y_n/y_{n-1}}{f_n/f_{n-1}} = 1 + \frac{r_N}{r_{n-1}} \frac{1 - r_{n-1}/r_n}{1 - r_N/r_{k-1}}, \quad (2.14)$$

onde novamente  $r_n = f_n/g_n$ . De acordo com (2.12), é razoável se esperar que  $|r_N| \lesssim |r_n|$  nesta recorrência. Isto significa que o erro numérico introduzido na recorrência retrógrada tende de fato a diminuir com as iterações; a solução  $f_n$  se comporta como a solução dominante na recorrência retrógrada. Neste caso, existem três possibilidades relevantes.

#### 1. Os valores de $f_N$ e $f_{N-1}$ são conhecidos ou podem ser obtidos por algum outro método.

Neste caso, basta tomar  $N = k+2$  e aplicar (2.13) em um laço.

Os outros dois casos se referem a situações onde os valores de  $f_N$  e  $f_{N-1}$  não são conhecidos ou não podem ser obtidos. Nesta situação, é importante observar que (2.14) mostra que são as *razões* dos resultados numéricos ( $y_n/y_{n-1}$ ) que convergem às respectivas razões dos valores exatos, independente dos valores iniciais específicos. Por isto, nesta situação é comum tomar-se  $f_N = 0$  e  $f_{N-1} = 1$  (independente se  $f_n$  é real ou complexo) e escolher  $N \gg k$ , sendo que  $k$  é o último elemento solicitado na sequência  $\{f_n\}_0^k$ . De acordo com (2.14), se  $N$  for suficientemente grande, então para todos os termos da sequência  $\{f_n\}_0^k$  as diferenças relativas entre as razões numéricas  $y_n/y_{n-1}$  e os valores exatos  $f_n/f_{n-1}$  estarão dentro de uma tolerância (exatidão), a qual pode ser fornecida como um parâmetro de entrada que irá determinar para qual valor de  $N$  a acurácia exigida será satisfeita. Desta forma todos os elementos da sequência foram determinados a menos de uma constante de normalização, a qual pode ser determinada de duas maneiras distintas; o que leva às duas outras possibilidades.

## 2. O valor de $f_0 \neq 0$ é conhecido ou pode ser obtido por um outro método.

Neste caso, após a aplicação da recorrência retrógrada, resulta que  $f_0 = \alpha y_0$ ; ou seja,  $\alpha = f_0/y_0$  é a constante de normalização. Assim, a partir da sequência  $\{y_n\}_0^k$ , obtém-se uma nova sequência  $\{w_n\}_0^k$ , com  $w_n = (f_0/y_0)y_n$  ( $n = 0, \dots, k$ ), a qual é a representação numérica de  $\{f_n\}_0^k$  dentro da tolerância exigida.

Resta agora estimar o valor de  $N \gg k$  para que a sequência  $\{w_n\}_0^k$  tenha todos os seus termos acurados dentro de uma tolerância. A partir de (2.14), a seguinte estimativa pode ser empregada na maior parte das aplicações:

$$\epsilon_k = \frac{y_k/y_{k-1}}{f_k/f_{k-1}} - 1 \simeq \frac{r_N}{r_k},$$

sendo que a razão  $r_N/r_k$  pode ser obtida a partir do teorema de Perron. O algoritmo 2.5 implementa este método.

---

**Algoritmo 2.5** Método de Miller quando  $f_0 \neq 0$  é conhecido.

---

**Entrada:**  $\epsilon$  (tolerância),  $k$  e  $f_0$

**Saída:**  $\{w_n\}_0^k$  (aproximação numérica para  $\{f_n\}_0^k$ )

Encontre  $N \gg k$  tal que  $|r_N/r_k| < \epsilon$

Inicialize  $y_n = 0$  e  $y_{n-1} = 1$

**Para**  $n = N - 1, \dots, 1$  **faça**

$$y_{n-1} = -\frac{1}{a_n} (y_{n+1} + b_n y_n)$$

**Fim Laço**

$$\alpha = f_0/y_0$$

**Para**  $n = 0, \dots, k$  **faça**

$$w_n = \alpha y_n$$

**Fim Laço**

---

## 3. A solução mínima satisfaz uma regra de soma.

Em situações onde  $f_0$  não pode ser obtido por um outro método ou pode ser nulo, como ocorre com funções que possuem raízes, a constante de normalização pode ser obtida se a solução mínima da RR satisfaz uma regra de soma do tipo

$$S = \sum_{n=0}^{\infty} \lambda_n f_n,$$

com os coeficientes  $\lambda_n$  conhecidos.

Neste caso, se  $\{y_n\}_0^k$  é a sequência obtida a partir da recorrência retrógrada partindo de  $N \gg k$ , resulta que  $f_n \simeq \alpha y_n$ . Então

$$S \approx \sum_{n=0}^p \lambda_n f_n \approx \alpha S_p, \text{ onde } S_p = \sum_{n=0}^p \lambda_n y_n,$$

para  $p \leq N$  o qual deve ser determinado levando-se em conta a velocidade de convergência da série e a tolerância exigida. Portanto, a constante de normalização é dada por  $\alpha = S/S_p$ , a partir da qual obtém-se a sequência numérica  $\{w_n\}_0^k$ , com  $w_n = \alpha y_n$ .

Para otimizar o tempo computacional, a soma  $S_p$  pode ser computada simultaneamente à recorrência retrógrada. Definindo

$$s_n = \frac{1}{y_{n-1}} \sum_{m=n}^p \lambda_m y_m$$

e assumindo que nenhum  $y_n$  seja nulo, então  $s_n$  satisfaz

$$s_n = H_n (\lambda_n + s_{n+1}), \text{ sendo } H_n = \frac{y_n}{y_{n-1}},$$



a qual é computada de forma retrógrada até  $s_1$ , de onde se obtém  $S_p = (\lambda_0 + s_1) y_0$ . As aproximações numéricas são então  $w_n = (S/S_p) y_n$  ( $n = 0, \dots, k$ ), as quais também podem ser escritas como  $w_n = H_n w_{n-1}$ , partindo de  $w_0 = S y_0 / S_p = S / (\lambda_0 + s_1)$ .

Este método é implementado pelo algoritmo 2.6. Considerações a respeito da convergência do método, estabilidade do cômputo da regra de soma e estimativas para o valor de  $N$  podem ser vistas em [15, sec. 4.6.2].

---

**Algoritmo 2.6** Método de Miller empregando uma regra de soma.

---

**Entrada:**  $\epsilon$  (tolerância),  $k$  e  $S = \sum_{n=0}^{\infty} \lambda_n f_n$

**Saída:**  $\{w_n\}_0^k$  (aproximação numérica para  $\{f_n\}_0^k$ )

Determine valor para  $N$

Inicialize  $H_N = 0$  e  $s_N = 0$

**Para**  $n = N - 1, \dots, 1$  **faça**

$$H_n = -\frac{a_n}{b_n + H_{n+1}}$$

$$s_n = H_n (\lambda_n + s_{n+1})$$

**Fim Laço**

$$w_0 = \frac{S}{\lambda_0 + s_1}$$

**Para**  $n = 1, \dots, k$  **faça**

$$w_n = H_n w_{n-1}$$

**Fim Laço**

---

## 2.5 SÉRIES DE FOURIER-Chebyshev

O valor de uma função também pode ser calculado a partir de uma série de Fourier [10, cap. 19]. De particular importância para métodos numéricos são as séries de Fourier definidas a partir de polinômios ortogonais [10, sec. 12.1] [17], particularmente os polinômios de Chebyshev.<sup>5</sup>

Os polinômios de Chebyshev do primeiro tipo  $\{T_n(x)\}$  são ortogonais no intervalo  $(-1, 1)$  com respeito a função-peso  $w(x) = (1 - x^2)^{-1/2}$ . Suas principais características são:

- Para  $n \geq 0$  e  $-1 \leq x \leq 1$ ,

$$T_n(x) = \cos(n \cos^{-1} x).$$

- Fórmula de Rodrigues:

$$T_n(x) = \frac{(-1)^n}{(2n-1)!!} \sqrt{1-x^2} \left(\frac{d}{dx}\right)^n (1-x^2)^{n-1/2}.$$

- Equação diferencial:

$$(1-x^2)y'' - xy' + n^2y = 0.$$

- Relação de ortogonalidade:

$$\int_{-1}^1 \frac{T_n(x) T_m(x)}{\sqrt{1-x^2}} dx = \begin{cases} 0, & i \neq j \\ \frac{\pi}{2}, & i = j \neq 0 \\ \pi, & i = j = 0. \end{cases}$$

- Relação de recorrência:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \quad (n \geq 1).$$

- O polinômio  $T_n(x)$  ( $n \geq 1$ ) possui  $n$  raízes em  $-1 \leq x \leq 1$ , localizadas em

$$\bar{x}_k = \cos\left(\frac{2k-1}{2n}\pi\right), \quad (k = 1, \dots, n).$$

---

<sup>5</sup>Definidos pelo matemático russo Pafnuty Lvovich Chebyshev (1821–1894).



Os primeiros polinômios de Chebyshev são

$$T_0(x) = 1$$

$$T_3(x) = 4x^3 - 3x$$

$$T_1(x) = x$$

$$T_4(x) = 8x^4 - 8x^2 + 1$$

$$T_2(x) = 2x^2 - 1$$

$$T_5(x) = 16x^5 - 20x^3 + 5x.$$

A figura 2.1 mostra os gráficos dos primeiros polinômios. Pode-se notar que  $-1 \leq T_n(x) \leq 1$ , para  $-1 \leq x \leq 1$ .

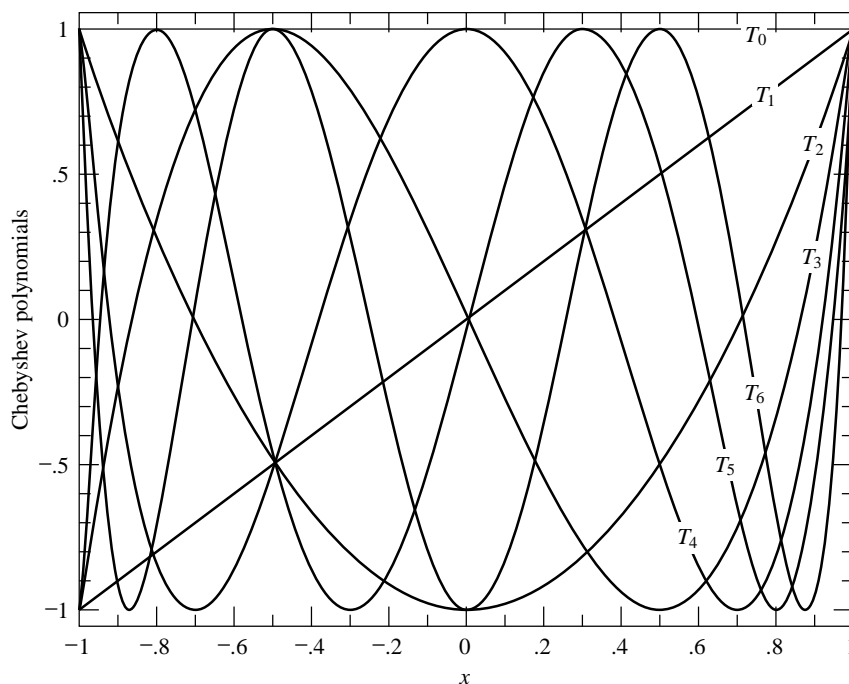


Figura 2.1: Gráficos dos polinômios de Chebyshev  $T_0(x)$  a  $T_6(x)$ .

Seja então uma função  $f(x)$  definida no intervalo  $[-1, 1]$ . A condição de ortogonalidade dos polinômios de Chebyshev permite desenvolver a mesma na *série de Fourier-Chebyshev*

$$f(x) = \sum_{n=0}^{\infty} c_n T_n(x), \quad (-1 \leq x \leq 1),$$

pois, escrevendo

$$f(x) = \frac{1}{2}c_0 + \sum_{m=1}^{\infty} c_m T_m(x),$$

multiplica-se ambos os lados por  $T_n(x)/\sqrt{1-x^2}$  e integra-se, resultando

$$\int_{-1}^1 \frac{f(x) T_n(x)}{\sqrt{1-x^2}} dx = \frac{1}{2}c_0 \int_{-1}^1 \frac{T_0(x) T_n(x)}{\sqrt{1-x^2}} dx + \sum_{m=1}^{\infty} c_m \int_{-1}^1 \frac{T_m(x) T_n(x)}{\sqrt{1-x^2}} dx,$$

$$c_n = \frac{2}{\pi} \int_{-1}^1 \frac{f(x) T_n(x)}{\sqrt{1-x^2}} dx = \frac{2}{\pi} \int_0^{\pi} f(\cos \theta) T_n(\cos \theta) d\theta, \quad (n = 0, 1, \dots).$$

Para certas funções  $f(x)$ , as integrais que devem ser calculadas para determinar os coeficientes  $\{c_n\}$  podem ser realizadas analiticamente. Caso as mesmas não sejam conhecidas, pode-se aproximar as integrais a partir dos métodos discutidos no capítulo 4.

Se o valor absoluto dos coeficientes  $c_n$  cair rápido o suficiente para  $n \rightarrow \infty$ , uma boa aproximação para  $f(x)$  será uma série de Fourier-Chebyshev truncada por

$$S_N(x) = \frac{1}{2}c_0 + \sum_{n=1}^N c_n T_n(x).$$

O teorema a seguir fornece uma medida do erro que resulta do truncamento da série para uma função  $f(x)$  suficientemente bem comportada.

**Teorema.** *Seja uma função  $f(x)$  que possui  $m+1$  derivadas contínuas em  $x \in [-1, 1]$ , para  $m \geq 0$ . Então,*

$$|f(x) - S_N(x)| \sim \mathcal{O}(N^{-m})$$

à medida que  $N \rightarrow \infty$ , para todo  $-1 \leq x \leq 1$ .

A técnica da série de Fourier-Chebyshev pode ser aplicada para a mesma função  $f(x)$ , mas em intervalos arbitrários do domínio. Para tanto, basta realizar a troca de variável

$$t = \frac{1}{2}(b+a) + \frac{1}{2}(b-a)x,$$

a qual irá mapear o intervalo  $[-1, 1]$  para  $[a, b]$ .

## 2.6 FRAÇÕES CONTINUADAS

Sejam  $\{a_n\}_1^\infty$  e  $\{b_n\}_0^\infty$  duas sequências de números reais ou complexos, as quais podem ser funções de uma variável  $x$  (também real ou complexa). A partir destas sequências, é possível construir uma representação para uma função  $f(x)$  dada pela *fração continuada*

$$f(x) = b_0 + \frac{a_1}{b_1 + \frac{a_2}{b_2 + \frac{a_3}{b_3 + \frac{a_4}{b_4 + \dots}}}} \quad (2.15a)$$

$$\equiv b_0 + \frac{a_1}{b_1} \frac{a_2}{b_2} \frac{a_3}{b_3} \frac{a_4}{b_4} \dots, \quad (2.15b)$$

onde a segunda forma é apenas uma representação simplificada da primeira. Por exemplo, a representação da função tangente em frações continuadas é

$$\tan x = \frac{x}{1 - \frac{x^2}{3 - \frac{x^2}{5 - \frac{x^2}{7 - \dots}}}} = \frac{x}{1} \frac{x^2}{3} \frac{x^2}{5} \frac{x^2}{7} \dots$$

Frações continuadas muitas vezes convergem mais rapidamente que expansões em séries de potências e em uma região às vezes maior do domínio da função e distinta da região de convergência da série.

### 2.6.1 FRAÇÕES CONTINUADAS A PARTIR DE RELAÇÕES DE RECORRÊNCIAS OU SÉRIES DE POTÊNCIAS

Frações continuadas para uma dada função  $f(x)$  podem ser obtidas de diferentes maneiras. Aqui serão discutidas frações obtidas a partir de relações de recorrências ou de séries de potências. Se a função satisfaz uma relação de recorrência do tipo (2.11), a mesma pode ser inicialmente escrita como

$$\frac{y_n}{y_{n-1}} = -\frac{a_n}{b_n + y_{n+1}/y_n}.$$

Reescrevendo esta expressão para valores consecutivamente maiores de  $n$  e inserindo os resultados no último denominador, resulta

$$\frac{y_n}{y_{n-1}} = -\frac{a_n}{b_n - \frac{a_{n+1}}{b_{n+1} - \frac{a_{n+2}}{b_{n+2} - \frac{a_{n+3}}{b_{n+3} - \dots}}}} = -\frac{a_n}{b_n} \frac{a_{n+1}}{b_{n+1}} \frac{a_{n+2}}{b_{n+2}} \frac{a_{n+3}}{b_{n+3}} \dots$$

Segundo o teorema de Pincherle [15, sec. 4.4], esta relação de recorrência converge se e somente se a mesma possui uma solução mínima. Métodos para o uso desta fração continuada para o cálculo da relação de recorrência são fornecidos por Gil et al. (2007) [15].

Para séries de potências, Cuyt et al. (2008) [18] fornecem diversos métodos. O primeiro método a ser apresentado é o **método de Euler**. Dada a série geral

$$f = \sum_{k=0}^{\infty} c_k, \quad (2.16)$$

é possível estabelecer-se uma sequência de aproximações parciais em termos de uma fração continuada a partir da sequência de somas parciais  $\{f_n\}_0^{\infty}$  dadas por  $f_n = \sum_{k=0}^n c_k$ .

De acordo com Cuyt et al. (2008), a fração continuada associada à série de  $f$  é

$$f = c_0 + \frac{c_1}{1 + \frac{-c_2/c_1}{1 + \frac{-c_3/c_2}{\dots \frac{-c_k/c_{k-1}}{1 + c_k/c_{k-1}} \dots}}}, \quad (2.17a)$$

ou, na forma extensa,

$$f = c_0 + \frac{c_1}{1 + \frac{-c_2/c_1}{1 + c_2/c_1 + \frac{-c_3/c_2}{1 + c_3/c_2 + \frac{-c_4/c_3}{1 + c_4/c_3 + \dots}}}}. \quad (2.17b)$$

A fração continuada acima pode ser obtida a partir da sequência  $\{f_n\}_0^{\infty}$  usando-se indução matemática. Iniciando-se por  $f_2$ ,

$$f_2 = c_0 + c_1 + c_2 = c_0 + \frac{c_1}{\frac{1}{1 + c_2/c_1}} = c_0 + \frac{c_1}{1 + \frac{-c_2/c_1}{1 + c_2/c_1}}.$$

Manipulando agora  $f_3$ ,

$$\begin{aligned} f_3 &= c_0 + c_1 + c_2 + c_3 = c_0 + \frac{c_1}{\frac{1}{1 + c_2/c_1 + c_3/c_1}} = c_0 + \frac{c_1}{1 + \frac{-c_2/c_1 - c_3/c_1}{1 + c_2/c_1 + c_3/c_1}} = \dots \\ &= c_0 + \frac{c_1}{1 + \frac{-c_2/c_1}{1 + c_2/c_1 + \frac{-c_3/c_2}{1 + c_3/c_2}}}. \end{aligned}$$

Finalmente, manipulando  $f_4$  da mesma maneira, obtém-se

$$\begin{aligned} f_4 &= c_0 + c_1 + c_2 + c_3 + c_4 = \dots \\ &= c_0 + \frac{c_1}{1 + \frac{-(c_2/c_1)}{1 + (c_2/c_1) + \frac{-(c_3/c_2)}{1 + (c_3/c_2) + \frac{-(c_4/c_3)}{1 + (c_4/c_3)}}}}. \end{aligned}$$

Por indução, resulta a fórmula geral (2.17).

Assumindo-se agora uma série de potências com uma variável independente,

$$f(z) = \sum_{k=0}^{\infty} a_k z^k.$$

Obviamente, por comparação com (2.16), observa-se que  $c_k = a_k z^k$  e, portanto, a fração continuada de  $f(z)$  é dada, a partir de (2.17), como

$$f(z) = a_0 + \frac{a_1 z}{1 + \frac{-(a_2/a_1) z}{1 + (a_2/a_1) z + \frac{-(a_3/a_2) z}{1 + (a_3/a_2) z + \dots \frac{-(a_k/a_{k-1}) z}{1 + (a_k/a_{k-1}) z} \dots}}}. \quad (2.18)$$

De acordo com *Cuyt et al. (2008)*, os resultados dados por (2.17) e (2.18) são de utilidade limitada, uma vez que as frações continuadas obtidas têm as mesmas propriedades de convergência e divergência apresentadas pelas séries.

### 2.6.2 MÉTODO DE VISKOVATOV

Como um método mais poderoso, *Cuyt et al. (2008)* apresentam então a sequência de Viskovatov, a qual é obtida a partir de (2.16), realizando-se diversas manipulações. Dados os coeficientes

$$\begin{aligned} d_{00} &= 1, \quad d_{0j} = 0, \quad (j \geq 1) \\ d_{1j} &= c_{j+1}, \quad (j \geq 0) \\ d_{2j} &= -d_{1,j+1} = -c_{j+2}, \quad (j \geq 0) \\ d_{kj} &= d_{k-1,0}d_{k-2,j+1} - d_{k-2,0}d_{k-1,j+1}, \quad (k \geq 3, j \geq 0), \end{aligned}$$

a fração continuada de Viskovatov para a função  $f$  é

$$f = c_0 + \frac{d_{10}}{d_{00} + d_{10}} \frac{d_{20}}{d_{10} + d_{20}} \frac{d_{30}}{d_{20} + d_{30}} \frac{d_{40}}{d_{30} + d_{40}} \frac{d_{50}}{d_{40} + d_{50}} \cdots \quad (2.19a)$$

Os textos sobre frações continuadas muitas vezes escrevem este resultado de forma compacta como [18, Eq. 1.1.1d]

$$f = c_0 + \mathbf{K}_{m=1}^{\infty} \left( \frac{d_{m,0}}{d_{m-1,0}} \right). \quad (2.19b)$$

A série obtida da sequência de Viskovatov apresenta propriedades de convergência distintas da série original. *Cuyt et al. (2008)* argumentam que em geral esta série converge em uma região mais ampla que o raio de convergência da série de potências original.

### 2.6.3 TRANSFORMAÇÕES DE EQUIVALÊNCIA E CONTRAÇÕES

Em *Gil et al. (2007)* [15, Chap. 6] são discutidos métodos de manipulação de uma fração continuada na forma (2.15). Uma *transformação de equivalência* sobre  $f(z)$  consiste em escolher um conjunto  $\{c_j\}$  ( $c_j \neq 0$ ) adequado, de tal forma que

$$f(z) = b_0 + \frac{c_1 a_1}{c_1 b_1 +} \frac{c_1 c_2 a_2}{c_2 b_2 +} \frac{c_2 c_3 a_3}{c_3 b_3 +} \frac{c_3 c_4 a_4}{c_4 b_4 +} \frac{c_4 c_5 a_5}{c_5 b_5 +} \frac{c_5 c_6 a_6}{c_6 b_6 +} \cdots.$$

A *contração par* da fração continuada equivalente acima é obtida escolhendo-se  $c_k = b_k^{-1}$ ,

$$f(z) = b_0 + \frac{a_1/b_1}{1+} \frac{a_2/(b_1 b_2)}{1+} \frac{a_3/(b_2 b_3)}{1+} \frac{a_4/(b_3 b_4)}{1+} \frac{a_5/(b_4 b_5)}{1+} \frac{a_6/(b_5 b_6)}{1+} \frac{a_7/(b_6 b_7)}{1+} \cdots.$$

Mas este último resultado também pode ser escrito como

$$f(z) = b_0 + \frac{a_1/b_1}{1+a_2/(b_1 b_2)-} \frac{[a_2/(b_1 b_2)][a_3/(b_2 b_3)]}{1+[a_3/(b_2 b_3)]+[a_4/(b_3 b_4)]-} \frac{[a_4/(b_3 b_4)][a_5/(b_4 b_5)]}{1+[a_5/(b_4 b_5)]+[a_6/(b_5 b_6)]-} \frac{[a_6/(b_5 b_6)][a_7/(b_6 b_7)]}{1+[a_7/(b_6 b_7)]+[a_8/(b_7 b_8)]-} \cdots.$$

Esta é a contração par de  $f(z)$ , a qual pode ser escrita de forma compacta como

$$f(z) = b_0 + \frac{a_1/b_1}{1 + \frac{a_2}{b_1 b_2} - \mathbf{K}_{m=1}^{\infty} \left( \frac{[a_{2m}/(b_{2m-1} b_{2m})][a_{2m+1}/(b_{2m} b_{2m+1})]}{1+[a_{2m+1}/(b_{2m} b_{2m+1})]+[a_{2m+2}/(b_{2m+1} b_{2m+2})]} \right)}. \quad (2.20)$$

Uma contração par de uma fração continuada usualmente converge duas vezes mais rápido do que a fração original, desde que os novos coeficientes não sejam demasiado complicados.

Dada agora a série

$$f(z) = \sum_{k=0}^{\infty} \alpha_k z^k,$$

a fração continuada de Euler da mesma é identificada a partir de (2.18) fazendo-se a substituição  $a_k \rightarrow \alpha_k$  e a partir de (2.15) por

$$a_1 = \alpha_1 z \quad a_j = -(\alpha_j / \alpha_{j-1}) z \quad (j \geq 2)$$

$$b_0 = \alpha_0$$

$$b_1 = 1$$

$$b_j = 1 - a_j \quad (j \geq 2).$$

Então, a contração par da série de potência fica dada por (2.20), resultando

$$f(z) = \alpha_0 + \frac{\alpha_1 z}{1 - \frac{\alpha_2}{\alpha_1} z / \left(1 + \frac{\alpha_2}{\alpha_1} z\right) - \frac{\left[\frac{\alpha_2}{\alpha_1} z / \left(1 + \frac{\alpha_2}{\alpha_1} z\right)\right] \left\{\frac{\alpha_3}{\alpha_2} z / \left[\left(1 + \frac{\alpha_2}{\alpha_1} z\right) \left(1 + \frac{\alpha_3}{\alpha_2} z\right)\right]\right\}}{1 - \frac{\alpha_3}{\alpha_2} z / \left[\left(1 + \frac{\alpha_2}{\alpha_1} z\right) \left(1 + \frac{\alpha_3}{\alpha_2} z\right)\right] - \frac{\alpha_4}{\alpha_3} z / \left[\left(1 + \frac{\alpha_3}{\alpha_2} z\right) \left(1 + \frac{\alpha_4}{\alpha_3} z\right)\right]} - \dots} \quad (2.21)$$

$$\mathbf{K}_{j=2}^{\infty} \left( \frac{\left\{ \frac{\alpha_{2j}}{\alpha_{2j-1}} z / \left[\left(1 + \frac{\alpha_{2j-1}}{\alpha_{2j-2}} z\right) \left(1 + \frac{\alpha_{2j}}{\alpha_{2j-1}} z\right)\right]\right\} \left\{ \frac{\alpha_{2j+1}}{\alpha_{2j}} z / \left[\left(1 + \frac{\alpha_{2j}}{\alpha_{2j-1}} z\right) \left(1 + \frac{\alpha_{2j+1}}{\alpha_{2j}} z\right)\right]\right\}}{1 - \frac{\alpha_{2j+1}}{\alpha_{2j}} z / \left[\left(1 + \frac{\alpha_{2j}}{\alpha_{2j-1}} z\right) \left(1 + \frac{\alpha_{2j+1}}{\alpha_{2j}} z\right)\right] - \frac{\alpha_{2j+2}}{\alpha_{2j+1}} z / \left[\left(1 + \frac{\alpha_{2j+1}}{\alpha_{2j}} z\right) \left(1 + \frac{\alpha_{2j+2}}{\alpha_{2j+1}} z\right)\right]} \right).$$

## 2.6.4 COMPUTAÇÃO DE FRAÇÕES CONTINUADAS

Retornando à fração continuada escrita na forma (2.15), para realizar *de facto* a sua computação, é necessário transformá-la numa função racional, i. e., se  $f_j$  é o  $j$ -ésimo **aproximante** de  $f(z)$ , então

$$f_j = \frac{A_j}{B_j},$$

onde  $A_j$  e  $B_j$  são funções dos  $a$ 's e  $b$ 's:

$$A_{-1} = 1 \quad B_{-1} = 0 \quad A_0 = b_0 \quad B_0 = 1$$

$$A_j = b_j A_{j-1} + a_j A_{j-2} \quad B_j = b_j B_{j-1} + a_j B_{j-2} \quad (j \geq 1).$$

De fato, dada (2.15), os primeiros aproximantes são

$$f_0 = b_0 \quad f_1 = b_0 + \frac{a_1}{b_1} = \frac{b_0 b_1 + a_1}{b_1} \quad f_2 = b_0 + \frac{a_1}{b_1 + \frac{a_2}{b_2}} = \frac{b_0 (b_1 b_2 + a_2) + a_1 b_2}{b_1 b_2 + a_2}$$

$$f_3 = b_0 + \frac{a_1}{b_1 + \frac{a_2}{b_2 + \frac{a_3}{b_3}}} = \frac{b_0 [b_1 (b_2 b_3 + a_3) + a_2 b_3] + a_1 (b_2 b_3 + a_3)}{b_1 (b_2 b_3 + a_3) + a_2 b_3} = \frac{[(b_0 b_1 + a_1) b_2 + a_2 b_0] b_3 + (b_0 b_1 + a_1) a_3}{(b_1 b_2 + a_2) b_3 + a_3 b_1}.$$

Já usando a forma racional,

$$f_1 = \frac{A_1}{B_1} = \frac{b_1 b_0 + a_1}{b_1} \quad f_2 = \frac{A_2}{B_2} = \frac{b_2 (b_1 b_0 + a_1) + a_2 b_0}{b_2 b_1 + a_2}$$

$$f_3 = \frac{A_3}{B_3} = \frac{b_3 [b_2 (b_1 b_0 + a_1) + a_2 b_0] + a_3 (b_1 b_0 + a_1)}{b_3 (b_2 b_1 + a_2) + a_3 b_1}.$$

Ou seja, de fato as relações de recorrência para  $A_j$  e  $B_j$  estão corretas.

Há alguns métodos usuais para realizar essa computação. Como o melhor método genérico, Press et al. (2007) [7] e Gil et al. (2007) [15] sugerem o uso do *algoritmo modificado de Lenz*, por sua robustez contra overflows e estabilidade contra erros de arredondamento. De acordo com o mesmo, o aproximante  $f_j$  é dado por

$$f_j = f_{j-1} C_j D_j,$$

onde

$$C_j = \frac{A_j}{A_{j-1}} \quad D_j = \frac{B_{j-1}}{B_j}.$$

Desta forma a possibilidade de overflows no numerador e/ou denominador é contornada.

Além disso, existem relações de recorrência para  $C_j$  e  $D_j$ . Calculando primeiro

$$C_j C_{j-1} - b_j C_{j-1} = \frac{b_j A_{j-1} + a_j A_{j-2} - b_j (b_{j-1} A_{j-2} + a_{j-1} A_{j-3})}{b_{j-2} A_{j-3} + a_{j-2} A_{j-4}} = a_j,$$

onde foi empregada a relação de recorrência para  $A_j$ . E,

$$D_j (b_j + a_j D_{j-1}) = 1.$$

Ou seja,

$$C_j = b_j + \frac{a_j}{C_{j-1}} \qquad D_j = \frac{1}{b_j + a_j D_{j-1}},$$

o que diminui consideravelmente o acúmulo dos erros de arredondamento.

Existe ainda o perigo de que os denominadores das últimas relações de recorrência sejam nulos ou próximos a zero. Para tanto, o algoritmo 2.7 implementa o método modificado de Lenz, levando em conta essa possibilidade.

---

**Algoritmo 2.7** Algoritmo de Lenz modificado para o cômputo de uma fração continuada.

---

**Entrada:**  $\{a_n\}_1^\infty$ ,  $\{b_n\}_0^\infty$ ,  $\epsilon$  (tolerância) e tiny

**Saída:**  $f_j$  ( $j = 0, 1, \dots$ )

$f_0 = b_0$

**Se**  $b_0 = 0$  **então**

$f_0 = \text{tiny}$

**Fim Se**

$c_0 = f_0$ ;  $d_0 = 0$

**Para**  $j = 1, 2, \dots$  **faça**

$d_j = b_j + a_j d_{j-1}$

**Se**  $|d_j| < \text{tiny}$  **então**

$d_j = \text{tiny}$

**Fim Se**

$c_j = b_j + a_j / c_{j-1}$

**Se**  $|c_j| < \text{tiny}$  **então**

$c_j = \text{tiny}$

**Fim Se**

$d_j = 1/d_j$

$h_j = c_j d_j$

$f_j = f_{j-1} h_j$

**Se**  $|h_j - 1| < \epsilon$  **então**

**exit**

**Fim Se**

**Fim Laço**

*Obs:* É sugerida a seguinte relação entre  $\epsilon$  e tiny:  $10^{-30} \leq \text{tiny} \ll \epsilon |b_j|$ .

---

# DERIVAÇÃO NUMÉRICA

## 3.1 INTRODUÇÃO

Derivação e integração numéricas são alguns dos métodos que mais se utiliza em física computacional. Com frequência é necessário calcular  $f'(x)$  ou  $\int f(x)dx$  para uma determinada função  $f(x)$  para a qual pode existir ou não uma expressão analítica. O objetivo deste capítulo é introduzir alguns dos métodos mais empregados na derivação numérica, mantendo-se como objetivo a precisão numérica dos métodos.

Ao contrário da integração numérica, que será apresentada no próximo capítulo, a derivação possui alta suscetibilidade a erros de truncamento e, principalmente, de arredondamento. Isto se deve à própria natureza da derivação, definida por

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}. \quad (3.1)$$

Como sempre há erros associados com o cálculo de  $f(x)$ , estes serão ampliados à medida que se executa numericamente o processo de limite  $h \rightarrow 0$ . Como a representação finita de números reais em computadores impõe um limite inferior no valor de  $f(x+h) - f(x)$ , erros de arredondamento rapidamente começam a se acumular durante o processo de limite, levando o cálculo, finalmente, a um processo de cancelamento catastrófico, como se pode observar no exemplo da figura 1.5. Portanto, derivação numérica somente deve ser realizada quando não houver outro método para a solução do problema em estudo.

Nas próximas seções serão apresentados métodos que possibilitarão o cálculo numérico de derivadas primeiras e segundas de funções que possuem ou não uma expressão analítica conhecida. Alguns destes métodos serão posteriormente empregados no cálculos de equações diferenciais ordinárias ou parciais.

## 3.2 FÓRMULAS CLÁSSICAS DE DIFERENÇA FINITA

Como já foi adiantado no exemplo 1.12, a maneira mais óbvia (e mais ingênua) de se calcular numericamente uma derivada consiste em tomar literalmente a definição (3.1) e substituí-la por uma fórmula de diferença finita, resultando na fórmula de diferença adiantada (*forward difference*).

### 3.2.1 FÓRMULA DE DIFERENÇA ADIANTADA (*forward difference*)

Esta fórmula consiste em tomar (3.1) e ignorar o processo de limite:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}. \quad (3.2)$$

Aplicado desta forma, este procedimento está fadado a ter uma exatidão limitada ou ao quase certo fracasso, o que não o impede de ser largamente utilizado no cálculo numérico de equações diferenciais parciais. Portanto a fórmula (3.2) somente deve ser empregada quando considerações de tempos computacionais forem importantes.

Há duas fontes importantes de erros em (3.2): erro de truncamento e erro de arredondamento. O erro de truncamento pode ser estimado a partir do desenvolvimento de  $f(x+h)$  em uma série

de McLaurin em torno de  $x$ :

$$f(x \pm h) = f(x) \pm f'(x)h + \frac{1}{2}f''(x)h^2 \pm \frac{1}{6}f'''(x)h^3 + \frac{1}{24}f^{iv}(x)h^4 + \dots, \quad (3.3)$$

resultando

$$\frac{f(x+h) - f(x)}{h} = f'(x) + \frac{1}{2}f''(x)h + \mathcal{O}(h^2).$$

Ou seja, o erro de truncamento é da ordem

$$\epsilon_t \sim |f''h|.$$

Para diminuir este erro (para valores finitos de  $f''$ ), poder-se-ia tentar diminuir o valor de  $h$ , o que acabaria levando ao aumento do valor do erro de arredondamento a partir de um valor crítico de  $h$ , como se pode observar na figura 1.5.

O erro de arredondamento possui duas origens. A primeira seria o erro no processo de arredondamento realizado na representação finita dos pontos  $x$  e  $x+h$ , conforme discutido na seção 1.5.3. Mesmo se as representações de  $x$ ,  $h$ ,  $x+h$ ,  $f(x)$  e  $f(x+h)$  fosse consideradas “exatas,” ainda resta o erro de arredondamento no processo de cálculo da diferença finita  $[f(x+h) - f(x)]/h$ . Este erro é da ordem

$$\epsilon_a \sim \epsilon_f \left| \frac{f(x)}{h} \right|,$$

onde  $\epsilon_f$  é a precisão fracional no cálculo de  $f(x)$ . Para uma função bem comportada,  $\epsilon_f \approx \epsilon_m$ , sendo  $\epsilon_m$  a precisão da máquina (1.5). A suposição que  $\epsilon_a \propto h^{-1}$  pode ser inferido a partir da figura 1.5. Assim, o erro total no cálculo de (3.2) pode ser estimado como

$$\epsilon_{\text{total}}(h) = \epsilon_t(h) + \epsilon_a(h) \sim |f''h| + \epsilon_f \left| \frac{f(x)}{h} \right|. \quad (3.4)$$

O valor de  $h$  que minimiza este erro pode ser estimado pelo cálculo do mínimo de  $\epsilon_{\text{total}}$ , resultando

$$h_{\min} \simeq \sqrt{\epsilon_f \left| \frac{f}{f''} \right|} \equiv \sqrt{\epsilon_f} x_c,$$

onde  $x_c = \sqrt{f/f''}$  é denominado **escala de curvatura** de  $f(x)$  ou de **escala característica** sobre a qual  $f(x)$  varia. Na ausência de uma melhor informação, usualmente usa-se  $x_c \approx x$ .

Portanto, o erro relativo na melhor estimativa da derivada, conforme dada por (3.2) é:

$$\frac{\epsilon_{\text{total}}}{|f'|} \approx \sqrt{\epsilon_f} \left( \frac{|ff''|}{f'^2} \right)^{1/2} \approx \sqrt{\epsilon_f},$$

onde se supôs que  $f$ ,  $f'$  e  $f''$  sejam todos da mesma ordem de grandeza. Pode-se ver que a fórmula de diferença adiantada fornece como melhor estimativa somente a raiz quadrada do epsilon de máquina. Para precisão simples,  $\epsilon_m \sim 10^{-7}$ , portanto,  $\sqrt{\epsilon_m} \sim 10^{-4}$ , ou seja, 4 dígitos significativos corretos. Já para precisão dupla,  $\epsilon_m \sim 10^{-16}$ , resultando  $\sqrt{\epsilon_m} \sim 10^{-8}$ , ou 8 dígitos significativos corretos. Esta estimativa está coerente com os resultados mostrados na figura 1.5.

### 3.2.2 FÓRMULA DE DIFERENÇA ATRASADA (*backward difference*)

Quando as condições de contorno do sistema em estudo assim o exigirem, pode ser necessário usar uma fórmula de diferença atrasada para estimar a derivada de uma função. Esta fórmula consiste simplesmente em tomar:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}. \quad (3.5)$$

Neste caso,

$$\frac{f(x) - f(x-h)}{h} = f'(x) - \frac{1}{2}f''(x)h + \mathcal{O}(h^2),$$

e, portanto, o erro total é igual a (3.4).



### 3.2.3 FÓRMULA DE DIFERENÇA CENTRADA (*centered difference*)

Uma melhor estimativa para a derivação é obtida a partir da *fórmula de diferença centrada*

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}. \quad (3.6)$$

Utilizando (3.3), observa-se que

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + \frac{1}{6}f'''(x)h^2 + \mathcal{O}(h^4). \quad (3.7)$$

Ou seja, o erro de truncamento deste método é da ordem

$$\epsilon_t \sim |f'''(x)|h^2,$$

resultando na seguinte estimativa de valor de  $h$  que minimiza o erro total:

$$h_{min} \sim \sqrt[3]{\epsilon_f \left| \frac{f}{f'''} \right|} \sim \sqrt[3]{\epsilon_f} x_c$$

e com um erro relativo igual a

$$\frac{\epsilon_{total}}{|f'|} \sim \sqrt[3]{f^2 \left| \frac{f'''}{f'^3} \right|} (\epsilon_f)^{2/3} \sim (\epsilon_f)^{2/3}.$$

Assim, para precisão simples,  $\epsilon_m^{2/3} \sim 10^{-5}$ , ou seja, 5 dígitos significativos corretos, e para precisão dupla,  $\epsilon_m^{2/3} \sim 10^{-11}$ , ou 11 dígitos significativos corretos, aumentando sensivelmente a exatidão na estimativa da derivada. A vantagem desta fórmula comparada com a fórmula (3.2) é claramente visível na figura 1.5.

### 3.2.4 FÓRMULA DE 5 PONTOS

Uma aproximação ainda melhor pode ser obtida partindo-se de (3.3) e (3.7), de onde vem

$$\begin{aligned} f(x+h) - f(x-h) &= 2f'(x)h + \frac{1}{3}f'''(x)h^3 + \frac{1}{60}f^{(5)}(x)h^5 + \mathcal{O}(h^7) \\ f(x+2h) - f(x-2h) &= 4f'(x)h + \frac{8}{3}f'''(x)h^3 + \frac{8}{15}f^{(5)}(x)h^5 + \mathcal{O}(h^7). \end{aligned}$$

Combinando-se adequadamente ambas as fórmulas, obtém-se

$$f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h) = 12f'(x)h - \frac{2}{5}f^{(5)}(x)h^5 + \mathcal{O}(h^7),$$

ou seja,

$$\frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h} = f'(x) - \frac{1}{30}f^{(5)}(x)h^4 + \mathcal{O}(h^6).$$

Portanto,

$$f'(x) \approx \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h} + \frac{1}{30}f^{(5)}(x)h^4, \quad (3.8)$$

a qual é conhecida como *fórmula de derivada de 5 pontos*.

O erro de truncamento agora é da ordem

$$\epsilon_t \sim |f^{(5)}| h^4,$$

o que implica na seguinte estimativa de  $h_{min}$  que minimiza o erro total:

$$h_{min} \sim \sqrt[5]{\frac{\epsilon_f |f|}{|f^{(5)}|}},$$

com um erro relativo da ordem

$$\frac{\epsilon_{total}}{|f'|} \sim \sqrt[5]{\frac{|f^{(5)}| |f|^4}{|f'|^5}} \epsilon_f^{4/5}. \quad (3.9)$$

Para precisão simples,  $\epsilon_m^{4/5} \sim 10^{-6}$ , ou seja, 6 dígitos significativos corretos, enquanto que em precisão dupla,  $\epsilon_m^{4/5} \sim 10^{-13}$ , ou seja, 13 dígitos significativos corretos.

Combinações ainda mais elaboradas levam a aproximações ainda mais acuradas para a derivada. Entretanto, o número de cálculos da função  $f(x)$  em diferentes pontos aumenta com a ordem do método empregado. Por conseguinte, não é usualmente vantajoso usar um método ainda mais acurado que a fórmula da derivada de 5 pontos (3.8).

### 3.3 FÓRMULAS DE DIFERENÇAS FINITAS PARA A DERIVADA SEGUNDA

As fórmulas introduzidas na seção 3.2 podem ser estendidas para o cálculo da derivada de segunda ordem de  $f(x)$ , com diferentes ordens de convergência.

#### 3.3.1 FÓRMULA DE TRÊS PONTOS

Realizando-se a seguinte combinação linear a partir de (3.3):

$$f(x+h) - 2f(x) + f(x-h) = f''(x)h^2 + \frac{1}{12}f^{iv}(x)h^4 + \mathcal{O}(h^6) + \dots,$$

resulta,

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{1}{12}f^{iv}(x)h^2, \quad (3.10)$$

a qual já parte de uma exatidão equivalente ao método de diferença centrada (3.6) para a derivada de primeira ordem.

#### 3.3.2 FÓRMULA DE CINCO PONTOS

Uma aproximação ainda melhor é obtida a partir da combinação:

$$-f(x-2h) + 16f(x-h) - 30f(x) + 16f(x+h) - f(x+2h) \approx 12f''(x)h^2 - \frac{2}{15}f^{vi}(x)h^6,$$

resultando na fórmula de cinco pontos para a derivada segunda:

$$f''(x) \approx -\frac{f(x-2h) - 16f(x-h) + 30f(x) - 16f(x+h) + f(x+2h)}{12h^2} + \frac{1}{90}f^{vi}(x)h^4. \quad (3.11)$$

Expressões para derivadas de ordens ainda mais altas também pode ser obtidas a partir de combinações lineares de  $f(x)$  calculada em diferentes pontos. As mesmas fórmulas também podem ser empregadas para a implementação do cálculo numérico de derivadas parciais e de operadores vetoriais. Estas generalizações são apresentadas por Abramowitz & Stegun (1970) [21, cap. 25] e Temme (2010) [22, sec. 3.4].

### 3.4 FÓRMULAS PARA O CÁLCULO DE DERIVADAS EM PONTOS FORA DA REDE

As fórmulas apresentadas nas seções 3.2 e 3.3 são úteis quando a função  $f(x)$  possui uma expressão analítica conhecida. Neste caso, para um valor arbitrário de  $h$ , sempre é possível calcular-se  $f(x \pm h)$ . Entretanto, em muitas aplicações práticas, a forma analítica de  $f(x)$  não é conhecida, sendo esta definida somente em pontos regularmente espaçados em uma rede, como em uma tabela.

O problema consiste agora em determinar o valor da derivada de  $f(x)$  em pontos tanto sobre quanto fora da rede. Agora, o parâmetro  $h$  irá representar o espaçamento da rede, ou seja,  $f(x)$  é conhecida nos pontos  $x = x_i$ ,  $i = 1, 2, \dots, n$ , tais que  $x_1 < x_2 < x_3 < \dots < x_n$  e  $x_{i+1} - x_i = h$ . Nestes pontos,  $f(x_i) \equiv f_i$ . Deseja-se então calcular a derivada em um ponto  $x = x_i + ph$ , sendo que a quantidade  $p$  não necessariamente é inteira. Ou seja, quer-se obter  $f'_p = f'(x_i + ph)$ .

Nas seções abaixo estão listadas algumas fórmulas oferecidas por Abramowitz & Stegun (1970) [21] para derivações em pontos fora da rede.

### 3.4.1 DERIVADA DE TRÊS PONTOS

$$f'(x_i + ph) \approx \frac{1}{h} \left[ \left(p - \frac{1}{2}\right) f_{-1} - 2pf_i + \left(p + \frac{1}{2}\right) f_1 \right] + 0,13f'''(\xi)h^2, \quad (x_1 < \xi < x_n)$$

onde  $f_{-1} = f(x_i - h)$  e  $f_1 = f(x_i + h)$ . Assim, se  $p = 0$ , esta fórmula se reduz à derivada centrada (3.6).

### 3.4.2 DERIVADA DE QUATRO PONTOS

$$f'(x_i + ph) \approx -\frac{1}{2h} \left[ \frac{3p^2 - 6p + 2}{3} f_{-1} - (3p^2 - 4p - 1) f_i + (3p^2 - 2p - 2) f_1 - \frac{3p^2 - 1}{3} f_2 \right] + \begin{cases} 0,096f^{iv}(\xi)h^3, & (0 < p < 1) \\ 0,168f^{iv}(\xi)h^3, & (-1 < p < 0) \end{cases}$$

onde  $f_2 = f(x_i + 2h)$ .

### 3.4.3 DERIVADA DE CINCO PONTOS

$$f'(x_i + ph) \approx \frac{1}{2h} \left[ \frac{2p^3 - 3p^2 - p + 1}{6} f_{-2} - \frac{4p^3 - 3p^2 - 8p + 4}{3} f_{-1} + (2p^2 - 5) pf_i - \frac{4p^3 + 3p^2 - 8p - 4}{3} f_1 + \frac{2p^3 + 3p^2 - p - 1}{6} f_2 \right] + 0,06f^v(\xi)h^4,$$

onde agora  $f_{-2} = f(x_i - 2h)$  e, para  $p = 0$ , a fórmula acima reduz-se à derivada de cinco pontos (3.8).

## 3.5 EXTRAPOLAÇÃO DE RICHARDSON E ESTIMATIVA DE ERRO

Nesta seção será introduzida uma idéia útil em diversos ramos da análise numérica, conhecida como *Extrapolação de Richardson*. Será mostrado como se pode obter uma aproximação muito boa do erro absoluto de qualquer das fórmulas apresentadas para derivação numérica e como esta estimativa pode ser usada para incrementar substancialmente a acurácia do resultado.

Este tratamento depende da possibilidade de variação livre do espaçamento de grade  $h$  ou de qualquer outro parâmetro de controle no método e, portanto, não é útil quando a função for conhecida somente em uma grade fixa, como no caso de valores experimentais. Para estes casos, o programador está restrito às fórmulas e às estimativas de erro apresentadas na seção 3.4. Contudo, quando a forma funcional de  $f(x)$  for conhecida, o método da extrapolação de Richardson possibilita um resultado extremamente acurado, juntamente com uma excelente estimativa de erro.

Considera-se, então, um algoritmo numérico destinado a executar uma determinada operação. Sendo  $f_{\text{exato}}$  o valor exato do resultado desta operação e  $fl(f, h)$  a aproximação a  $f_{\text{exato}}$  obtida com o uso do algoritmo numérico, o qual é de ordem  $n$ , regulado pelo parâmetro de controle  $h$ . Pode-se então escrever

$$f_{\text{exato}} = fl(f, h) + \mathcal{A}[f]h^n + \mathcal{B}[f]h^{n+m} + \dots,$$

onde  $\mathcal{A}$  e  $\mathcal{B}$  são funcionais aplicados a  $f$ . Sendo o algoritmo de ordem  $n$ , o erro predominante é dado pelo termo  $\mathcal{A}[f]h^n$ , ao passo que o termo posterior na correção é dado pelo termo  $\mathcal{B}[f]h^{n+m}$ , para  $m \geq 1$ . Aplicando-se então o algoritmo para valores do parâmetro  $h_1 = h$  e  $h_2 = h/R$ , sendo  $R > 1$ , resulta

$$f_{\text{exato}} = fl(f, h) + \mathcal{A}[f]h^n + \mathcal{B}[f]h^{n+m} + \dots \quad (3.12a)$$

$$f_{\text{exato}} = fl\left(f, \frac{h}{R}\right) + \mathcal{A}[f] \left(\frac{h}{R}\right)^n + \mathcal{B}[f] \left(\frac{h}{R}\right)^{n+m} + \dots \quad (3.12b)$$

Os erros absolutos nas aproximações  $fl(f, h)$  e  $fl(f, h/R)$  são dados, em mais baixa ordem, respectivamente por

$$\begin{aligned} EA(h) &= f_{\text{exato}} - fl(f, h) \simeq \mathcal{A}[f]h^n \\ EA\left(\frac{h}{R}\right) &= f_{\text{exato}} - fl\left(f, \frac{h}{R}\right) \simeq \mathcal{A}[f] \left(\frac{h}{R}\right)^n. \end{aligned}$$

Subtraindo (3.12b) de (3.12a), obtém-se, até a ordem  $n + m$ :

$$\begin{aligned} 0 &= fl(f, h) - fl\left(f, \frac{h}{R}\right) + \mathcal{A}[f]h^n - \mathcal{A}[f] \left(\frac{h}{R}\right)^n + \mathcal{O}(h^{n+m}), \\ 0 &= fl(f, h) - fl\left(f, \frac{h}{R}\right) + \frac{(R^n - 1)}{R^n} \mathcal{A}[f]h^n + \mathcal{O}(h^{n+m}). \end{aligned}$$

Resolvendo a equação acima para  $h_1$  e  $h_2$ , obtém-se então as estimativas de erro, exatas até a ordem  $n + m$ :

$$\mathcal{A}[f]h^n = EA(h) \simeq \left[ fl\left(f, \frac{h}{R}\right) - fl(f, h) \right] \frac{R^n}{R^n - 1} \quad (3.13a)$$

$$\mathcal{A}[f] \left(\frac{h}{R}\right)^n = EA\left(\frac{h}{R}\right) \simeq \left[ fl\left(f, \frac{h}{R}\right) - fl(f, h) \right] \frac{1}{R^n - 1}. \quad (3.13b)$$

Expressão (3.13a) consiste em uma estimativa do valor do erro do algoritmo empregando o parâmetro  $h_1$ , ao passo que (3.13b) é a estimativa do erro do mesmo algoritmo usando o parâmetro  $h_2$ .

Uma vez obtida a estimativa de erro para a melhor aproximação (usando  $h_2$ ), pode-se adicionar esta estimativa ao valor aproximado  $fl(f, h_2)$  para se realizar um refinamento no resultado; isto é, uma vez que

$$f_{\text{exato}} = fl\left(f, \frac{h}{R}\right) + \mathcal{A}[f] \left(\frac{h}{R}\right)^n + \mathcal{B}[f] \left(\frac{h}{R}\right)^{n+m} + \mathcal{C}[f] \left(\frac{h}{R}\right)^{n+m+\ell} + \dots,$$

substituindo (3.13b) obtém-se o valor *extrapolado*

$$f_{\text{exato}} = fl\left(f, \frac{h}{R}\right) + \left[ fl\left(f, \frac{h}{R}\right) - fl(f, h) \right] \frac{1}{R^n - 1} + \mathcal{B}[f] \left(\frac{h}{R}\right)^{n+m} + \mathcal{C}[f] \left(\frac{h}{R}\right)^{n+m+\ell} + \dots,$$

ou

$$f_{\text{exato}} = \frac{1}{R^n - 1} \left[ R^n fl\left(f, \frac{h}{R}\right) - fl(f, h) \right] + \mathcal{B}[f] \left(\frac{h}{R}\right)^{n+m} + \mathcal{C}[f] \left(\frac{h}{R}\right)^{n+m+\ell} + \dots, \quad (3.14)$$

o qual possui um erro da ordem  $h^{n+m}$ .

Observa-se que com o emprego da técnica da extrapolação de Richardson obteve-se um estimativa bastante acurada do erro resultante da aplicação do método numérico para um valor de parâmetro  $h_2 = h/R$ , sem haver a necessidade de se desenvolver a forma operatorial de  $\mathcal{A}[f]$ . Além disso, o valor extrapolado (3.14) resulta ser de mais alta ordem ( $n + m$ ) que a aproximação original  $fl(f, h_2)$ .

Parece, assim, que um único processo de cálculo fornece dois refinamentos: uma estimativa de erro ao mesmo e um valor extrapolado de ordem mais alta. Contudo, *Press et al. (1992)* [6] constantemente enfatizam que *ordem mais alta não significa necessariamente maior acurácia*. O valor extrapolado pode ser de ordem mais alta, mas não existe nenhuma estimativa do erro associado ao mesmo; o erro calculado está associado ao valor não extrapolado  $fl(f, h_2)$ . Para que se obtenha uma estimativa do erro de (3.14) é necessário aplicar-se uma vez mais o processo de extrapolação.

Escrevendo-se as aproximações não extrapoladas realizadas inicialmente como

$$fl_0(h) \equiv fl(f, h), \quad fl_0\left(\frac{h}{R}\right) \equiv fl\left(f, \frac{h}{R}\right),$$

pode-se escrever a aproximação extrapolada uma vez (3.14) como

$$fl_1(h) \equiv fl\left(f, \frac{h}{R}\right) + \left[fl\left(f, \frac{h}{R}\right) - fl(f, h)\right] \frac{1}{R^n - 1} = \frac{R^n fl(f, h/R) - fl(f, h)}{R^n - 1} \\ \equiv \frac{R^n fl_0(h/R) - fl_0(h)}{R^n - 1},$$

podendo-se então escrever (3.14) como

$$f_{\text{exato}} = fl_1(h) + \mathcal{B}[f] \left(\frac{h}{R}\right)^{n+m} + \mathcal{C}[f] \left(\frac{h}{R}\right)^{n+m+\ell} + \dots$$

Escrevendo agora esta última expressão para  $h/R$ ,

$$f_{\text{exato}} = fl_1\left(\frac{h}{R}\right) + \mathcal{B}[f] \left(\frac{h}{R^2}\right)^{n+m} + \mathcal{C}[f] \left(\frac{h}{R^2}\right)^{n+m+\ell} + \dots,$$

e subtraindo ambas as expressões, obtém-se as seguintes expressões para os erros:

$$EA_1(h) \simeq \mathcal{B}[f] \left(\frac{h}{R}\right)^{n+m} = \left[fl_1\left(\frac{h}{R}\right) - fl_1(h)\right] \frac{R^{n+m}}{R^{n+m} - 1} \quad (3.15a)$$

$$EA_1\left(\frac{h}{R}\right) \simeq \mathcal{B}[f] \left(\frac{h}{R^2}\right)^{n+m} = \left[fl_1\left(\frac{h}{R}\right) - fl_1(h)\right] \frac{1}{R^{n+m} - 1}, \quad (3.15b)$$

sendo que  $EA_1(h)$  é a estimativa de erro para  $fl_1(h)$ , a qual não havia sido obtida na iteração anterior, e  $EA_1(h/R)$  é a estimativa de erro para  $fl_1(h/R)$ . Agora, o novo valor extrapolado passa a ser  $fl_2(h)$ , dado por

$$fl_2(h) \equiv fl_1\left(\frac{h}{R}\right) + \left[fl_1\left(\frac{h}{R}\right) - fl_1(h)\right] \frac{1}{R^{n+m} - 1} = \frac{R^{n+m} fl_1(h/R) - fl_1(h)}{R^{n+m} - 1}, \quad (3.16a)$$

e

$$f_{\text{exato}} = fl_2(h) + \mathcal{C}[f] \left(\frac{h}{R^2}\right)^{n+m+\ell} + \dots \quad (3.16b)$$

Esta nova aproximação possui um erro de ordem  $h^{n+m+\ell}$ .

Pode-se induzir assim que próximo termo extrapolado será

$$fl_3(h) = \frac{R^{n+m+\ell} fl_2(h/R) - fl_2(h)}{R^{n+m+\ell} - 1}, \quad (3.17a)$$

e

$$f_{\text{exato}} = fl_3(h) + \mathcal{D}[f] \left(\frac{h}{R^3}\right)^{n+m+\ell+p} + \dots, \quad (3.17b)$$

com uma estimativa de erros igual a

$$EA_3(h) \simeq \mathcal{C}[f] \left(\frac{h}{R^2}\right)^{n+m+\ell} = \left[fl_3\left(\frac{h}{R}\right) - fl_3(h)\right] \frac{R^{n+m+\ell}}{R^{n+m+\ell} - 1} \quad (3.18a)$$

$$EA_3\left(\frac{h}{R}\right) \simeq \mathcal{C}[f] \left(\frac{h}{R^3}\right)^{n+m} = \left[fl_3\left(\frac{h}{R}\right) - fl_3(h)\right] \frac{1}{R^{n+m+\ell} - 1}. \quad (3.18b)$$

E assim sucessivamente para ordens mais altas. Deve-se ressaltar por fim que o algoritmo  $fl(f, h)$  somente foi utilizado no cálculo dos termos não extrapolados  $fl_0(h)$ . Os termos restantes são obtidos apenas com o uso da regra de extrapolação. Contudo, para obter-se  $fl_k(h)$  para  $k > 0$ , é necessário conhecer-se os termos extrapolados anteriores, o que implica, ao final das contas, que é necessário aplicar-se o algoritmo  $fl_0$   $k$  vezes, para valores de incrementos consecutivamente menores:  $fl_0(h)$ ,  $fl_0(h/R)$ ,  $fl_0(h/R^2)$ ,  $\dots$ ,  $fl_0(h/R^k)$ .

**EXEMPLO 3.1.** Como exemplo de uso da extrapolação de Richardson para o cálculo de derivação numérica, emprega-se a expressão para derivação centrada, juntamente com a sua estimativa de erro (3.7),

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{1}{6}f'''(x)h^2 + \mathcal{O}(h^4),$$

a qual mostra que o método é de ordem  $n = 2$ , enquanto que o próximo termo é de ordem  $n + m = 4$ . Também identifica-se  $\mathcal{A}[f] = -f'''(x)/6$ . Tomando-se o valor  $R = 2$  e empregando-se a fórmula acima para  $h_1 = h$  e  $h_2 = h/2$ , obtêm-se as seguintes expressões:

$$\begin{aligned} f'(x) &= \frac{f(x+h) - f(x-h)}{2h} - \frac{1}{6}f'''(x)h^2 + \mathcal{O}(h^4), \\ f'(x) &= \frac{f(x+h/2) - f(x-h/2)}{h} - \frac{1}{6}f'''(x)\frac{h^2}{4} + \mathcal{O}\left(\frac{h^4}{16}\right). \end{aligned}$$

Chamando então

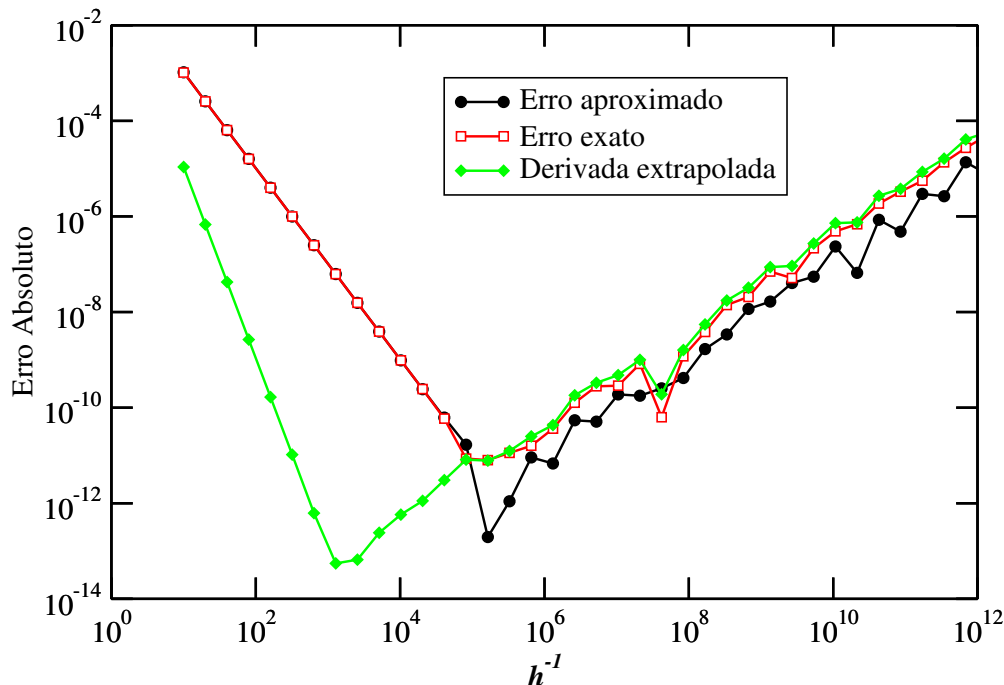
$$fl(f', h_1) = \frac{f(x+h) - f(x-h)}{2h} \text{ e } fl(f', h_2) = \frac{f(x+h/2) - f(x-h/2)}{h},$$

a estimativa de erro obtida para  $fl(f', h_2)$  é dada por (3.13b),

$$EA(h_2) \simeq \frac{1}{2^2 - 1} [fl(f', h_2) - fl(f', h_1)] = \frac{1}{3} [fl(f', h_2) - fl(f', h_1)], \quad (3.19a)$$

ao passo que o valor extrapolado para o cálculo da derivada é dado por (3.14),

$$f'(x) = fl(f', h_2) + \frac{1}{3} [fl(f', h_2) - fl(f', h_1)] + \mathcal{O}(h^4). \quad (3.19b)$$



**Figura 3.1:** Gráfico log-log da estimativa de erro absoluto no cálculo da derivada centrada da função  $f(x) = \sin x^2$ , em  $x = 0,5$ , juntamente com o valor exato do erro e o valor extrapolado da derivada.

O resultado obtido com o emprego das expressões (3.19a,b) pode ser visto na figura 3.1. Nesta, a exemplo do que se fez na figura 1.5, calculou-se numericamente a derivada numérica da função  $f(x) = \sin x^2$  no ponto  $x = 0,5$  a partir da fórmula de diferença centrada. Neste caso, porém, pode-se calcular também a estimativa de erro (3.19a) e o valor extrapolado (3.19b). Observa-se que a estimativa de erro é tão boa que se torna indistinguível do valor exato do erro

durante todo o intervalo de valores de  $h$  para os quais o erro de truncamento é mais importante que o erro de arredondamento. Além disso, observa-se também que o valor extrapolado refina o resultado em mais de 2 ordens de grandeza. O programa em Fortran que gerou os dados apresentados na figura 3.1 está no Programa 3.1.

**Listing 3.1:** Programa em Fortran 95 que calculou o erro e o valor extrapolado apresentados na Figura 3.1.

```

program derivadas_extrapol
implicit none
integer, parameter :: dp= 8
integer :: i
real(kind= dp) :: h= 0.1_dp      ! h inicializado a 1/50.
real(kind= dp), parameter :: x= 0.5_dp ! Valor de x fixo.
real(kind= dp) :: df1, df2, df, erro_est, fl
!
fl= 2.0_dp*x*cos(x*x) ! Valor correto da derivada em x.
open(unit=10, file='derivs_ext.dat')
do i= 1, 45
  df1= fp3(x,h) ! Derivada diferença centrada
  df2= fp3(x,0.5_dp*h)
  erro_est= (df2 - df1)/3.0_dp
  df= df2 + erro_est
  write(10, '(4(e10.4,1x)) ') 1.0_dp/h, abs(erro_est), &
                                abs(df2 - fl), abs(df - fl)
  h= 0.5_dp*h ! h é dividido por 2.
end do
CONTAINS
function f(x)
real(kind= dp) :: f
real(kind= dp), intent(in) :: x
f= sin(x*x)
return
end function f
!
function fp3(x,h)
real(kind= dp) :: fp3
real(kind= dp), intent(in) :: x,h
fp3= 0.5_dp*(f(x+h) - f(x-h))/h
return
end function fp3
end program derivadas_extrapol

```

Caso se queira empregar este método para uma função definida somente em pontos de rede, os incrementos  $h_1$  e  $h_2$  devem necessariamente ser pontos desta rede. Neste caso, colocando  $h_1 = 2h$  e  $h_2 = h$  em (3.1b), resulta

$$\begin{aligned}
 f'(x) &= fl(f', h) + \frac{1}{3} [fl(f', h) - fl(f', 2h)] + \mathcal{O}(h^4) \\
 &= \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h} + \mathcal{O}(h^4),
 \end{aligned}$$

a qual é justamente a fórmula de 5 pontos (3.8). Portanto, o uso do valor extrapolado não é útil para pontos fixos de rede, uma vez que o valor de  $h$  não pode ser variado.

A subrotina DFDX\_RICH, listada no Programa 3.2 implementa o cálculo da derivada numérica usando o Método de Richardson. A rotina tem como parâmetros de entrada o nome da função  $f(x)$  analítica a ser derivada, a qual deve ser fornecida por meio de uma função externa, o ponto onde calcular a derivada, o tamanho inicial do parâmetro  $h$  e o limite superior solicitado para o erro relativo do resultado. Como saídas, a rotina fornece o valor numérico de  $f'(x)$  e uma estimativa do valor do erro relativo. Como o resultado da Método de Richardson é equivalente ao resultado do método de 5 pontos (seção 3.2.4), o menor erro relativo possível é estimado igual ao fornecido pela fórmula (3.9). Portanto, se o valor solicitado para o erro relativo máximo for

menor que este valor, a rotina automaticamente irá interromper o processamento, pois os erros de arredondamento irão impedir a obtenção de um resultado com a precisão solicitada.



**Listing 3.2:** Subrotina que calcula numericamente a derivada pelo Método da Extrapolação de Richardson.

```

! Calcula a derivada numérica de uma função analítica pelo
! Método da Extrapolação de Richardson.
! Argumentos:
! f: Função externa a ser derivada (entrada).
! x: Ponto onde calcular a derivada (entrada).
! h_ini: Valor inicial para o intervalo de diferença finita (entrada).
! errest: Valor máximo solicitado para o erro relativo (entrada).
! dfdx: Valor da derivada numérica (saida).
! err_sai: Valor estimado do erro relativo da derivada (saida).
!
! Obs.: Caso parametro errest seja menor que a estimativa de
! erro relativo mínimo para o método, a rotina interrompe
! automaticamente o processamento.
subroutine dfdx_rich(f, x, h_ini, errest, dfdx, err_sai)
implicit none
integer, parameter :: dp= selected_real_kind(10,200)
real(kind= dp), intent(in) :: x, h_ini, errest
real(kind= dp), intent(out) :: dfdx, err_sai
interface
    function f(x)
        integer, parameter :: dp= selected_real_kind(10,200)
        real(kind= dp), intent(in) :: x
        real(kind= dp) :: f
    end function f
end interface
! Variaveis locais.
integer :: i
real(kind= dp) :: h, df1, df2, err_abs
real(kind= dp), parameter :: errrel_min= 3.0e-13_dp
!
if(errest <= errrel_min)then
    print'("O erro relativo solicitado é muito pequeno." ,/, &
        "Erros de arredondamento irão impedir que a rotina",/, &
        " atinja a precisão solicitada.")'
    STOP
end if
h= h_ini
df1= df_cent(x, h)
do
    df2= df_cent(x, 0.5_dp*h)
    err_abs= (df2 - df1)/3.0_dp
    dfdx= df2 + err_abs
    err_sai= abs(err_abs/dfdx)
    if(err_sai <= errest)exit
    df1= df2
    h= 0.5_dp*h
end do
return
CONTAINS
    function df_cent(x, h)
        real(kind= dp) :: df_cent
        real(kind= dp), intent(in) :: x, h
        df_cent= 0.5_dp*(f(x+h) - f(x-h))/h
        return
    end function df_cent
end subroutine dfdx_rich

```



# INTEGRAÇÃO NUMÉRICA

## 4.1 INTRODUÇÃO

Integração numérica, também denominada *quadratura*, possui uma história que se estende desde antes da invenção do cálculo. O fato de integrais de funções elementares não poderem, em geral, ser calculadas analiticamente, ao passo que suas derivadas são facilmente obtidas, serviu de razão para enfatizar esta área da análise numérica já nos séculos XVIII e XIX.

Em contraste com a dificuldade de se calcular analiticamente uma integral, o cálculo numérico pode ser realizado de forma relativamente simples, exatamente ao contrário do que acontece com a derivação. A definição de uma integral de Riemann consiste no limite da soma da área delimitada por regiões retangulares à medida que a largura  $h$  dos retângulos vai a zero e o seu número total vai a infinito:

$$\int_a^b f(x)dx = \lim_{h \rightarrow 0} \left[ h \sum_{i=1}^{(b-a)/h} f(x_i) \right].$$

Uma maneira tradicional de medir numericamente a área sob  $f(x)$  consiste em traçar o seu gráfico sobre um papel milimetrado e contar o número de quadrados sob a curva. Por esta razão a integração numérica também foi denominada inicialmente de *quadratura numérica*.

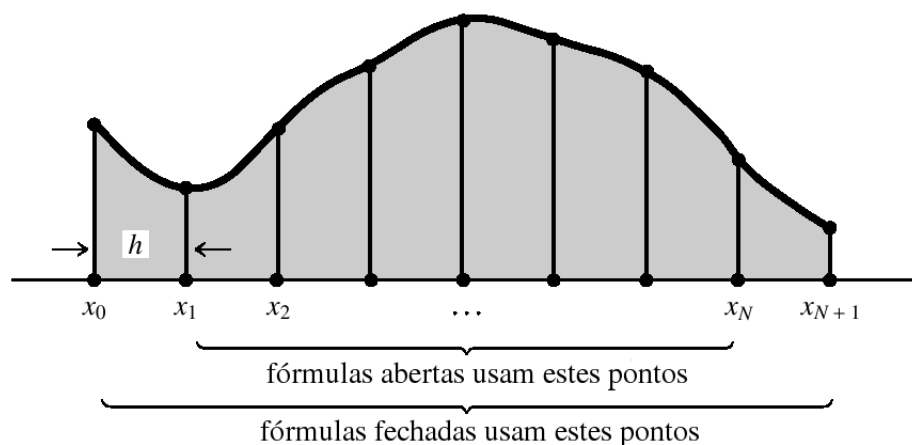
A integral da função  $f(x)$  é aproximada numericamente de uma forma equivalente à soma dos quadrados ou retângulos. A maior parte das fórmulas abordadas neste capítulo podem ser colocadas na forma:

$$\int_a^b f(x)dx = \sum_{i=1}^N f(x_i)w_i + \epsilon_N. \quad (4.1)$$

Aqui,  $f(x)$  é calculada em  $N$  pontos situados no intervalo  $[a, b]$  (para fórmulas *fechadas*, isto é, que envolvem os limites) ou no intervalo  $(a, b)$  (para fórmulas *abertas*, que não envolvem os limites). Os valores das funções calculados em cada ponto do intervalo,  $f_i = f(x_i)$  são então somados com o intermédio de um peso  $w_i$ . A quantidade  $\epsilon_N$  consiste na estimativa do erro de truncamento do método empregado. Embora os métodos, em geral, somente forneçam o resultado exato para  $N \rightarrow \infty$ , alguns deles fornecem o resultado exato para certas classes especiais de funções (como polinômios, por exemplo) para  $N$  finito.

Os diferentes algoritmos de integração utilizam distintos conjuntos de pontos  $\{x_i\}$  e de pesos  $\{w_i\}$ . Geralmente, a precisão aumenta com  $N$ , mas erros de arredondamento eventualmente acabam por limitar a precisão final. Uma vez que o “melhor” método depende do comportamento específico de  $f(x)$ , não existe um método que possa ser universalmente considerado o melhor. De fato, alguns dos esquemas automáticos de integração numérica, que podem ser encontrados em bibliotecas tais como a IMSL, irão testar diferentes métodos até encontrar aquele que forneça o melhor resultado.

Nos esquemas mais simples de integração, o integrando é aproximado por uns poucos termos no desenvolvimento em série de McLaurin de  $f(x)$ , sendo estes os termos a ser integrados. Exceto no caso do integrando apresentar um comportamento não usual em algum intervalo de valores de  $x$ , termos sucessivos, obtidos com o aumento de  $N$ , irão fornecer precisão cada vez maior, até que os erros de arredondamento se tornem suficientemente importantes. Nestes esquemas, denominados de *Newton-Cotes*, o intervalo total é dividido em subintervalos iguais, conforme ilustrado na figura 4.1, com o integrando calculado em pontos igualmente espaçados  $x_i$ . Estes algoritmos incluem a regra trapezoidal (primeira ordem) e a regra de Simpson (segunda ordem).



**Figura 4.1:** Fórmulas de quadratura com abscissas igualmente espaçadas calculam a integral de uma função entre  $x_0$  e  $x_{N+1}$ . Fórmulas fechadas calculam o valor da função nos pontos extremos do intervalo, enquanto que fórmulas abertas não usam estes pontos.

Esquemas mais acurados de integração são possíveis se os pontos não necessariamente forem regularmente espaçados. Métodos de *quadratura Gaussiana* possuem a habilidade de integrar exatamente (exceto pelo erro de arredondamento) o produto de uma função por um polinômio de grau  $(2N - 1)$ , utilizando somente  $N$  valores de  $f(x)$ . Em geral, resultados obtidos pela quadratura Gaussiana são superiores aos obtidos pelos métodos de Newton-Cotes, desde que não haja singularidades no integrando ou em sua derivada.

## 4.2 FÓRMULAS DE NEWTON-COTES

As fórmulas de Newton-Cotes (1711) para a integração numérica são caracterizadas por pontos de integração igualmente espaçados no intervalo de integração  $(a, b)$ . Seja  $N$  um número inteiro que determina o número total de pontos onde a função  $f(x)$  deve ser calculada e a quantidade  $h$  é o espaçamento dos pontos, conforme pode ser observado na figura 4.1. Os pontos do conjunto  $\{x_i\}$  que serão realmente empregados no cálculo da integração numérica dependem se a quadratura será aberta ou fechada, conforme também está representado na figura 4.1.

### 4.2.1 FÓRMULAS FECHADAS DE NEWTON-COTES

As fórmulas fechadas de Newton-Cotes que serão aqui introduzidas são ilustradas pela figura 4.2. Serão utilizados  $N + 1$  pontos igualmente espaçados, identificados pelo índice  $i$  ( $i = 0, 1, \dots, N$ ), com o espaçamento entre os pontos dado por

$$h = \frac{x_N - x_0}{N} = \frac{b - a}{N}.$$

Os pontos de integração das fórmulas de Newton-Cotes serão definidos por:

$$x_i = a + ih, \quad (i = 0, \dots, N),$$

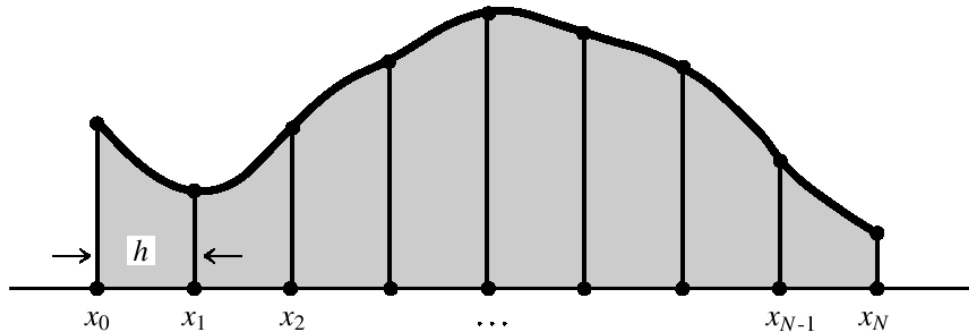
enquanto que os valores da função nos pontos  $x_i$  serão representados por

$$f_i \equiv f(x_i).$$

### POLINÔMIO DE LAGRANGE.

A função  $f(x)$  será agora aproximada por um polinômio interpolador, isto é, no lugar de  $f(x)$  considera-se um polinômio de grau  $N$ ,  $p_N(x)$ , o qual possui os mesmos valores da função nos pontos  $x_i$ , isto é,

$$p_N(x_i) = f_i, \quad (i = 0, \dots, N).$$



**Figura 4.2:** Nas fórmulas fechadas, são utilizados  $N + 1$  pontos, que variam de  $x_0 = a$  a  $x_N = b$ .

Sem demonstração,<sup>1</sup> estes polinômios são dados por

$$\begin{aligned} p_N(x) &= f(x_0)l_0^N(x) + f(x_1)l_1^N(x) + \cdots + f(x_N)p_N^N(x) \\ &= \sum_{n=0}^N f_n l_n^N(x), \end{aligned} \quad (4.2a)$$

onde  $l_n^N(x)$  são os polinômios de Lagrange, definidos por

$$l_n^N(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_{n-1})(x - x_{n+1}) \cdots (x - x_N)}{(x_n - x_0)(x_n - x_1) \cdots (x_n - x_{n-1})(x_n - x_{n+1}) \cdots (x_n - x_N)} = \prod_{\substack{i=0 \\ i \neq n}}^N \frac{(x - x_i)}{(x_n - x_i)}. \quad (4.2b)$$

A verificação de que  $p_N(x)$  é o polinômio interpolador segue diretamente da substituição dos pontos  $x_i$ :

$$\forall n \in \{0, 1, \dots, N\}, l_n^N(x_n) = 1 \text{ e } l_n^N(x_k) = 0, \text{ para } k \neq n \implies p_N(x_n) = f_n.$$

O erro associado à aproximação fornecida pelos polinômios de Lagrange,  $\epsilon_N(x) \equiv f(x) - p_N(x)$ , é dado por:

$$\epsilon_N(x) = \frac{f^{(N+1)}(\alpha)}{(N+1)!} [(x - x_0)(x - x_1) \cdots (x - x_N)], \quad (4.2c)$$

onde

$$\alpha \in [a, b].$$

Desta forma, pode-se escrever,

$$f(x) = p_N(x) + \epsilon_N(x), \quad (4.2d)$$

para  $x \in [a, b]$ .

## USO DOS POLINÔMIOS DE LAGRANGE NA INTEGRAÇÃO NUMÉRICA

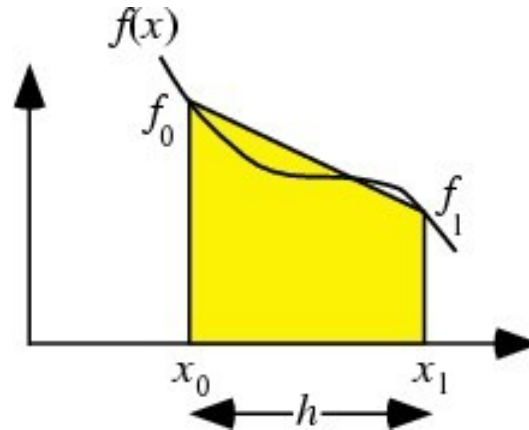
Fazendo uso então do polinômio de Lagrange de grau  $N$  para aproximar  $f(x)$  pelos  $N + 1$  pontos ilustrados na figura 4.2, obtém-se:

$$\begin{aligned} \int_a^b f(x) dx &= \int_a^b [p_N(x) + \epsilon_N(x)] dx \\ &= \sum_{i=0}^N f_i \int_a^b l_i^N(x) dx + \frac{f^{(N+1)}(\alpha)}{(N+1)!} \int_a^b [(x - x_0)(x - x_1) \cdots (x - x_N)] dx. \end{aligned} \quad (4.3a)$$

Pode-se então escrever a integral na forma (4.1), onde os pesos da fórmula de integração são obtidos por:

$$w_i = \int_a^b l_i^N(x) dx = \int_a^b \frac{(x - x_0)(x - x_1) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_N)}{(x_i - x_0)(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_N)} dx, \quad (4.3b)$$

<sup>1</sup>A qual será apresentada em um capítulo posterior desta Apostila.



**Figura 4.3:** Integração numérica pela regra trapezoidal. A área sob a curva  $f(x)$  entre  $x_0$  e  $x_1$  é aproximada pela área do trapézio amarelo.

ao passo que os erros de truncamento serão dados por

$$\epsilon_N = \frac{f^{(N+1)}(\alpha)}{(N+1)!} \int_a^b [(x-x_0)(x-x_1)\dots(x-x_N)] dx. \quad (4.3c)$$

Deve-se enfatizar aqui que a fórmula (4.3a) é exata para um polinômio de grau  $\leq N$ .

Desta forma, pode-se obter fórmulas do tipo Newton-Cotes para polinômios de qualquer grau. Historicamente, as primeiras fórmulas foram estabelecidas para polinômios de graus baixos. Algumas destas fórmulas serão apresentadas a seguir.

Fórmulas fechadas são aquelas que utilizam os pontos extremos do intervalo  $[a, b]$ , isto é,

$$x_0 = a \text{ e } x_N = b.$$

Estas fórmulas são adequadas quando o integrando for bem comportado nos limites de integração; não apresentando singularidades, por exemplo.

#### 4.2.1.1 REGRA TRAPEZOIDAL ( $N = 1$ )

A regra trapezoidal, ou fórmula do trapézios, corresponde à interpolação de  $f(x)$  a ser integrada por um polinômio de grau 1. Como a interpolação linear necessita de somente 2 pontos, estes serão os extremos do intervalo de integração, isto é,  $x_0 = a$  e  $x_1 = b$ , com  $N = 1$  e  $h = b - a$ .

As fórmulas (4.3a-c) nos permitem encontrar os pesos:

$$w_0 = \int_{x_0}^{x_1} \frac{(x-x_1)}{(x_0-x_1)} dx = -\frac{1}{2} \frac{(x-b)^2}{(b-a)} \Big|_a^b = \frac{1}{2}h,$$

$$w_1 = \int_{x_0}^{x_1} \frac{(x-x_0)}{(x_1-x_0)} dx = \frac{1}{2} \frac{(x-a)^2}{(b-a)} \Big|_a^b = \frac{1}{2}h$$

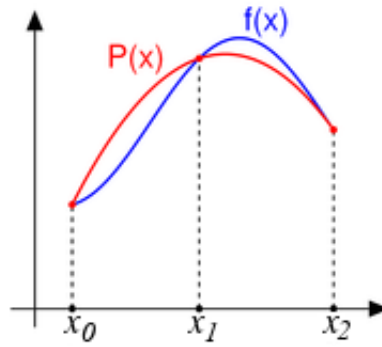
e o erro

$$\epsilon_1 \equiv \epsilon_T = \frac{f''(\alpha)}{2} \int_{x_0}^{x_1} (x-x_0)(x-x_1) dx = -\frac{1}{12} f''(\alpha) h^3.$$

Portanto, a regra trapezoidal para integração no intervalo  $(x_0, x_1)$  fica

$$\int_a^b f(x) dx = \frac{h}{2} (f_0 + f_1) - \frac{1}{12} f''(\alpha) h^3. \quad (4.4)$$

O nome *fórmula dos trapézios* vem do fato de se aproximar a função  $f(x)$  por um trapézio de lados  $f_0$  e  $f_1$  e de base  $h = b - a$ , conforme está representado na figura 4.3. Pode-se observar que este método, bastante simples, já fornece um erro de ordem  $\epsilon_T \sim h^3$ .



**Figura 4.4:** Integração numérica pela regra de Simpson. A área sob a curva  $f(x)$  entre  $x_0$  e  $x_2$  é aproximada pela área sob a parábola  $P(x)$ .

#### 4.2.1.2 REGRA DE SIMPSON ( $N = 2$ )

Esta é uma das regras de integração mais conhecidas e utilizadas. A função  $f(x)$  é aproximada por um polinômio de grau 2 que coincide com esta em três pontos:  $x_0$ ,  $x_1$  e  $x_2$ . Portanto, é necessário conhecer 3 valores de  $f(x)$ , igualmente espaçados, para aplicar esta regra.

Tomando  $N = 2$ ,  $x_0 = a$ ,  $x_1 = (a + b)/2$ ,  $x_2 = b$  e  $h = (b - a)/2$  em (4.3a-c), obtém-se para os pesos

$$\begin{aligned} w_0 &= \int_{x_0}^{x_2} \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} dx = \frac{1}{3}h \\ w_1 &= \int_{x_0}^{x_2} \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} dx = \frac{4}{3}h \\ w_2 &= \int_{x_0}^{x_2} \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} dx = \frac{1}{3}h, \end{aligned}$$

ao passo que o erro de truncamento fica dado por

$$\epsilon_2 = \frac{f'''(\alpha)}{3!} \int_{x_0}^{x_2} (x - x_0)(x - x_1)(x - x_2) dx.$$

Contudo, surpreendentemente obtém-se que  $\epsilon_2 = 0$ ! Isto pode ser facilmente visto realizando-se a mudança de variáveis  $t = x - x_1$  na integração acima, resultando

$$\int_{x_0}^{x_2} (x - x_0)(x - x_1)(x - x_2) dx = \int_{-h}^h (t + h)t(t - h) dt = \int_{-h}^h (t^2 - h^2)t dt = 0.$$

Isto não significa que o erro na regra de Simpson é sempre nulo. Neste caso é necessário tomar o próximo termo no cálculo do erro  $\epsilon_N$  de um polinômio interpolador de grau  $N$ , dado inicialmente por (4.2c), porém agora acrescentando o próximo termo no desenvolvimento em série de McLaurin de  $f(x)$  em torno de  $x = x_1$ :

$$\frac{f^{(iv)}(\alpha)}{4!} (x - x_1)^4.$$

Desta forma, o erro fica:

$$\epsilon_S = \frac{f^{(iv)}(\alpha)}{4!} \int_{x_0}^{x_2} (x - x_0)(x - x_1)^2(x - x_2) dx = \frac{1}{4!} f^{(iv)}(\alpha) \int_{-h}^h (t + h)t^2(t - h) dt = -\frac{1}{90} f^{(iv)}(\alpha) h^5.$$

Portanto, a regra de Simpson para integração no intervalo  $(x_0, x_2)$  fica

$$\boxed{\int_a^b f(x) dx = \frac{h}{3} (f_0 + 4f_1 + f_2) - \frac{1}{90} f^{(iv)}(\alpha) h^5.} \quad (4.5)$$

A figura 4.4 ilustra a aplicação da regra de Simpson para o cálculo da quadratura. Das expressões obtidas para os erros das duas fórmulas polinomiais, (4.4) e (4.5), pode-se observar que a fórmula dos trapézios é exata se  $f(x)$  for um polinômio de grau 1 (pois  $f''(x) = 0$ ,  $\forall x$ ), ao passo que a fórmula de Simpson é exata se  $f(x)$  for um polinômio de grau igual ou menor que 3 (pois  $f^{(iv)}(x) = 0$ ,  $\forall x$ ).

### 4.2.1.3 REGRA DE SIMPSON DOS 3/8 ( $N = 3$ )

Existe uma regra de quatro pontos cujo erro é da mesma ordem de grandeza da regra de Simpson (4.5). Tomando  $N = 3$ ,  $x_0 = a$ ,  $x_3 = b$  e  $h = (b - a)/3$ , pode-se mostrar:

$$\int_a^b f(x)dx = \frac{3}{8}h(f_0 + 3f_1 + 3f_2 + f_3) - \frac{3}{80}f^{(iv)}(\alpha)h^5. \quad (4.6)$$

Pode-se ver que o erro é da mesma ordem de grandeza que (4.5).

### 4.2.1.4 REGRA DE BODE ( $N = 4$ )

Esta regra usa 5 valores de  $f(x)$  regularmente espaçados. Tomando  $N = 4$ ,  $x_0 = a$ ,  $x_4 = b$  e  $h = (b - a)/4$ , obtém-se

$$\int_a^b f(x)dx = \frac{2}{45}h(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4) - \frac{8}{945}f^{(vi)}(\alpha)h^7. \quad (4.7)$$

Pode-se observar que agora o erro de truncamento ( $\epsilon_B \sim h^7$ ) é bem menor que o erro obtido pela regra de Simpson.

### 4.2.1.5 REGRAS EM ORDENS MAIS ALTAS ( $N \geq 5$ )

Outras expressões, com erros de truncamento sucessivamente menores, podem ser obtidas aumentando-se o grau dos polinômios interpoladores. Em contrapartida, é necessário calcular  $f(x)$  em um número cada vez maior de pontos e a quantidade total de operações de ponto flutuante também aumenta. A relação de compromisso entre a precisão obtida, o esforço computacional necessário e o erro de arredondamento resultante vai depender então da aplicação em estudo. Essas outras fórmulas fechadas de quadratura podem ser obtidas, por exemplo, em Abramowitz & Stegun [21, seção 25.4].

Escrevendo-se uma forma geral para as fórmulas fechadas de Newton-Cotes:

$$\int_a^b f(x)dx = dh \sum_{i=0}^N f_i w_i + \epsilon_N, \quad (4.8)$$

a tabela 4.1 apresenta os valores dos parâmetros  $d$  e  $h$ , dos pesos  $w_i$  e dos erros de truncamento  $\epsilon_N$ .

**EXEMPLO 4.1.** Sabe-se que

$$\ln 2 = \int_1^2 \frac{dx}{x} = 0,6931471805599453094172322145818 \dots$$

Usando as fórmulas de Newton-Cotes (4.4 – 4.7), obtém-se os seguintes resultados para  $\ln 2$ .

**Regra trapezoidal.** Com  $h = 1$ :

$$\int_1^2 \frac{dx}{x} \approx \frac{1}{2} \left( 1 + \frac{1}{2} \right) = 0,75; \text{ erro relativo: } 8,2\%.$$

**Tabela 4.1:** Fórmulas fechadas de Newton-Cotes, dadas por (4.8).

$N$	$d$	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$\epsilon_N$
1	1/2	1	1							$-\frac{1}{12}f''(\alpha)h^3$
2	1/3	1	4	1						$-\frac{1}{90}f^{(iv)}(\alpha)h^5$
3	3/8	1	3	3	1					$-\frac{3}{80}f^{(iv)}(\alpha)h^5$
4	2/45	7	32	12	32	7				$-\frac{8}{945}f^{(vi)}(\alpha)h^7$
5	5/288	19	75	50	50	75	19			$-\frac{275}{12096}f^{(vi)}(\alpha)h^7$
6	1/140	41	216	27	272	27	216	41		$-\frac{9}{1400}f^{(viii)}(\alpha)h^9$
7	7/17280	751	3577	1323	2989	2989	1323	3577	751	$-\frac{8183}{518400}f^{(viii)}(\alpha)h^9$



**Regra de Simpson.** Com  $h = 1/2$ :

$$\int_1^2 \frac{dx}{x} \approx \frac{1}{6} \left( 1 + \frac{8}{3} + \frac{1}{2} \right) = 0,69444 \dots; \text{ erro relativo: } 0,19\%.$$

**Regra de Simpson dos 3/8.** Com  $h = 1/3$ :

$$\int_1^2 \frac{dx}{x} \approx \frac{3}{24} \left( 1 + \frac{9}{4} + \frac{9}{5} + \frac{1}{2} \right) = 0,69375; \text{ erro relativo: } 0,09\%.$$

**Regra de Bode.** Com  $h = 1/4$ :

$$\int_1^2 \frac{dx}{x} \approx \frac{1}{90} \left( 7 + 32\frac{4}{5} + 12\frac{2}{3} + 32\frac{4}{7} + \frac{7}{2} \right) = 0,69317460 \dots; \text{ erro relativo: } 0,004\%.$$

### 4.2.2 FÓRMULAS ABERTAS DE NEWTON-COTES

Uma fórmula aberta não utiliza os pontos extremos no intervalo de integração. Na figura 4.2 estes métodos utilizariam os pontos  $x_1, x_2, \dots, x_{N-1}$ , ou seja, fariam uso de  $N-1$  pontos. A principal motivação para o emprego de uma fórmula aberta ocorre quando o integrando apresenta um comportamento não usual próximo ao(s) limite(s) de integração, como uma singularidade, por exemplo.

Contudo, as fórmulas abertas raramente são empregadas, pelas seguintes razões:

1. Fórmulas abertas não podem ser facilmente compostas juntas para formar uma regra estendida, como as fórmulas fechadas, que serão discutidas na seção 4.2.3.
2. Há outras classes de fórmulas de quadratura abertas largamente superiores às fórmulas de Newton-Cotes. Um exemplo consiste nas fórmulas de quadratura gaussianas.
3. O polinômio interpolador raramente reproduz fidedignamente a forma de  $f(x)$  próxima aos pontos singulares, o que reduz significativamente a utilidade de uma fórmula aberta.

Devido a estas razões, as fórmulas abertas não serão detalhadamente discutidas aqui. Somente será apresentada a fórmula geral para uma quadratura aberta,

$$\int_a^b f(x)dx = dh \sum_{i=1}^{N-1} f_i w_i + \epsilon_N, \quad (4.9)$$

sendo que os parâmetros  $d$  e  $h$  e os pesos  $w_i$  podem ser obtidos em [21, seção 25.4] e são dados na tabela 4.2. Dentre as fórmulas apresentadas na tabela, a mais útil é a **regra do ponto médio** (*midpoint rule*), correspondente a  $M = 0$ , na tabela 4.2.

**Tabela 4.2:** Fórmulas abertas de Newton-Cotes, dadas por (4.9). Na tabela,  $h = (b-a)/N$  e  $N = M + 2$ .

$M$	$d$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$\epsilon_N$
0	2	1							$\frac{1}{3}f''(\alpha)h^3$
1	3/2	1	1						$\frac{1}{4}f''(\alpha)h^3$
2	4/3	2	-1	2					$\frac{28}{90}f^{(iv)}(\alpha)h^5$
3	5/24	11	1	1	11				$\frac{95}{144}f^{(iv)}(\alpha)h^5$
4	6/20	11	-14	26	-14	11			$\frac{41}{140}f^{(vi)}(\alpha)h^7$
5	7/1440	611	-453	562	562	-453	611		$\frac{5257}{8640}f^{(vi)}(\alpha)h^7$
6	8/945	460	-954	2196	-2459	2196	-954	460	$\frac{3956}{14175}f^{(viii)}(\alpha)h^9$

### 4.2.3 FÓRMULAS FECHADAS ESTENDIDAS

Quando o intervalo de integração é grande, pode não ser conveniente aumentar o grau do polinômio interpolador para estabelecer fórmulas de integração mais precisas, uma vez que estas fórmulas tornam-se gradativamente mais complicadas com o aumento do grau do polinômio.

A alternativa mais empregada neste caso é subdividir o intervalo de integração e aplicar as fórmulas introduzidas na seção 4.2.1 repetidas vezes. Assim, são obtidas as *fórmulas estendidas* ou *compostas*.

**Algoritmo 4.1** Implementação da regra trapezoidal estendida.

**Dados:**  $h, f_i = f(x_i)$  para  $i = 0, 1, \dots, N$ .

1. soma=0
2. **Para**  $i = 1 : N - 1$ , **faça**
3.     soma= soma +  $f_i$
4.  $I_{TE} = \frac{h}{2} [2 \text{soma} + f_0 + f_N]$

#### 4.2.3.1 REGRA TRAPEZOIDAL ESTENDIDA

Divide-se o intervalo de integração  $[a, b]$  em  $N$  subintervalos de igual comprimento  $h = (b - a)/N$ . Aplicando-se então a fórmula (4.4)  $N$  vezes para se realizar as integrações nos intervalos  $[x_0, x_1], [x_1, x_2], \dots, [x_{N-1}, x_N]$  e adicionando estas integrações parciais, obtém-se a fórmula trapezoidal estendida:

$$\begin{aligned} \int_a^b f(x)dx &= \int_{x_0}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \dots + \int_{x_{N-1}}^{x_N} f(x)dx \\ &= \frac{h}{2} (f_0 + f_1) + \frac{h}{2} (f_1 + f_2) + \dots + \frac{h}{2} (f_{N-1} + f_N) \\ &\quad - \frac{1}{12} [f''(\alpha_1) + f''(\alpha_2) + \dots + f''(\alpha_N)] h^3 \\ &= \frac{h}{2} (f_0 + 2f_1 + 2f_2 + \dots + 2f_{N-1} + f_N) - \frac{1}{12} \sum_{i=1}^N f''(\alpha_i) h^3. \end{aligned}$$

Pode-se mostrar, usando o Teorema da Média, que

$$\sum_{i=1}^N f''(\alpha_i) = N f''(\beta), \text{ onde } \beta \in [a, b].$$

Portanto, obtém-se a regra trapezoidal estendida:

$$\boxed{\int_a^b f(x)dx = \frac{1}{2} h (f_0 + 2f_1 + 2f_2 + \dots + 2f_{N-1} + f_N) - \frac{1}{12} (b - a) f''(\beta) h^2.} \quad (4.10)$$

Pode-se notar que agora o erro de truncamento é proporcional a  $\epsilon_T \sim h^2$  ao passo que o erro para a fórmula (4.4) é proporcional a  $h^3$ . Portanto, em princípio o erro aumentou na fórmula estendida. Contudo, deve-se salientar que os espaçamentos nas fórmulas (4.4) e (4.10) têm valores distintos, o que não possibilita uma comparação direta entre ambos.

A regra trapezoidal estendida pode ser implementada por um programa de computador com base no algoritmo 4.1. Um exemplo pode ser visto na rotina `trapez.f90`, a qual pode ser acessada em [www.ufpel.edu.br/~rudi/grad/ModComp/Progs/trapez.f90](http://www.ufpel.edu.br/~rudi/grad/ModComp/Progs/trapez.f90).

#### 4.2.3.2 REGRA DE SIMPSON ESTENDIDA

Para implementar a regra de Simpson estendida, é necessário dividir o intervalo  $[a, b]$  em um número par de subintervalos, o que corresponde a um número total ímpar de pontos no conjunto  $\{x_i\}$ , isto é, a  $N$  par, uma vez que cada integração parcial será realizada com o uso de 3 pontos para a interpolação parabólica.

Assim, se  $N$  é um número par,

$$\begin{aligned} \int_a^b f(x)dx &= \int_{x_0}^{x_2} f(x)dx + \int_{x_2}^{x_4} f(x)dx + \dots + \int_{x_{N-2}}^{x_N} f(x)dx \\ &= \frac{h}{3} (f_0 + 4f_1 + f_2) + \frac{h}{3} (f_2 + 4f_3 + f_4) + \dots + \frac{h}{3} (f_{N-2} + 4f_{N-1} + f_N) \\ &\quad - \frac{1}{90} f^{(iv)}(\alpha_1) h^5 - \frac{1}{90} f^{(iv)}(\alpha_2) h^5 - \dots - \frac{1}{90} f^{(iv)}(\alpha_{N/2}) h^5, \end{aligned}$$

**Algoritmo 4.2** Implementação da regra de Simpson estendida.

**Dados:**  $N$  par,  $h$ ,  $f_i = f(x_i)$  para  $i = 0, 1, \dots, N$ .

1. soma = 0
2. **Para**  $i = 1, 3, 5, \dots, N - 1$ , **faça**
3.     soma = soma +  $2f_i + f_{i+1}$
4.  $I_{SE} = \frac{h}{3} (2 \text{ soma} + f_0 - f_N)$ .

$$\int_a^b f(x)dx = \frac{h}{3} (f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \dots + 2f_{N-2} + 4f_{N-1} + f_N) - \frac{h^5}{90} \sum_{i=1}^{N/2} f^{(iv)}(\alpha_i).$$

A aplicação do Teorema da Média neste caso também fornece a seguinte expressão:

$$\sum_{i=1}^{N/2} f^{(iv)}(\alpha_i) = \frac{N}{2} f^{(iv)}(\gamma), \quad a \leq \gamma \leq b.$$

Assim,

$$\int_a^b f(x)dx = \frac{h}{3} (f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \dots + 2f_{N-2} + 4f_{N-1} + f_N) - \frac{b-a}{180} f^{(iv)}(\gamma) h^4. \quad (4.11)$$

Aqui também, embora o erro da regra de Simpson estendida seja aparentemente maior que na regra (4.5), os valores de  $h$  em ambos os casos em geral são bastante diferentes. O algoritmo 4.2 mostra a implementação da regra de Simpson estendida por um programa de computador. Um exemplo de rotina que implementa esta algoritmo pode ser visto na função `simpson.f90`, a qual pode ser acessada em [www.ufpel.edu.br/~rudi/grad/ModComp/Progs/simpson.f90](http://www.ufpel.edu.br/~rudi/grad/ModComp/Progs/simpson.f90).

**EXEMPLO 4.2.** Ainda calculando aproximações para  $\ln 2$ , pode-se agora aplicar as fórmulas compostas.

**Regra trapezoidal estendida.** Tomando quatro trapézios no intervalo  $[1, 2]$ , resulta  $N = 4$ ,  $h = 0,25$  e

$$\int_1^2 \frac{dx}{x} \approx \frac{0,25}{2} \left( 1 + \frac{8}{5} + \frac{4}{3} + \frac{8}{7} + \frac{1}{2} \right) = 0,6970238 \dots; \text{ erro relativo: } 0,56\%.$$

**Regra de Simpson estendida.** Tomando duas parábolas na fórmula estendida de de Simpson, resulta  $N = 4$ ,  $h = 0,25$  e

$$\int_1^2 \frac{dx}{x} \approx \frac{0,25}{3} \left( 1 + \frac{16}{5} + \frac{4}{3} + \frac{16}{7} + \frac{1}{2} \right) = 0,693253968 \dots; \text{ erro relativo: } 0,015\%.$$

Comparando os resultados deste exemplo com o anterior, percebe-se que para cada regra individual, o erro obtido foi menor, contudo, o erro é maior quando se comparam métodos que utilizam o mesmo número de pontos. Por exemplo, o método trapezoidal estendido obteve um erro maior que a regra de Simpson dos  $3/8$ , a qual também utiliza 4 pontos para calcular a quadratura.

## 4.2.4 FÓRMULAS ABERTAS ESTENDIDAS

Somente será apresentada aqui a fórmula estendida da regra do ponto médio (equação 4.9 com  $M = 0$ ). Esta é uma das poucas fórmulas que pode ser estendida com relativa facilidade e que permanece útil para a integração automática, discutida na seção 4.4. A fórmula do ponto médio estendida será apresentada na forma de um teorema, o qual será apresentado sem demonstração.

**Teorema.** Seja  $f(x)$  uma função diferenciável até a ordem 2 em  $[a, b]$ ,  $n \in \mathbb{N}$  par,  $h = (b - a)/(n + 2)$  e  $x_j = a + (j + 1)h$  para  $j = -1, 0, \dots, n + 1$ . Existe um  $\alpha \in (a, b)$  para o qual a **regra do ponto médio estendida** para  $n/2 + 1$  sub-intervalos pode ser escrita como

$$\int_a^b f(x) dx = 2h \sum_{j=0}^{n/2} f(x_{2j}) + \frac{b-a}{6} f''(\alpha) h^2. \quad (4.12)$$

### 4.2.5 ESTIMATIVAS DE ERRO NAS FÓRMULAS DE NEWTON-COTES

Embora as fórmulas de Newton-Cotes estudadas nas seções anteriores apresentem todas uma expressão para o erro de truncamento, na prática, a aplicação das expressões não é factível. Isto porque expressões como as obtidas nas fórmulas (4.10) e (4.11),

$$\epsilon_{TE} = -\frac{1}{12}(b-a)f''(\beta)h^2, \quad (a \leq \beta \leq b) \text{ e } \epsilon_{SE} = -\frac{b-a}{180}f^{(iv)}(\gamma)h^4, \quad (a \leq \gamma \leq b)$$

têm o seu cálculo impedido pelo desconhecimento do valor exato de  $\beta$  e  $\gamma$ .

Entretanto, há maneiras de se realizar estimativas dos valores máximos que estes erros podem assumir e a partir destas estimativas pode-se calcular o valor ideal para  $h$ , o espaçamento entre os pontos, que permite satisfazer um requisito inicial de valor máximo para os erros.

Ou seja, se o intervalo de integração for fechado e  $f(x)$  tiver derivadas contínuas neste intervalo até uma ordem  $k \geq N$ , onde  $N$  é o grau da regra de Newton-Cotes empregada, sempre é possível escrever

$$|\epsilon_{TE}| = \frac{1}{12}(b-a)|f''(\beta)|h^2 \leq \frac{1}{12}(b-a)h^2 \max_{a \leq x \leq b} |f''(x)| \quad (4.13a)$$

$$|\epsilon_{SE}| = \frac{b-a}{180} |f^{(iv)}(\gamma)| h^4 \leq \frac{b-a}{180} h^4 \max_{a \leq x \leq b} |f^{(iv)}(x)|, \quad (4.13b)$$

sendo que agora é necessário realizar estimativas para os valores máximos das derivadas de  $f(x)$  dentro do intervalo considerado. Estas estimativas podem ser realizadas de diversas maneiras, tanto numericamente quanto analiticamente. O exemplo abaixo ilustra a aplicação destas estimativas.

**EXEMPLO 4.3.** Quantos subintervalos e qual o espaçamento que devem ser empregados no cálculo de

$$\int_0^1 e^{-x^2} dx$$

para que a aproximação tenha erro menor que  $10^{-4}$  nos casos (a) Regra dos Trapézios Estendida e (b) Regra de Simpson Estendida?

*Respostas:* o integrando não possui primitiva, mas as suas derivadas podem ser calculadas em qualquer ordem. Assim,

$$\begin{aligned} f'(x) &= -2xe^{-x^2} & f''(x) &= 2(2x^2 - 1)e^{-x^2} \\ f'''(x) &= 4x(3 - 2x^2)e^{-x^2} & f^{(iv)}(x) &= 4(3 - 12x^2 + 4x^4)e^{-x^2} \\ f^{(v)}(x) &= -8x(15 - 20x^2 + 4x^4)e^{-x^2}. \end{aligned}$$

(a) Para a fórmula dos trapézios, de acordo com (4.13a) é necessário então encontrar o máximo de  $f''(x)$  no intervalo  $[0, 1]$ . Isto é possível uma vez que se conhece a única raiz de  $f'''(x)$  neste intervalo,  $r = 0$ . Portanto,

$$\max_{0 \leq x \leq 1} |f''(x)| = |f''(r)| = 2$$

e assim,

$$|\epsilon_{TE}| \leq \frac{1}{6} h_{TE}^2 \leq 10^{-4} \implies h_{TE} \leq \sqrt{6 \times 10^{-4}} \simeq 0,0245,$$

o que corresponde a um número de subintervalos

$$N_{TE} \geq \text{Int} \left( \frac{1}{h_{TE}} \right) = 41.$$

(b) Para a fórmula de Simpson, de acordo com (4.13b) é necessário encontrar o máximo de  $f^{(iv)}(x)$  no intervalo  $[0, 1]$ . Isto é possível uma vez que se conhecem as raízes de  $f^{(v)}(x)$  neste intervalo:

$$r_1 = 0, \quad r_2 = \sqrt{\frac{1}{2} \left( 5 - \sqrt{10} \right)} \simeq 0,9586.$$

Como

$$\left| f^{(iv)}(r_1) \right| = 12 \text{ e } \left| f^{(iv)}(r_2) \right| = 16 \left( \sqrt{10} - 2 \right) \exp \left( -\frac{5}{2} + \sqrt{\frac{5}{2}} \right) \simeq 7,4195,$$

obtem-se

$$\max_{0 \leq x \leq 1} \left| f^{(iv)}(x) \right| = \left| f^{(iv)}(r_1) \right| = 12$$

e, portanto,

$$|\epsilon_{SE}| \leq \frac{h_{SE}^4}{15} \leq 10^{-4} \implies h_{SE} \leq \sqrt[4]{15 \times 10^{-4}} \simeq 0,197,$$

o que corresponde a um número de subintervalos

$$N_{SE} \geq \text{Int} \left( \frac{1}{h_{Se}} \right) = 6.$$

Pode-se ver, portanto, que o número de subintervalos necessários para a regra de Simpson atingir um determinado limiar de erro é substancialmente menor que o número requerido pela regra trapezoidal.

## 4.3 QUADRATURA GAUSSIANA

Nas fórmulas da seção 4.2, a quadratura de uma função foi aproximada pela soma de seus valores funcionais em um conjunto de pontos regularmente espaçados  $\{f(x_i)\}$ , multiplicados pelos pesos  $\{w_i\}$ . Observou-se que escolhas adequadas nos pesos  $w_i$  permitem a obtenção de fórmulas de quadratura de ordens cada vez mais altas.

Nesta seção, serão introduzidas fórmulas de quadratura nas quais não somente os pesos  $\{w_i\}$  na fórmula genérica (4.1) poderão ser escolhidos, mas também as abscissas  $\{x_i\}$  serão determinadas de tal forma que a quadratura resultante será *superacurada*. Uma vez que as abscissas não serão mais regularmente espaçadas, as fórmulas obtidas terão o dobro de graus de liberdade que as fórmulas de Newton-Cotes possuem, resultando em fórmulas de quadratura de ordem essencialmente duas vezes maior que as fórmulas de Newton-Cotes, com o mesmo número de cálculos do integrando.

Esta idéia foi inicialmente introduzida por Wilhelm Gauss (1814), portanto cerca de um século após a introdução das fórmulas de Newton-Cotes. Por esta razão, estas fórmulas são conhecidas como **Fórmulas gaussianas** ou **Quadratura gaussiana**. Na sua formulação original, Gauss utilizou frações continuadas na obtenção de suas fórmulas. Em 1826, Jacobi derivou novamente as fórmulas gaussianas, agora utilizando polinômios ortogonais. O tratamento sistemático de funções-peso arbitrarias  $W(x)$  usando os polinômios ortogonais, da forma como hoje são usualmente empregadas as fórmulas gaussianas, é devido em grande parte a Christoffel, em 1877.

O conceito de polinômios ortogonais frente a uma função-peso  $W(x)$  no intervalo  $(a, b)$  se deve à definição de ortogonalidade de duas funções reais  $f(x)$  e  $g(x)$ , pertencentes ao espaço vetorial das funções contínuas por partes em  $(a, b)$  frente a uma função peso  $W(x)$ . Uma condição suficiente para que  $f$  e  $g$  sejam ortogonais é que o seu produto interno seja nulo,

$$\langle f|g \rangle \equiv \int_a^b W(x) f(x) g(x) dx = 0.$$

Adicionalmente, se o produto interno  $\sqrt{\langle f|f \rangle}$ , definido como a *norma* de  $f(x)$ , for unitário, então  $f(x)$  é dita *normalizada*. Um conjunto de vetores  $\{f_i(x)\}$ ,  $i = 0, 1, 2, \dots$ , simultaneamente ortogonais entre si e individualmente normalizados é denominado de *conjunto ortonormal*.

O emprego de polinômios ortogonais para a obtenção das fórmulas gaussianas será apresentado nas seções a seguir.

### 4.3.1 IDÉIA BÁSICA NA QUADRATURA GAUSSIANA

A idéia básica consiste em escrever a fórmula geral de quadratura (4.1) da seguinte maneira:

$$\int_a^b F(x)dx \equiv \int_a^b W(x)f(x)dx \approx \sum_{i=1}^N w_i f(x_i),$$

onde o integrando é escrito  $F(x) \equiv W(x)f(x)$ , sendo que  $W(x)$  passa a desempenhar o papel de função-peso na fórmula gaussiana. A escolha da forma de  $W(x)$  pode ser feita de tal modo que o integrando restante,  $f(x)$ , resulte ser o mais suave possível, ou de forma a salientar possíveis singularidades em  $F(x)$ . Isto é necessário para que  $f(x)$  possa ser satisfatoriamente aproximada por um polinômio. Um exemplo seria a quadratura adequada para aproximar a integral

$$\int_{-1}^1 \frac{\exp(-\cos^2 x)}{\sqrt{1-x^2}} dx.$$

A escolha natural para a função-peso seria

$$W(x) = \frac{1}{\sqrt{1-x^2}}.$$

Esta escolha em particular define o uso da fórmula de Gauss-Chebyshev, conforme visto na seção 4.3.2. Há um conjunto particular de formas para  $W(x)$  que constitui as fórmulas gaussianas tradicionais e que possuem valores tabelados para os pesos  $\{w_i\}$  e as abscissas  $\{x_i\}$ . Algumas destas formas tradicionais serão estudadas na seção 4.3.2.

Então, se para um polinômio de grau  $k$  qualquer,  $p_k(x)$ , vale a igualdade

$$\int_a^b p_k(x)dx = \sum_{i=1}^N w_i p_k(x_i), \quad (4.14)$$

determina-se o conjunto  $\{x_i, w_i\}$  de tal forma que a igualdade acima vale para qualquer polinômio de grau  $\leq k$ . Em princípio, esta escolha não introduz vantagem nenhuma em relação ao uso dos polinômios de Legendre, usados nas fórmulas de Newton-Cotes, pois estes também são exatamente representados por esta fórmula de quadratura, como se pode notar nas equações (4.3a-c). A vantagem consiste na escolha de um conjunto de polinômios ortogonais e nas suas raízes para as abscissas. Neste caso, conforme se demonstra no teorema abaixo, a fórmula (4.14) será exata para polinômios de grau  $\leq 2k+1$ ! Neste sentido que se referiu às fórmulas gaussianas como *superacuradas*.

#### TEOREMA 4.1

Sejam

1.  $\psi_k(x)$ ,  $k = 0, 1, \dots, N$ , polinômios de grau  $k$ , ortogonais relativamente ao produto interno

$$\langle \psi_i | \psi_j \rangle = \int_a^b \psi_i(x) \psi_j(x) dx = 0, \text{ para } j \neq i.$$

2.  $\{x_i\}$ ,  $i = 0, 1, \dots, N$  as raízes de  $\psi_{k+1}(x)$ .

Se a fórmula de quadratura

$$\int_a^b f(x)dx \approx \sum_{i=1}^N w_i p_k(x_i)$$

é exata para polinômios de grau  $\leq N$ , então ela também será exata para polinômios de grau  $\leq 2N+1$ .

**DEMONSTRAÇÃO.** Se  $p_{2N+1}(x)$  é um polinômio qualquer de grau  $\leq 2N + 1$ , então este pode ser escrito como:

$$p_{2N+1}(x) = \psi_{N+1}(x)q_N(x) + r_N(x),$$

onde  $q_N(x)$  e  $r_N(x)$  são polinômios de grau  $\leq N$  e  $\psi_{N+1}(x)$  é um polinômio de grau  $\leq N + 1$  da família ortogonal. Integrando esta expressão no intervalo  $(a, b)$ :

$$\int_a^b p_{2N+1}(x)dx = \int_a^b \psi_{N+1}(x)q_N(x)dx + \int_a^b r_N(x)dx,$$

observa-se que  $q_N(x)$  sempre pode ser escrito na forma de uma combinação linear dos polinômios  $\psi_1(x), \dots, \psi_N(x)$ , pois estes são ortogonais. Portanto,

$$\int_a^b \psi_{N+1}(x)q_N(x)dx = 0.$$

Assim, se for usada a quadratura exata para  $r_N(x)$ , dada por (4.14), resulta

$$\int_a^b p_{2N+1}(x)dx = \int_a^b r_N(x)dx = \sum_{i=1}^N w_i r_N(x_i).$$

Lembrando agora que  $r_N(x) = p_{2N+1}(x) - \psi_{N+1}(x)q_N(x)$ , obtém-se

$$\int_a^b p_{2N+1}(x)dx = \sum_{i=1}^N w_i [p_{2N+1}(x_i) - \psi_{N+1}(x_i)q_N(x_i)].$$

Agora, se as abscissas  $\{x_i\}$ ,  $i = 0, 1, \dots, N$ , forem escolhidas como as raízes do polinômio  $\psi_{N+1}(x)$ , isto é,  $\psi_{N+1}(x_i) = 0$ ,  $\forall i = 0, 1, \dots, N$ , resulta finalmente

$$\int_a^b p_{2N+1}(x)dx = \sum_{i=1}^N w_i p_{2N+1}(x_i).$$

O que demonstra ser a quadratura (4.14) exata para um polinômio de grau  $\leq 2N + 1$ .

### 4.3.2 FÓRMULAS GAUSSIANAS CLÁSSICAS

Deseja-se então um conjunto de polinômios  $\{p_j(x)\}$  ( $j = 0, 1, 2, \dots$ ), mutuamente ortogonais frente a uma função peso  $W(x)$  no intervalo  $(a, b)$ ,

$$\langle p_i | p_j \rangle = \int_a^b W(x) p_i(x) p_j(x) dx = 0, \quad i \neq j.$$

Um procedimento garantido para gerar este conjunto é fornecido pela relação de recorrência

$$p_{-1}(x) \equiv 0 \tag{4.15a}$$

$$p_0(x) \equiv 1 \tag{4.15b}$$

$$p_{j+1}(x) = (x - a_j) p_j(x) - b_j p_{j-1}(x), \quad (j = 0, 1, 2, \dots), \tag{4.15c}$$

onde

$$a_j = \frac{\langle x p_j | p_j \rangle}{\langle p_j | p_j \rangle} \tag{4.16a}$$

$$b_j = \frac{\langle p_j | p_j \rangle}{\langle p_{j-1} | p_{j-1} \rangle}. \tag{4.16b}$$

A fórmula de quadratura gaussiana com  $N$  pontos, então, é:

$$\int_a^b W(x) f(x) dx = \sum_{j=1}^N w_j f(x_j) + R_N, \tag{4.17}$$

onde o conjunto de abscissas  $\{x_j\}$  ( $j = 1, \dots, N$ ) consiste nas raízes de  $p_N(x)$ ,

$$p_N(x_j) = 0, \quad j = 1, \dots, N, \quad \text{tais que } a < x_1 < x_2 < \dots < x_N < b, \quad (4.18)$$

o conjunto de pesos  $\{w_j\}$  é dado por

$$w_j = \frac{\langle p_{N-1} | p_{N-1} \rangle}{p_{N-1}(x_j) p'_N(x_j)} \quad (4.19)$$

e  $R_N$  é o erro de truncamento da quadratura.

O cálculo das regras de quadratura gaussiana clássicas envolvem, então, duas fases:

1. a geração dos polinômios ortogonais  $p_0(x), p_1(x), \dots, p_N(x)$  (4.15a-c) através da obtenção dos seus coeficientes  $\{a_j, b_j\}$  (4.16a,b);
2. a determinação das raízes de  $p_N(x)$  e o cálculos dos pesos associados  $\{w_j\}$  (4.19).

Para o caso dos polinômios ortogonais clássicos, os coeficientes  $\{a_j, b_j\}$  são explicitamente conhecidos e a primeira fase pode ser omitida. Contudo, caso se queira utilizar uma função peso  $W(x)$  não clássica, os respectivos polinômios devem ser deduzidos a partir das relações de recorrência (4.15a-c).

Caso se deseje calcular a quadratura de  $F(x) = W(x)f(x)$  em um intervalo  $(x_1, x_2)$ , distinto do intervalo  $(a, b)$  onde os polinômios  $p_j(x)$  são mutuamente ortogonais, basta realizar-se uma mudança na variável de integração,

$$x = \frac{b-a}{x_2-x_1}y + \frac{ax_2-bx_1}{x_2-x_1} \iff y = \frac{x_2-x_1}{b-a}x + \frac{bx_1-ax_2}{b-a},$$

de forma que

$$\int_{x_1}^{x_2} W(y)f(y)dy = \frac{x_2-x_1}{b-a} \int_a^b W(x)f(x)dx = \frac{x_2-x_1}{b-a} \sum_{j=1}^N w_j f(y_j) + R_N, \quad (4.20)$$

onde

$$y_j = \frac{x_2-x_1}{b-a}x_j + \frac{bx_1-ax_2}{b-a}, \quad j = 1, \dots, N,$$

sendo  $\{x_j\}$  as raízes de  $p_N(x)$  (4.18).

A seguir, serão vistas algumas das regras de quadratura gaussiana clássicas.

#### 4.3.2.1 FÓRMULA DE GAUSS-LEGENDRE

Esta fórmula pode ser utilizada quando  $W(x) = 1$ , juntamente com os polinômios de Legendre  $P_n(x)$ :

$$P_0(x) = 1 \quad P_1(x) = x \quad P_2(x) = \frac{1}{2}(3x^2 - 1) \quad P_3(x) = \frac{1}{2}(5x^3 - 3x), \\ (j+1)P_{j+1}(x) = (2j+1)xP_j(x) - jP_{j-1}(x),$$

os quais são ortogonais no intervalo  $(-1, 1)$ . Neste caso, a fórmula de Gauss-Legendre fica, a partir de (4.17),

$$\int_{-1}^1 f(x)dx = \sum_{j=1}^N w_j f(x_j) + R_N, \quad (4.21)$$

onde  $\{x_j\}$  ( $j = 1, \dots, N$ ) são as raízes de  $P_N(x)$ ,

$$P_N(x_j) = 0, \quad j = 1, \dots, N, \quad \text{tais que } -1 < x_1 < x_2 < \dots < x_N < 1 \quad (4.22)$$

e os pesos são

$$w_j = \frac{2}{(1-x_j^2)[P'_N(x_j)]^2}, \quad j = 1, \dots, N$$



**Tabela 4.3:** Abcissas  $\{x_j\}$  (raízes dos polinômios de Legendre) e pesos  $\{w_j\}$  para integração de Gauss-Legendre.

$\pm x_j$		$w_j$
	$N = 2$	
$1/\sqrt{3}$		1
	$N = 3$	
0		8/9
$\sqrt{3/5}$		5/9
	$N = 4$	
0.3399810435848562648		0.65214515486254614263
0.8611363115940525752		0.34785484513745385737
	$N = 5$	
0.00000000000000000000		128/225
0.53846931010568309104		0.47862867049936646804
0.90617984593866399280		0.23692688505618908751
	$N = 6$	
0.23861918608319690863		0.46791393457269104739
0.66120938646626451366		0.36076157304813860757
0.93246951420315202781		0.17132449237917034504
	$N = 7$	
0.00000000000000000000		512/1225
0.40584515137739716691		0.38183005050511894495
0.74153118559939443986		0.27970539148927666790
0.94910791234275852453		0.129484966168869693271

e o erro de truncamento é

$$R_N = \frac{2^{2N+1} (N!)^4}{(2N+1) [(2N)!]^3} f^{(2N)}(\xi), \quad (-1 < \xi < 1).$$

A tabela 4.3 mostra as abcissas e os pesos para as fórmulas de Gauss-Legendre até  $N = 7$ . Valores de  $\{x_j\}$  e  $\{w_j\}$  para  $N > 7$  podem ser encontrados em Abramowitz & Stegun [21, capítulo 25]. Valores exatos para as raízes e os pesos somente podem ser encontrados para um número finito de polinômios. Para os restantes, é necessário obter-se estas quantidades numericamente.

Caso seja necessário calcular a quadratura no intervalo  $(x_1, x_2)$  qualquer, a fórmula de Gauss-Legendre fica, a partir de (4.20),

$$\int_{x_1}^{x_2} f(y) dy = \frac{x_2 - x_1}{2} \int_{-1}^1 f(x) dx = \frac{x_2 - x_1}{2} \sum_{j=1}^N w_j f(y_j) + R_N, \quad (4.23)$$

sendo

$$y_j = \frac{x_2 - x_1}{2} x_j + \frac{x_1 + x_2}{2}, \quad j = 1, \dots, N.$$

**EXEMPLO 4.4.** Calcula-se novamente a quadratura  $\int_1^2 \frac{dx}{x}$ , porém agora utilizando-se as fórmulas de Gauss-Legendre para um intervalo geral (4.23), com  $x_1 = 1$  e  $x_2 = 2$ .

**FÓRMULA DE DOIS PONTOS.** Usando apenas 2 pontos,  $j = 1, 2$ ,  $N = 2$ , obtém-se

$$\int_1^2 \frac{dy}{y} \approx \frac{1}{2} \left( w_1 \frac{1}{y_1} + w_2 \frac{1}{y_2} \right),$$

onde

$$y_1 = \frac{1}{2}x_1 + \frac{3}{2}, \quad y_2 = \frac{1}{2}x_2 + \frac{3}{2},$$

$x_1$  e  $x_2$  são as raízes de

$$P_2(x_j) = \frac{1}{2} (3x_j^2 - 1) = 0 \implies \begin{cases} x_1 = -\frac{1}{\sqrt{3}} \\ x_2 = \frac{1}{\sqrt{3}} \end{cases}$$

e

$$w_j = \frac{2}{(1-x_j^2)[P_2'(x_j)]^2}, \quad j=1,2 \implies w_1 = w_2 = 1.$$

Portanto,

$$y_1 = \frac{3}{2} - \frac{1}{2\sqrt{3}}, \quad y_2 = \frac{3}{2} + \frac{1}{2\sqrt{3}}$$

$$\int_1^2 \frac{dy}{y} \approx \frac{1}{2} \left( \frac{1}{y_1} + \frac{1}{y_2} \right) = 0,692307692,$$

cujo erro relativo é de apenas 0,12%, o qual é um pouco melhor que o resultado obtido com a regra de Simpson (seção 4.2.1.2), para a qual foram necessários 3 pontos.

**FÓRMULA DE 3 PONTOS.** Usando-se 3 pontos,  $N = 3$  e

$$\int_1^2 \frac{dy}{y} \approx \frac{1}{2} \left( w_1 \frac{1}{y_1} + w_2 \frac{1}{y_2} + w_3 \frac{1}{y_3} \right),$$

onde

$$y_1 = \frac{1}{2}x_1 + \frac{3}{2}, \quad y_2 = \frac{1}{2}x_2 + \frac{3}{2}, \quad y_3 = \frac{1}{2}x_3 + \frac{3}{2},$$

sendo  $x_1, x_2$  e  $x_3$  as raízes de  $P_3(x)$ :

$$P_3(x_j) = \frac{1}{2}(5x_j^3 - 3x_j) = 0 \implies \begin{cases} x_1 = -\sqrt{\frac{3}{5}} \\ x_2 = 0 \\ x_3 = \sqrt{\frac{3}{5}} \end{cases}$$

e

$$w_j = \frac{2}{(1-x_j^2)[P_3'(x_j)]^2}, \quad j=1,2,3 \implies w_1 = w_3 = \frac{5}{9}, \quad w_2 = \frac{8}{9}.$$

Portanto,

$$y_1 = \frac{3}{2} - \frac{1}{2}\sqrt{\frac{3}{5}}, \quad y_2 = \frac{3}{2}, \quad y_3 = \frac{3}{2} + \frac{1}{2}\sqrt{\frac{3}{5}}$$

e

$$\int_1^2 \frac{dy}{y} \approx \frac{1}{2} \left( \frac{5}{9} \frac{1}{y_1} + \frac{8}{9} \frac{1}{y_2} + \frac{5}{9} \frac{1}{y_3} \right) = 0,693121693,$$

o qual tem um erro relativo de 0,0037%, um pouco melhor que o resultado obtido com a regra de Bode (seção 4.2.1.4), a qual necessitou de 5 pontos.

#### 4.3.2.2 FÓRMULA DE GAUSS-Chebyshev

Nesta fórmula são empregados os polinômios de Chebyshev:

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_2(x) = 2x^2 - 1, \quad T_3(x) = 4x^3 - 3x,$$

$$T_{j+1}(x) = 2xT_j(x) - T_{j-1}(x),$$

os quais são ortogonais no intervalo  $-1 < x < 1$  frente à função peso

$$W(x) = \frac{1}{\sqrt{1-x^2}}.$$

Neste caso, a fórmula de Gauss-Chebyshev fica, a partir de (4.17),

$$\boxed{\int_{-1}^1 \frac{f(x)}{\sqrt{1-x^2}} dx = \sum_{j=1}^N w_j f(x_j) + R_N,} \quad (4.24)$$

onde  $\{x_j\}$ , ( $j = 1, 2, \dots, N$ ) são as raízes de  $T_N(x) = 0$ ,

$$x_j = \cos \left( \frac{(j - \frac{1}{2})\pi}{N} \right)$$

e  $\{w_j\}$  são os pesos, dados simplesmente por

$$w_j = \frac{\pi}{N}.$$

O erro de truncamento  $R_N$  no uso da fórmula (4.24) é

$$R_N = \frac{\pi}{(2n)!2^{2n-1}} f^{(2n)}(\xi), \quad (-1 < \xi < 1).$$

Caso seja necessário calcular a quadratura no intervalo  $(x_1, x_2)$  qualquer, a fórmula de Gauss-Chebyshev fica, a partir de (4.20),

$$\int_{x_1}^{x_2} \frac{f(y) dy}{\sqrt{(y-x_1)(x_2-y)}} = \int_{-1}^1 \frac{f(x)}{\sqrt{1-x^2}} dx = \sum_{j=1}^N w_j f(y_j) + R_N, \quad (4.25)$$

sendo

$$y_j = \frac{x_2 - x_1}{2} x_j + \frac{x_1 + x_2}{2}, \quad j = 1, \dots, N.$$

A regra de Gauss-Chebyshev possui uma implementação numérica bastante simples. A rotina apresentada abaixo, função `gauss_chebyshev` (programa 4.1), ilustra como esta implementação pode ser realizada com o Fortran 95.

**Listing 4.1:** Implementação da fórmula de Gauss-Chebyshev em Fortran 95.

```
! ***** FUNÇÃO Gauss_Chebyshev *****
! Implementa a fórmula de Gauss-Chebyshev para uma integral definida de
!   limites arbitrários.
!
! Argumentos:
!   f: Função a ser integrada (integrando menos função peso).
!   x1: Limite inferior de integração.
!   x2: Limite superior de integração.
!   n: Número de pontos usados na quadratura gaussiana (n > 1).
!
! Autor: Rudi Gaelzer, IF-UFRGS.
! Data: Maio/2008.
!
function gauss_chebyshev(f,x1,x2,n)
  real(kind= dp)                :: gauss_chebyshev
  real(kind= dp), intent(in)    :: x1, x2
  integer, intent(in)           :: n
  real(kind= dp) :: x_menos, x_mais, y, xj, wj
  integer :: j
  INTERFACE
    function f(x)
      use Modelos_Computacionais_Dados
      real(kind= dp) :: f
      real(kind= dp), intent(in) :: x
    end function f
  END INTERFACE
!
  x_menos= 0.5*(x2 - x1)
  x_mais= 0.5*(x1 + x2)
  wj= pi/real(n,dp)
  gauss_chebyshev= 0.0_dp
  do j= 1, n
    xj= cos((j-0.5)*pi/real(n,dp))
    y= x_menos*xj + x_mais
    gauss_chebyshev= gauss_chebyshev + wj*f(y)
  end do
  return
end function gauss_chebyshev
```

**Tabela 4.4:** Abcissas  $\{x_j\}$  (raízes dos polinômios de Laguerre) e pesos  $\{w_j\}$  para integração de Gauss-Laguerre.

$x_j$	$N = 2$	$w_j$
$2 - \sqrt{2}$		$\frac{2-\sqrt{2}}{4(-1+\sqrt{2})^2}$
$2 + \sqrt{2}$		$\frac{2+\sqrt{2}}{4(1+\sqrt{2})^2}$
	$N = 3$	
0.41577455678347908331		0.71109300992917301545
2.29428036027904171982		0.27851773356924084880
6.28994508293747919866		0.01038925650158613575
	$N = 4$	
0.32254768961939231180		0.60315410434163360164
1.74576110115834657569		0.35741869243779968664
4.53662029692112798328		0.03888790851500538427
9.39507091230113312923		0.00053929470556132745
	$N = 5$	
0.26356031971814091020		0.52175561058280865281
1.41340305910651679222		0.39866681108317592745
3.59642577104072208122		0.07594244968170759539
7.08581000585883755692		0.00361175867992204845
12.6408008442757826594		0.00002336997238577623

#### 4.3.2.3 FÓRMULA DE GAUSS-LAGUERRE

Nesta fórmula são empregados os polinômios de Laguerre :

$$L_0(x) = 1, \quad L_1(x) = 1 - x, \quad L_2(x) = \frac{1}{2}(2 - 4x + x^2), \quad L_3(x) = \frac{1}{6}(6 - 18x + 9x^2 - x^3),$$

$$(j+1)L_{j+1}(x) = (2j+1-x)L_j(x) - jL_{j-1}(x),$$

os quais são ortogonais no intervalo  $0 \leq x < \infty$  frente à função peso

$$W(x) = e^{-x}.$$

Neste caso, a fórmula de Gauss-Laguerre fica, a partir de (4.17),

$$\int_0^\infty e^{-x} f(x) dx = \sum_{j=1}^N w_j f(x_j) + R_N, \quad (4.26)$$

onde  $\{x_j\}$  ( $j = 1, \dots, N$ ) são as raízes de  $L_N(x)$ ,

$$x_j \text{ tais que } L_N(x_j) = 0, \text{ onde } j = 1, \dots, N \text{ e } 0 < x_1 < x_2 < \dots < x_N$$

e  $\{w_j\}$  são os pesos, dados por

$$w_j = \frac{x_j}{N^2 [L_{N-1}(x_j)]^2}.$$

O erro de truncamento  $R_N$  no uso da fórmula (4.27) é

$$R_N = \frac{(N!)^2}{(2N)!} f^{(2N)}(\xi), \quad (0 < \xi < \infty).$$

A tabela 4.4 mostra as abcissas e os pesos para as fórmulas de Gauss-Laguerre até  $N = 5$ . Uma listagem mais completa pode ser encontrada em [21, capítulo 25].

#### 4.3.2.4 FÓRMULA DE GAUSS-HERMITE

Nesta fórmula são empregados os polinômios de Hermite:

$$H_0(x) = 1, \quad H_1(x) = 2x, \quad H_2(x) = 4x^2 - 2, \quad H_3(x) = 8x^3 - 12x,$$

**Tabela 4.5:** Abcissas  $\{x_j\}$  (raízes dos polinômios de Hermite) e pesos  $\{w_j\}$  para integração de Gauss-Hermite.

$\pm x_j$		$w_j$
	$N = 2$	
$1/\sqrt{2}$		$\sqrt{\pi}/2$
	$N = 3$	
$0.0$		$2\sqrt{\pi}/3$
$\sqrt{3}/2$		$\sqrt{\pi}/6$
	$N = 4$	
$0.52464762327529031788$		$0.80491409000551283651$
$1.65068012388578455588$		$0.08131283544724517714$
	$N = 5$	
$0.00000000000000000000$		$8\sqrt{\pi}/15$
$0.95857246461381850711$		$0.39361932315224115983$
$2.02018287045608563293$		$0.01995324205904591321$
	$N = 6$	
$0.43607741192761650868$		$0.72462959522439252409$
$1.33584907401369694971$		$0.15706732032285664392$
$2.35060497367449222283$		$0.00453000990550884564$

$$H_{j+1}(x) = 2xH_j(x) - 2jH_{j-1}(x),$$

os quais são ortogonais no intervalo  $-\infty < x < \infty$  frente à função peso

$$W(x) = e^{-x^2}.$$

Neste caso, a fórmula de Gauss-Hermite fica, a partir de (4.17),

$$\int_{-\infty}^{\infty} e^{-x^2} f(x) dx = \sum_{j=1}^N w_j f(x_j) + R_N, \quad (4.27)$$

onde  $\{x_j\}$  ( $j = 1, \dots, N$ ) são as raízes de  $H_N(x)$ ,

$$x_j \text{ tais que } H_N(x_j) = 0, \text{ onde } j = 1, \dots, N \text{ e } 0 < x_1 < x_2 < \dots < x_N$$

e  $\{w_j\}$  são os pesos, dados por

$$w_j = \frac{2^{N-1} N! \sqrt{\pi}}{N^2 [H_{N-1}(x_j)]^2}.$$

O erro de truncamento  $R_N$  no uso da fórmula (4.27) é

$$R_N = \frac{N! \sqrt{\pi}}{2^N (2N)!} f^{(2N)}(\xi), \quad (-\infty < \xi < \infty).$$

A tabela 4.5 mostra as abcissas e os pesos para as fórmulas de Gauss-Hermite até  $N = 6$ . Uma listagem mais completa pode ser encontrada em [21, capítulo 25].

## 4.4 INTEGRAÇÃO AUTOMÁTICA E ADAPTATIVA

Nesta seção serão abordadas, em menor grau de detalhe, técnicas mais avançadas para a quadratura numérica, tanto *via* fórmulas de Newton-Cotes quanto *via* fórmulas gaussianas. As técnicas aqui mencionadas fornecem, além do cálculo da quadratura, também a obtenção de uma estimativa de erro, o que possibilita o desenvolvimento de algoritmos que implementam o cálculo de uma quadratura com a imposição de um valor superior no seu erro, para qualquer integrando, de uma forma automática ou adaptativa.

As técnicas aqui mencionadas formam as bases teóricas de rotinas modernas para o cálculo de quadraturas, oferecidas por diversos pacotes comerciais de computação numérica.

### 4.4.1 INTEGRAÇÃO DE ROMBERG

Uma rotina de integração *automática* é aquela que, aplicando uma determinada regra de quadratura para valores consecutivamente menores de espaçamento entre os pontos da abscissa, calcula também uma estimativa de erro independente da forma específica de  $f(x)$ , interrompendo a sua execução quando o resultado estiver dentro de uma tolerância exigida pelo programador, a qual pode ser as estimativas de erro absoluto ou relativo.

Este tipo de algoritmo é relativamente simples de ser implementado usando as regras de Newton-Cotes; quando se utiliza a regra trapezoidal estendida (seção 4.2.3.1) para implementar uma rotina integradora automática baseada no método de extrapolação de Richardson (seção ??), esta rotina denomina-se *Integração de Romberg*.

De acordo com o método de extrapolação de Richardson, deve-se aplicar o algoritmo de integração para dois valores distintos do parâmetro  $h$ . A estimativa de erro então obtida pode ser utilizada tanto para realizar controle de erro quanto para a extrapolação. Relembrando os principais resultados desta regra, se a fórmula de quadratura é aplicada com o parâmetro  $h$ , obtendo o resultado  $I(h)$  e posteriormente para o espaçamento  $h/R$ , resultando  $I(h/R)$ , as fórmulas (??a, b) fornecem como estimativas de erro absoluto:

$$EA(h) \simeq \left[ I\left(\frac{h}{R}\right) - I(h) \right] \frac{R^n}{R^n - 1} \text{ e } EA\left(\frac{h}{R}\right) = \left[ I\left(\frac{h}{R}\right) - I(h) \right] \frac{1}{R^n - 1}, \quad (4.28a)$$

onde  $n$  é a ordem do erro da fórmula de quadratura. A fórmula de extrapolação é então dada por (??):

$$I_{\text{extrapolado}} = I\left(\frac{h}{R}\right) + EA\left(\frac{h}{R}\right) = \frac{1}{R^n - 1} [R^n I(h/R) - I(h)]. \quad (4.28b)$$

#### 4.4.1.1 INTEGRAIS DEFINIDAS DE ROMBERG

Será apresentada inicialmente a regra de Romberg para integrais definidas, para as quais pode-se fazer uso das fórmulas fechadas estendidas discutidas na seção 4.2.3. Para implementar uma integração automática neste caso, utiliza-se a regra trapezoidal estendida (4.10),

$$\int_a^b f(x)dx \approx \frac{1}{2}h(f_0 + 2f_1 + 2f_2 + \cdots + 2f_{N-1} + f_N) \equiv I_{TE}(h). \quad (4.29)$$

Pode-se mostrar [6, Eq. 4.2.1] que o erro total é dado por uma série de potências pares de  $h$ :<sup>2</sup>

$$EA_{TE} = \frac{1}{12}(f'_0 - f'_N)h^2 + \frac{1}{720}(f'''_0 - f'''_N)h^4 + \cdots + \frac{B_{2k}}{(2k)!}(f_0^{(2k-1)} - f_N^{(2k-1)})h^{2k} + \cdots,$$

sendo  $\{B_k\}$  o conjunto dos *números de Bernoulli*. Portanto,  $n = m = \ell = \cdots = 2$  nas fórmulas extrapoladas (?? - ??) e nas estimativas de erros (??, ??, ??). Desta forma, o resultado  $I_{\text{extrapolado}}$  possui um erro agora da ordem  $\mathcal{O}(h^4)$ .

Inicia-se o procedimento escolhendo um valor inicial para o parâmetro  $h$ , calculam-se as quadraturas numéricas  $I_{TE}(h)$  e  $I_{TE}(h/2)$  a partir de (4.29). Estes valores iniciais são identificados por  $I_R^{(0)}(h)$  e  $I_R^{(0)}(h/2)$ , respectivamente, com suas respectivas estimativas de erro obtidas para  $R = 2$ :

$$\begin{aligned} I_R^{(0)}(h) &\equiv I_{TE}(h) & I_R^{(0)}\left(\frac{h}{2}\right) &\equiv I_{TE}\left(\frac{h}{2}\right) \\ EA_0(h) &= \frac{2^2}{2^2 - 1} \left[ I_R^{(0)}\left(\frac{h}{2}\right) - I_R^{(0)}(h) \right] & EA_0\left(\frac{h}{2}\right) &= \frac{1}{2^2 - 1} \left[ I_R^{(0)}\left(\frac{h}{2}\right) - I_R^{(0)}(h) \right]. \end{aligned}$$

De acordo com (4.28), o valor extrapolado da quadratura passa a ser  $I_R^{(0)}(h/2) + EA_0(h/2)$ . Esta quantidade passa a ser identificada por  $I_R^{(1)}(h)$ :

$$I_R^{(1)}(h) = \frac{1}{2^2 - 1} \left[ 2^2 I_R^{(0)}\left(\frac{h}{2}\right) - I_R^{(0)}(h) \right],$$

<sup>2</sup>Esta é a *Fórmula de Euler-MacLaurin*. Uma derivação simplificada da mesma pode ser encontrada em DeVries (1994) [25, Cap. 4].

o qual possui um erro da ordem  $\mathcal{O}(h^4)$ . Contudo, não se conhece o valor deste erro; tudo o que se obteve até este momento foi o erro  $EA_0(h/2)$ , correspondente à aproximação  $I_R^{(0)}(h/2)$ .

Para se calcular o erro de  $I_R^{(1)}(h)$ , é necessário agora aplicar a fórmula (??), o que implica na necessidade do cálculo de  $I_R^{(1)}(h/2)$ . Desta forma, obtém-se  $EA_1(h)$  e  $EA_1(h/2)$ , dados por:

$$EA_1(h) = \frac{2^4}{2^4 - 1} \left[ I_R^{(1)}\left(\frac{h}{2}\right) - I_R^{(1)}(h) \right] \text{ e } EA_1\left(\frac{h}{2}\right) = \frac{1}{2^4 - 1} \left[ I_R^{(1)}\left(\frac{h}{2}\right) - I_R^{(1)}(h) \right].$$

O valor extrapolado agora passa a ser  $I_R^{(1)}(h/2) + EA_1(h/2)$ , o qual é identificado por  $I_R^{(2)}(h)$ :

$$I_R^{(2)}(h) = \frac{1}{2^4 - 1} \left[ 2^4 I_R^{(1)}\left(\frac{h}{2}\right) - I_R^{(1)}(h) \right],$$

cujo erro é da ordem  $\mathcal{O}(h^6)$ ; porém, a melhor estimativa de erro é  $EA_1(h/2)$ , correspondente a  $I_R^{(1)}(h/2)$ . Para se calcular o erro de  $I_R^{(2)}(h)$  é necessário calcular  $I_R^{(2)}(h/2)$ , o que reinicia o ciclo.

Para sistematizar, pode-se afirmar que, aplicando-se a regra trapezoidal estendida para uma sucessão de incrementos  $h$  cada vez menores, sendo que cada valor consecutivo de  $h$  é a metade do valor anterior ( $R = 2$ ), obtém-se de (4.29) as integrais de Romberg  $I_R^{(k)}(h)$ ,  $k = 0, 1, 2, \dots$ , e as melhores estimativas de erro, fornecidas por  $EA_{k-1}(h/2)$ , onde

$$\begin{aligned} I_R^{(0)}(h) &\equiv I_{TE}(h) \\ I_R^{(1)}(h) &= \frac{1}{2^2 - 1} \left[ 2^2 I_R^{(0)}\left(\frac{h}{2}\right) - I_R^{(0)}(h) \right], & EA_0\left(\frac{h}{2}\right) &= \frac{1}{2^2 - 1} \left[ I_R^{(0)}\left(\frac{h}{2}\right) - I_R^{(0)}(h) \right] \\ I_R^{(2)}(h) &= \frac{1}{2^4 - 1} \left[ 2^4 I_R^{(1)}\left(\frac{h}{2}\right) - I_R^{(1)}(h) \right], & EA_1\left(\frac{h}{2}\right) &= \frac{1}{2^4 - 1} \left[ I_R^{(1)}\left(\frac{h}{2}\right) - I_R^{(1)}(h) \right] \\ I_R^{(3)}(h) &= \frac{1}{2^6 - 1} \left[ 2^6 I_R^{(2)}\left(\frac{h}{2}\right) - I_R^{(2)}(h) \right], & EA_2\left(\frac{h}{2}\right) &= \frac{1}{2^6 - 1} \left[ I_R^{(2)}\left(\frac{h}{2}\right) - I_R^{(2)}(h) \right] \\ &\vdots & &\vdots \end{aligned}$$

Pode-se induzir o  $k$ -ésimo ( $k > 1$ ) valor extrapolado e o seu erro:

$$\boxed{\begin{aligned} I_R^{(k)}(h) &= \frac{1}{4^k - 1} \left[ 4^k I_R^{(k-1)}\left(\frac{h}{2}\right) - I_R^{(k-1)}(h) \right], \\ EA_{k-1}\left(\frac{h}{2}\right) &= \frac{1}{4^k - 1} \left[ I_R^{(k-1)}\left(\frac{h}{2}\right) - I_R^{(k-1)}(h) \right]. \end{aligned}} \quad (4.30)$$

Supondo então que se queira aplicar a regra de extrapolação (4.30) até  $k = 4$ . Para se obter  $I_R^{(4)}(h)$  é necessário calcular  $I_R^{(3)}(h)$ ,  $I_R^{(3)}(h/2)$ , o que implica em calcular antes  $I_R^{(2)}(h)$ ,  $I_R^{(2)}(h/2)$  e  $I_R^{(2)}(h/2^2)$ , para as quais são necessárias  $I_R^{(1)}(h)$ ,  $I_R^{(1)}(h/2)$ ,  $I_R^{(1)}(h/2^2)$  e  $I_R^{(1)}(h/2^3)$  e, finalmente,  $I_R^{(0)}(h)$ ,  $I_R^{(0)}(h/2)$ ,  $I_R^{(0)}(h/2^2)$ ,  $I_R^{(0)}(h/2^3)$  e  $I_R^{(0)}(h/2^4)$ . Ou seja, para uma extrapolação até o  $k$ -ésimo termo, é necessária a aplicação da quadratura trapezoidal para os intervalos  $h$ ,  $h/2$ ,  $\dots$ ,  $h/2^k$ , o que vai implicar em até  $N = 2^k$  subintervalos.

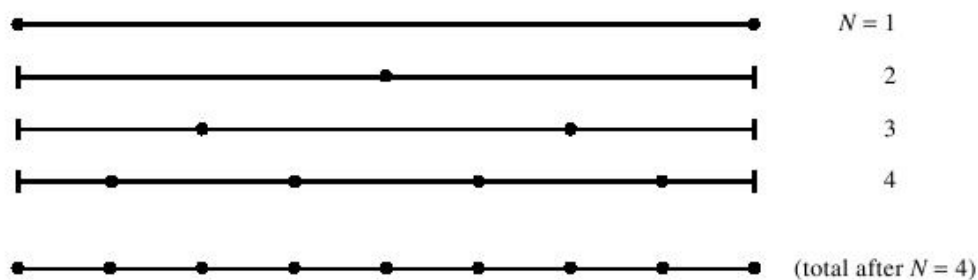
Para o cálculo da estimativa de erro  $EA_{k-1}(h/2)$ , é necessário que se conheça também  $I_R^{(0)}(h)$ ,  $\dots$ ,  $I_R^{(0)}(h/2^k)$ . O diagrama da figura 4.5 ilustra a interdependência entre os consecutivos estágios de extrapolação para  $k = 4$ . Generalizações para valores maiores de  $k$  são facilmente realizadas.

Antes de qualquer preocupação a respeito da implementação do controle de erro e critério de parada da rotina integradora, pode surgir agora a convicção de que o número de cálculos da quadratura (4.29) para as extrapolações se torna rapidamente tão grande que uma aplicação prática deste método se torna inviável. Felizmente, isto não é verdade. Para a regra trapezoidal entre limites fixos  $a$  e  $b$ , pode-se dobrar o número de subintervalos sem que se perca o trabalho realizado previamente. A implementação mais grosseira da regra trapezoidal seria tomar, na primeira chamada ( $N = 1$ ),  $h = b - a$ ,  $f_0 = f(a)$  e  $f_1 = f(b)$ , calculando-se então

$$I_R^{(0)}(h) = \frac{h}{2} (f_0 + f_1).$$

$$\begin{array}{ccccccc}
 I_R^{(0)}(h) & & & & & & \\
 & I_R^{(1)}(h) & & & & & \\
 I_R^{(0)}\left(\frac{h}{2}\right) & & I_R^{(2)}(h) & & & & \\
 & I_R^{(1)}\left(\frac{h}{2}\right) & & I_R^{(3)}(h) & & & \\
 I_R^{(0)}\left(\frac{h}{2^2}\right) & & I_R^{(2)}\left(\frac{h}{2}\right) & & I_R^{(4)}(h) & & \\
 & I_R^{(1)}\left(\frac{h}{2^2}\right) & & I_R^{(3)}\left(\frac{h}{2}\right) & & & \\
 I_R^{(0)}\left(\frac{h}{2^3}\right) & & I_R^{(2)}\left(\frac{h}{2^2}\right) & & I_R^{(4)}\left(\frac{h}{2}\right) & & \\
 & I_R^{(1)}\left(\frac{h}{2^3}\right) & & I_R^{(3)}\left(\frac{h}{2^2}\right) & & & \\
 I_R^{(0)}\left(\frac{h}{2^4}\right) & & I_R^{(2)}\left(\frac{h}{2^3}\right) & & & & \\
 & I_R^{(1)}\left(\frac{h}{2^4}\right) & & & & & \\
 I_R^{(0)}\left(\frac{h}{2^5}\right) & & & & & & 
 \end{array}$$

**Figura 4.5:** Integrais de Romberg  $I_R^{(k)}$  necessárias para o cálculo de  $I_R^{(4)}(h)$  e  $EA_4(h)$ , de acordo com (4.30). Nota-se que cada par de termos em uma dada coluna gera o termo centrado na coluna à direita.



**Figura 4.6:** Chamadas consecutivas da rotina que calcula a quadratura trapezoidal estendida incorporando a informação de chamadas anteriores e calculando o integrando somente nos novos pontos necessários para o refinamento da grade. A linha final mostra o número total de cálculos do integrando após as quarta chamada da rotina.



**Algoritmo 4.3** Calcula o  $n$ -ésimo refinamento da regra trapezoidal estendida (4.29), sendo dados  $f(x)$ , os limites de integração  $(a, b)$  e o resultado da quadratura no estágio anterior ( $I_{\text{Rom}}$ ). Os pontos incluídos em cada estágio são sempre distintos de todos os outros pontos anteriores, conforme ilustrado na figura 4.6. Quando chamado com  $n = 1$ , o algoritmo calcula a quadratura usando  $h = b - a$ ; quando chamado com  $n = 2, 3, \dots$ , o resultado será refinado pela adição de  $2^{n-2}$  pontos interiores adicionais.

**Dados:**  $f(x)$ ,  $a$ ,  $b$ ,  $n$  e  $I_{\text{Rom}}$ :

**Se**  $n = 1$  **então:**

$$I_{\text{Rom}} = (1/2) (b - a) [f(a) + f(b)]$$

**senão:**

$$npts = 2^{n-2}$$

$$\delta = (b - a) / npts$$

$$x = a + \delta/2$$

$$soma = 0$$

**Para**  $j = 1 : npts$ , **faça:**

$$soma = soma + f(x)$$

$$x = x + \delta$$

**final laço**

$$I_{\text{Rom}} = (1/2) [I_{\text{Rom}} + (b - a) soma / npts]$$

**final teste**

O primeiro estágio de refinamento consiste então em realizar a segunda chamada ( $N = 2$ ), na qual basta adicionar o valor da função no ponto central e realizar as transformações

$$(N = 2): \quad h \rightarrow \frac{h}{2}, \quad x_0 = a, \quad x_1 = a + \frac{b-a}{2}, \quad x_2 = b, \quad f_1 \rightarrow f_2 \quad \text{e} \quad f_1 = f(x_1),$$

resultando

$$I_R^{(0)}\left(\frac{h}{2}\right) = \frac{h}{2^2} (f_0 + 2f_1 + f_2) = \frac{1}{2} I_R^{(0)}(h) + \frac{h}{2} f_1;$$

o segundo estágio (chamada  $N = 3$ ) consiste na adição dos pontos em  $h/4$  e  $3h/4$ , através das transformações

$$(N = 3): \quad h \rightarrow \frac{h}{2}, \quad x_0 = a, \quad x_1 = a + \frac{1}{2} \frac{b-a}{2}, \quad x_2 = a + \frac{b-a}{2}, \quad x_3 = a + \frac{3}{2} \frac{b-a}{2}, \quad x_4 = b, \\ f_2 \rightarrow f_4, \quad f_1 \rightarrow f_2, \quad f_1 = f(x_1), \quad \text{e} \quad f_3 = f(x_3),$$

resultando

$$I_R^{(0)}\left(\frac{h}{2^2}\right) = \frac{h}{2^3} (f_0 + 2f_1 + 2f_2 + 2f_3 + f_4) = \frac{1}{2} I_R^{(0)}\left(\frac{h}{2}\right) + \frac{h}{2^2} (f_1 + f_3)$$

e assim consecutivamente. A figura 4.6 ilustra a aplicação prática desta idéia.

A implementação desta idéia é apresentada no algoritmo 4.3. Este algoritmo deve ser chamado pela rotina integradora para calcular os termos da primeira coluna do diagrama na figura 4.5. A primeira chamada deve ser realizada com  $n = 1$ , incrementando-se o valor de  $n$  por 1 a cada chamada subsequente, totalizando  $k + 1$  chamadas, sendo  $k$  o grau de extrapolação desejado na rotina de Romberg (4.30).

O algoritmo 4.3 está implementado em Fortran 95 na forma de uma função no programa 4.2.

**Listing 4.2:** Implementação do algoritmo 4.3 em Fortran 95 na forma de função.

```
! ***** FUNCAO TRAPEZ_ROM *****
! Calcula a quadratura numérica de uma função f(x) pela regra dos trapézios
! estendida.
! Criada como parte integrante do Método de Romberg para integração
! automática.
!
! Argumentos:
! f: Função na forma f(x) a ser integrada (Entrada).
! a: Limite inferior de integração (Entrada).
! b: Limite superior de integração (Entrada).
```

```

! n_ordem: Ordem de chamada da função           (Entrada).
!
! Autor: Rudi Gaelzer, IF-UFRGS.
! Data: Maio/2008.
!
function trapez_rom(f, a, b, n_ordem)
real(kind= dp) :: trapez_rom
integer, intent(in) :: n_ordem
real(kind= dp), intent(in) :: a, b
integer :: i
integer, save :: npts
real(kind= dp) :: h, delta, x, soma
real(kind= dp), save :: I_te, fat2
INTERFACE
  function f(x)
  use Modelos_Computacionais_Dados
  real(kind= dp) :: f
  real(kind= dp), intent(in) :: x
  end function f
END INTERFACE
h= b - a
! Testa primeira rodada.
select case(n_ordem)
case(0)           ! Primeira rodada.
  I_te= 0.5*h*(f(a) + f(b))
  fat2= 1.0_dp
  npts= 1
case default ! Rodadas subsequentes.
  delta= h/fat2
  x= a + 0.5*delta
  soma= 0.0_dp
  do i= 1, npts
    soma= soma + f(x)
    x= x + delta
  end do
  I_te= 0.5*(I_te + h*soma/fat2)
  fat2= 2.0_dp*fat2
  npts= 2*npts
end select
trapez_rom= I_te
return
end function trapez_rom

```

Tendo sido estabelecido um algoritmo eficiente para o cômputo de  $I_R^{(0)}(h)$ , ...,  $I_R^{(0)}(h/2^k)$ , outro se faz agora necessário para implementar a integração de Romberg, juntamente com um controle de erro que interrompe o processamento quando o erro absoluto ou relativo fica abaixo de um limite de tolerância fornecido pelo programador.

A subrotina 4.3 apresentado a seguir implementa o cálculo da quadratura pelo método de Romberg. A abordagem adotada consiste em percorrer as diagonais do diagrama apresentado na figura 4.5 duas vezes consecutivas a cada teste no valor do erro relativo obtido. Ou seja, partindo de  $I_R^{(0)}(h)$ , calcula-se em seqüência a diagonal composta por  $I_R^{(0)}(h/2)$  e  $I_R^{(1)}(h)$ , seguido do cálculo da diagonal composta por  $I_R^{(0)}(h/2^2)$ ,  $I_R^{(1)}(h/2)$  e  $I_R^{(2)}(h)$ , o que permite o cálculo de  $EA_1(h/2)$  e do erro relativo. Se a estimativa de erro desejada foi alcançada, o resultado é  $I_R^{(2)}(h)$ ; caso contrário, as próximas duas diagonais são calculadas, sendo testados  $EA_2(h/2)$  e  $I_R^{(3)}(h)$  e assim consecutivamente.

**Listing 4.3:** Programa que calcula a quadratura numérica pelo Método de Romberg.

```

! ***** SUBROTINA QUAD_ROM *****
! Calcula a quadratura numérica de uma função f(x) usando o Método de Romberg.

```

```

!
! Argumentos:
! f: Função na forma  $f(x)$  a ser integrada (Entrada).
! a: Limite inferior de integração (Entrada).
! b: Limite superior de integração (Entrada).
! errrel: Valor máximo admitido para o erro relativo (Entrada).
! result: Valor obtido para a quadratura numérica (Saida).
! errest: Valor estimado para o erro relativo (Saida).
!
! Autor: Rudi Gaelzer, IF-UFRGS.
! Data: Maio/2008.
!
subroutine quad_rom(f, a, b, errrel, result, errest)
real(kind= dp), intent(in) :: a, b, errrel
real(kind= dp), intent(out) :: result, errest
! Variáveis locais
integer :: k, i, np
real(kind= dp) :: quatro, quatroiml, errabs
real(kind= dp), dimension(:), allocatable :: Ikml, Ik, temp1, temp2
INTERFACE
function f(x)
use Modelos_Computacionais_Dados
real(kind= dp) :: f
real(kind= dp), intent(in) :: x
end function f
END INTERFACE
!
if(b == a)then
result= 0.0_dp
errest= 0.0_dp
return
end if
np= 100
allocate(Ikml(0:np), Ik(0:np))
Ik(0)= trapez_rom(f, a, b, 0)
Ikml(0)= trapez_rom(f, a, b, 1)
Ikml(1)= (4*Ikm1(0) - Ik(0))/3.0_dp
k= 2
do
if(k > np)then !Dobrar a alocação atual dos vetores
allocate(temp1(0:2*np), temp2(0:2*np))
temp1(0:np)= Ikml
temp2(0:np)= Ik
call move_alloc(temp1,Ikml)
call move_alloc(temp2,Ik)
np= 2*np
end if
! Realiza lacos ao longo das diagonais.
Ik(0)= trapez_rom(f, a, b, k)
quatro= 1.0_dp
do i= 1, k
quatro= 4.0_dp*quatro
quatroiml= quatro - 1.0_dp
Ik(i)= (quatro*Ikm1(i-1) - Ikml(i-1))/quatroiml
end do
! Calcula e compara erro
errabs= (Ik(k-1) - Ikml(k-1))/quatroiml
errest= abs(errabs/Ik(k))
if(errest <= errrel)then

```

```

      result= Ik(k)
      exit
    else
      k= k + 1
      Ikml= Ik
    end if
  end do
  return
end subroutine quad_rom

```

Cabe aqui mencionar que tanto a função `trapez_rom` quanto a subrotina `quad_rom` necessitam de informações adicionais para que todas as interfaces e espécies de variáveis sejam explicitadas. A melhor estratégia consiste em inserir estas rotinas em um módulo, o qual pode usar outros módulos que contenham declarações globais de variáveis ou rotinas auxiliares. Outro ponto que merece destaque é que a subrotina `quad_rom` faz uso de vetores alocáveis para acumular os resultados das diagonais da figura 4.5. Embora raramente possa acontecer, é possível que o tamanho declarado para os vetores seja excedido devido às exigências na acuracidade do resultado. Para evitar a ocorrência de um erro do tipo *out-of-bounds*, a subrotina faz uso da subrotina `move_alloc`, a qual irá duplicar o espaço disponível para os vetores usados na subrotina `quad_rom`, empregando dois vetores temporários `temp1` e `temp2`, os quais são automaticamente dealocados após a execução do `move_alloc`. Esta subrotina não existe no Fortran 95, mas foi criada como rotina intrínseca no Fortran 2003.

Todas as rotinas desenvolvidas neste capítulo, em conjunto com os módulos necessários para implementá-las, podem ser obtidos em <http://minerva.ufpel.edu.br/~rudi/grad/ModComp/Progs/>.

#### 4.4.1.2 INTEGRAIS IMPRÓPRIAS DE ROMBERG

O termo *integral imprópria* normalmente se aplica quando ocorre no mínimo um dos problemas a seguir:

1. O integrando possui uma singularidade removível; isto é, existe no mínimo um ponto no qual a função não está definida, mas possui um limite finito. Por exemplo,  $f(x) = \sin x/x$  em  $x = 0$ .
2. O integrando possui uma singularidade integrável em um ponto conhecido, o qual pode ser um dos limites de integração. Por exemplo,

$$\int_0^1 \frac{dx}{x^\alpha}, \text{ para } 0 < \alpha < 1.$$

3. O limite superior de integração tende a  $\infty$  e/ou o limite inferior tende a  $-\infty$ . Neste caso, a integral somente possuirá um valor se o integrando possuir o limite  $\lim_{x \rightarrow \pm\infty} f(x) \rightarrow 0$  e o limite a seguir também existir,

$$\lim_{R \rightarrow \infty} \int_{-R}^R |f(x)| dx \neq \infty.$$

Nesta seção será discutida a implementação de um integrador automático que possibilita o cálculo de integrais impróprias que satisfazem as condições acima, juntamente com o controle de erro no cálculo numérico da integral.

Para implementar o cálculo de integrais impróprias, uma regra de quadratura aberta se faz necessária para evitar o cálculo do integrando em um ponto singular finito ou no infinito. Será então empregada a regra do ponto médio estendida (4.12), pois esta possibilita uma implementação relativamente fácil e eficiente do método de extrapolação de Richardson. De fato, esta regra possui a mesma vantagem que a regra trapezoidal estendida possui para a integral de Romberg, qual seja, o fato de que os termos de erro de truncamento na fórmula de quadratura são sempre proporcionais a potências pares de  $h$ , conforme está expresso pela *Segunda Fórmula de Euler-MacLaurin* [5, Eq. 4.4.1]:

$$E_{APME} = \frac{B_2}{4} [f'(b) - f'(a)] h^2 + \dots + \frac{B_{2k}}{(2k)!} (1 - 2^{-2k+1}) [f^{(2k-1)}(b) - f^{(2k-1)}(a)] h^{2k} + \dots, \quad (4.31)$$

sendo  $EA_{PME}$  o erro absoluto resultando do uso desta regra.

Por outro lado, diferente do que ocorre com a regra trapezoidal estendida, para a qual os pontos utilizados no cálculo anterior podem ser aproveitados posteriormente, bastando para isso dobrar o número de sub-intervalos a cada iteração, na regra do ponto médio estendida isto somente ocorrerá se o número de sub-intervalos for *triplicado* entre dois cálculos subsequentes. Por exemplo, se a primeira aproximação para o valor de  $I_{PME}$ , isto é, o valor da quadratura usando a regra, for obtida utilizando somente 1 ponto, em  $x_0 = a + h/2$ , sendo  $h = b - a$  (o que corresponde a tomar-se  $n = 0$  em 4.12), este somente será utilizado na próxima iteração para  $n = 4$  (onde serão utilizados os 3 pontos em  $x_0 = a + h/6$ ,  $x_1 = a + h/2$  e  $x_2 = a + 5h/6$ ). Estes 3 pontos somente serão utilizados novamente se a próxima iteração ocorrer para  $n = 16$ , para a qual são utilizados os 9 pontos em  $x_0 = a + h/18$ ,  $x_1 = a + h/6$ ,  $x_2 = a + 5h/18$ ,  $x_3 = a + 7h/18$ ,  $x_4 = a + h/2$ ,  $x_5 = a + 11h/18$ ,  $x_6 = a + 13h/18$ ,  $x_7 = a + 5h/6$  e  $x_8 = a + 17h/18$ . Nesta última iteração, torna-se necessário calcular 6 pontos adicionais. De uma forma geral, para a  $i$ -ésima iteração ( $i \geq 2$ ), torna-se necessário calcular  $(2/3) \times 3^{i-1}$  pontos adicionais, sendo a relação entre o número total de pontos ( $n_{pts}$ ) para um dado valor de  $n$  dada por  $n = 2(n_{pts} - 1)$ , para  $n_{pts} = 1, 3, 9, 27, \dots, 3^i, \dots$ .

Quanto ao valor de  $I_{PME}$ , se para  $n = 0$  obtém-se (de acordo com 4.12)

$$I_{PME}^{(0)} = hf_0, \text{ sendo } f_0 = f\left(a + \frac{h}{2}\right),$$

este valor pode ser aproveitado no cálculo para  $n = 4$  da seguinte maneira,

$$I_{PME}^{(4)} = \frac{h}{3}(f_0 + f_1 + f_2), \text{ sendo } x_0 = a + \frac{1}{6}h, x_1 = a + \frac{1}{2}h \text{ e } x_2 = a + \frac{5}{6}h.$$

Então

$$I_{PME}^{(4)} = \frac{1}{3} \left[ h(f_0 + f_2) + I_{PME}^{(0)} \right].$$

Por sua vez, para  $n = 16$  resulta

$$I_{PME}^{(16)} = \frac{1}{3} \left[ \frac{h}{3}(f_0 + f_2 + f_3 + f_5 + f_6 + f_8) + I_{PME}^{(4)} \right].$$

Observa-se aqui um claro padrão que pode ser utilizado para a elaboração de um algoritmo.

O algoritmo 4.4 sugere uma implementação para o cálculo de  $I_{PME}$  fazendo uso do resultado da iteração anterior. Este algoritmo baseia-se na subrotina `midpnt` apresentada por Press *et al.* (1992) [5, seção 4.4], mas é bastante semelhante ao algoritmo 4.3. Este algoritmo está implementado na forma de uma função em Fortran 95 no programa 4.4. Esta função faz uso do construto `forall` para torná-la apta à paralelização do código. Para tanto, é necessário que a função a ser integrada seja pura.

Torna-se necessária agora uma subrotina, nos moldes da `quad_rom` (programa 4.3), para acionar a função `pntmed_rom` e realizar a integração imprópria utilizando o método da extrapolação de Richardson. O valor extrapolado e a estimativa de erro serão novamente dadas por (4.28a,b). Agora, porém, a cada ordem de refinamento o sub-intervalo anterior será dividido pelo fator  $R = 3$ . Como o erro absoluto na regra do ponto médio estendida é dado por uma série de potências pares de  $h$ , da mesma forma que para a regra trapezoidal estendida, pode-se usar a implementação desta última como guia para a integral de Romberg de integrais impróprias.

Portanto, chamando  $I_R^{(0)}(h) \equiv I_{PME}(h)$  o valor da quadratura na ordem mais baixa da regra do ponto médio, os valores extrapolados e as estimativas de erro nas ordens subsequentes serão dados por:

$$\begin{aligned} I_R^{(1)}(h) &= \frac{1}{3^2 - 1} \left[ 3^2 I_R^{(0)}\left(\frac{h}{3}\right) - I_R^{(0)}(h) \right], & EA_0\left(\frac{h}{3}\right) &= \frac{1}{3^2 - 1} \left[ I_R^{(0)}\left(\frac{h}{3}\right) - I_R^{(0)}(h) \right] \\ I_R^{(2)}(h) &= \frac{1}{3^4 - 1} \left[ 3^4 I_R^{(1)}\left(\frac{h}{3}\right) - I_R^{(1)}(h) \right], & EA_1\left(\frac{h}{3}\right) &= \frac{1}{3^4 - 1} \left[ I_R^{(1)}\left(\frac{h}{3}\right) - I_R^{(1)}(h) \right] \\ I_R^{(3)}(h) &= \frac{1}{3^6 - 1} \left[ 3^6 I_R^{(2)}\left(\frac{h}{3}\right) - I_R^{(2)}(h) \right], & EA_2\left(\frac{h}{3}\right) &= \frac{1}{3^6 - 1} \left[ I_R^{(2)}\left(\frac{h}{3}\right) - I_R^{(2)}(h) \right] \\ &\vdots & &\vdots \end{aligned}$$

Listing 4.4: Implementação do algoritmo 4.4 em Fortran 95 como uma função.

```

! ***** FUNÇÃO PNTMED_ROM *****
! Calcula a quadratura numérica de uma função  $f(x)$  pela regra dos pontos
! médios estendida.
! Criada como parte integrante do Metodo de Romberg para integração
! automática.
!
! Argumentos:
! f:      Função na forma  $f(x)$  a ser integrada (Entrada).
! a:      Limite inferior de integração      (Entrada).
! b:      Limite superior de integração      (Entrada).
! n_ordem: Ordem de chamada da função      (Entrada).
!
! Autor: Rudi Gaelzer, IF-UFRGS
! Data: Junho/2010.
!
function pntmed_rom(f, a, b, n_ordem)
real(kind= dp) :: pntmed_rom
integer, intent(in) :: n_ordem
real(kind= dp), intent(in) :: a, b
integer :: i, npts
real(kind= dp) :: h, del
real(kind= dp), save :: I_pme
real(kind= dp), dimension(2*(3**(n_ordem-1))) :: x, f_eval
INTERFACE
  pure function f(x)
  use Modelos_Computacionais_Dados
  real(kind= dp) :: f
  real(kind= dp), intent(in) :: x
  end function f
END INTERFACE
h= b - a
select case(n_ordem) ! Testa primeira rodada.
case(0)              ! Primeira rodada.
  I_pme= h*f(0.5_dp*(a+b))
case default ! Rodadas subsequentes.
  npts= 3**(n_ordem - 1)
  del= h/(6.0_dp*npts)
  forall(i= 1:npts)
    x(2*i-1)= a + (6*i-5)*del
    x(2*i)=   a + (6*i-1)*del
  end forall
  forall(i= 1:2*npts)
    f_eval(i)= f(x(i))
  end forall
I_pme= 2.0_dp*del*sum(f_eval) + I_pme/3.0_dp
end select
pntmed_rom= I_pme
! PRINT*, 'N_ORDEM=', N_ORDEM
! DO I= 1, 2*NPTS
!   PRINT*, 'I=', I, X(I), F_EVAL(I)
! END DO
! PRINT*, 'I=', PNTMED_ROM
return
end function pntmed_rom

```

Pode-se induzir então o  $k$ -ésimo ( $k \geq 1$ ) valor extrapolado e a estimativa de erro como

$$\begin{aligned} I_R^{(k)}(h) &= \frac{1}{9^k - 1} \left[ 9^k I_R^{(k-1)}\left(\frac{h}{3}\right) - I_R^{(k-1)}(h) \right], \\ EA_{k-1}\left(\frac{h}{3}\right) &= \frac{1}{9^k - 1} \left[ I_R^{(k-1)}\left(\frac{h}{3}\right) - I_R^{(k-1)}(h) \right]. \end{aligned}$$

A dependência de cada quadratura extrapolada nos valores da ordem anterior pode ser visualizada por um diagrama semelhante ao apresentado na figura 4.5, bastando para tanto substituir  $R = 2$  por  $R = 3$ . Devido às semelhanças entre os dois métodos, a implementação do cálculo da integral de Romberg pela regra do ponto médio será realizada por uma rotina baseada no programa 4.3. Esta implementação pode ser vista no programa 4.5. Esta subrotina pode servir como uma alternativa ao programa 4.3 para o cálculo de uma integral definida, usando, porém, o método do ponto médio estendido.

**Listing 4.5:** Subrotina para o cálculo da integral de Romberg usando a regra do ponto médio estendida.

```
! ***** SUBROTINA QPMS_ROM *****
! Calcula a quadratura numerica de uma funcao f(x) usando o Metodo de Romberg.
! A funcao pode conter singularidades integraveis nos limites de integracao.
!
! Argumentos:
!   f: Funcao na forma f(x) a ser integrada           (Entrada).
!   a: Limite inferior de integracao                   (Entrada).
!   b: Limite superior de integracao                   (Entrada).
!   errrel: Valor maximo admitido para o erro relativo (Entrada).
!   result: Valor obtido para a quadratura numerica    (Saida).
!   errest: Valor estimado para o erro relativo        (Saida).
!
! Rotinas usadas: pntmed_rom.
!
! Autor: Rudi Gaelzer, IFM - UFPel.
! Data: Junho/2010.
!
subroutine qpms_rom(f, a, b, errrel, result, errest)
```

**Algoritmo 4.4** Calcula o  $n$ -ésimo refinamento da regra do ponto médio estendida (4.12), sendo dados  $f(x)$ , os limites de integração  $(a, b)$  e o resultado da quadratura no estágio anterior ( $I_{PME}$ ). Os pontos incluídos em cada estágio são sempre distintos de todos os outros pontos anteriores. Quando chamado com  $n = 1$ , o algoritmo calcula a quadratura usando  $h = (b - a)/2$ ; quando chamado com  $n = 2, 3, \dots$ , o resultado será refinado pela adição de  $(2/3) \times 3^{n-1}$  pontos interiores adicionais.

**Dados:**  $f(x)$ ,  $a$ ,  $b$ ,  $n$  e  $I_{PME}$ :

**Se  $n = 1$  então:**

$$I_{PME} = (b - a) f\left(\frac{a+b}{2}\right)$$

**senão:**

$$npts = 3^{n-2}$$

$$\delta = (b - a) / (3npts)$$

$$\delta2 = 2\delta$$

$$x = a + \delta/2$$

$$soma = 0$$

**Para  $j = 1 : npts$ , faça:**

$$soma = soma + f(x)$$

$$x = x + \delta2$$

$$soma = soma + f(x)$$

$$x = x + \delta$$

**final laço**

$$I_{PME} = (1/3) [I_{PME} + (b - a) soma / npts]$$

**final teste**

```

real(kind= dp), intent(in) :: a, b, errrel
real(kind= dp), intent(out) :: result, errest
! Variaveis locais
integer :: k, i, np
real(kind= dp) :: novei, noveim1, errabs
real(kind= dp), dimension(:), allocatable :: lkml, lk, temp1, temp2
INTERFACE
  pure function f(x)
  use Modelos_Computacionais_Dados
  real(kind= dp) :: f
  real(kind= dp), intent(in) :: x
  end function f
END INTERFACE
!
  if(b == a)then
    result= 0.0_dp
    errest= 0.0_dp
    return
  end if
  np= 200
  allocate(lkml(0:np), lk(0:np))
  lk(0)= pntmed_rom(f, a, b, 0)
  lkml(0)= pntmed_rom(f, a, b, 1)
  lkml(1)= (9.0_dp*lkml(0) - lk(0))/8.0_dp
  k= 2
  do
    if(k > np)then !Dobrar a alocação atual dos vetores
      allocate(temp1(0:2*np), temp2(0:2*np))
      temp1(0:np)= lkml
      temp2(0:np)= lk
      call move_alloc(temp1, lkml)
      call move_alloc(temp2, lk)
      np= 2*np
    end if
    ! Realiza lacos ao longo das diagonais.
    lk(0)= pntmed_rom(f, a, b, k)
    novei= 1.0_dp
    do i= 1, k
      novei= 9.0_dp*novei
      noveim1= novei - 1.0_dp
      lk(i)= (novei*lk(i-1) - lkml(i-1))/noveim1
    end do
    ! Calcula e compara erro
    errabs= (lk(k-1) - lkml(k-1))/noveim1
    !   errest= abs(errabs/lk(k))
    !   ERREST= ABS(ERRABS/IK(K-1))
    !   if(errest <= errrel)then
    IF(ABS(ERRABS) <= ERRREL*ABS(IK(K-1)))THEN
      result= lk(k)
      exit
    else
      k= k + 1
      lkml= lk
    end if
  end do
  return
end subroutine qpms_rom

```

Contudo, como há mais de uma forma para uma integral imprópria, as implementações irão



diferenciar-se ligeiramente umas das outras, com graus de dificuldades distintos. A discussão abaixo mostra qual é o tipo de integral imprópria que pode ser facilmente implementado com o método abordado nesta seção.

### TIPO 1. INTEGRANDO COM SINGULARIDADES INTEGRÁVEIS E LIMITES DE INTEGRAÇÃO FINITOS

Neste caso, será suposto que os limites de integração sejam escolhidos de tal forma que as singularidades se encontrem exatamente nos limites de integração. Isto pode ser alcançado mediante uma divisão apropriada dos limites originais de integração. A integral abaixo ilustra o tipo de integral imprópria considerado aqui.

$$\int_{-1}^1 \frac{dx}{\sqrt{1-x^2}} = \pi.$$

O método não irá funcionar quando a singularidade não for diretamente integrável, como ocorre no cálculo do valor principal de Cauchy da integral:

$$\int_1^2 \frac{dx}{1-x} = \lim_{\epsilon \rightarrow 0^+} \left[ \int_0^{1-\epsilon} \frac{dx}{1-x} + \int_{1+\epsilon}^2 \frac{dx}{1-x} \right] = 0.$$

Neste último caso, uma implementação do cálculo da parte principal é necessária.

Mesmo no caso de uma integral do primeiro tipo acima, o método abordado nesta seção em geral não irá fornecer um resultado acurado. Como o integrando pode variar muito rapidamente em uma vizinhança estreita dos limites de integração e lentamente nas vizinhanças do ponto médio, normalmente a contribuição das primeiras vizinhanças para a integral, em comparação com a contribuição em torno do ponto médio, pode ser grande e o método fornece um resultado totalmente incorreto com uma estimativa de erro muito menor que o valor correto. Matematicamente, pode-se entender a razão deste comportamento pela fórmula de erro do método do ponto médio (4.31). Se  $f(x)$  possui uma singularidade em um ou ambos os pontos extremos, então neste limite,  $|f'(x)| \rightarrow \infty$ , assim como as suas derivadas de ordem mais alta, resultando em um erro absoluto gigantesco para a fórmula do ponto médio.

A melhor maneira de resolver este tipo de integral imprópria consiste em utilizar uma fórmula gaussiana (ver seção 4.4.2 abaixo), a qual não padece desta limitação das fórmulas newtonianas. Pacotes profissionais de quadratura numérica usam uma estratégia adicional que consiste em sub-dividir o intervalo de integração e analisar a convergência do método em cada sub-intervalo separadamente. Uma rotina que utiliza este tipo de estratégia é denominada **adaptativa**. Métodos adaptativos são rapidamente abordados na seção 4.4.3.

### TIPO 2. INTEGRANDO SEM SINGULARIDADES E INTERVALOS INFINITOS DE INTEGRAÇÃO

As integrais consideradas aqui são dos seguintes tipos:

$$\int_{-\infty}^{\infty} f(x) dx \text{ ou } \int_{-\infty}^a f(x) dx \text{ ou } \int_a^{\infty} f(x) dx, \quad (a = \text{cte.})$$

onde está suposto que  $f(x)$  não possui pontos singulares no intervalo de integração e que  $|f(x)| \xrightarrow{|x| \rightarrow \infty} 0$  de forma rápida o suficiente para que a integral exista. Nesta situação, normalmente ocorre também que  $|f^{(n)}(x)| \xrightarrow{|x| \rightarrow \infty} 0$ , ou seja, as derivadas em todas as ordens de  $f(x)$  também vão a zero neste limite. Neste caso, o erro da fórmula de ponto médio (4.31) torna-se pequeno e pode-se esperar resultados bastante acurados.

Há mais de uma maneira de implementar o algoritmo 4.4 para uma integral imprópria deste tipo, dependendo da forma predominante de  $f(x)$  para  $|x| \rightarrow \infty$ . Para simplificar a análise, será considerado somente um caso particular para uma integral imprópria deste tipo:

- $a > 0$ .
- $f(x) \rightarrow 0$  mais rapidamente que  $x^2$ , ou seja, para  $x \rightarrow \infty$ ,  $x^2 |f(x)| < 1/x^\alpha$ , sendo  $\alpha > 1$ .

Caso estas condições sejam satisfeitas, pode-se realizar uma transformação de variáveis do tipo  $x \rightarrow 1/u$ , a qual irá transformar o intervalo infinito em um intervalo finito, no qual pode-se usar o algoritmo 4.4. A integral a ser calculada neste caso, portanto, é obtida a partir da mudança de variável de integração  $u = 1/(1+x)$ , ou seja,

$$I = \int_a^\infty f(x) dx = \int_0^{1/(1+a)} \frac{f(1/u - 1)}{u^2} du.$$

O programa 4.6 apresenta a função `pntmed_int_rom`, a qual consiste em uma variação de `pntmed_rom` para o intervalo de integração considerado. Já a subrotina `qpmi_rom`, apresentada na listagem de programa 4.7 implementa a integral de Romberg para uma integral imprópria do tipo considerado.

**Listing 4.6:** Função que implementa a regra estendida dos pontos médios para um intervalo infinito.

```
! ***** FUNCAO PNTMED_INF_ROM *****
! Calcula a quadratura numerica de uma funcao f(x) pela regra dos pontos
! medios estendida.
! Criada como parte integrante do Metodo de Romberg para a integracao
! automatica de uma funcao no intervalo [a, Infinito).
!
! Argumentos:
! f: Funcao na forma f(x) a ser integrada (Entrada).
! a: Limite inferior de integracao (Entrada).
! n_ordem: Ordem de chamada da funcao (Entrada).
!
! Autor: Rudi Gaelzer, IFM – UFPel.
! Data: Junho/2010.
!
function pntmed_inf_rom(f, a, n_ordem)
real(kind= dp) :: pntmed_inf_rom
integer, intent(in) :: n_ordem
real(kind= dp), intent(in) :: a
integer :: i, npts
real(kind= dp) :: h, del
real(kind= dp), save :: I_pme
real(kind= dp), dimension(2*(3**(n_ordem-1))) :: x, f_eval
INTERFACE
  pure function f(x)
  use Modelos_Computacionais_Dados
  real(kind= dp) :: f
  real(kind= dp), intent(in) :: x
  end function f
END INTERFACE
h= 1.0_dp/(1.0_dp + a)
select case(n_ordem) ! Testa primeira rodada.
case(0) ! Primeira rodada.
  I_pme= h*func(0.5_dp*h)
case default ! Rodadas subsequentes.
  npts= 3**(n_ordem - 1)
  del= h/(6.0_dp*npts)
  forall(i= 1:npts)
    x(2*i-1)= (6*i-5)*del
    x(2*i)= (6*i-1)*del
  end forall
  forall(i= 1:2*npts)
    f_eval(i)= func(x(i))
  end forall
I_pme= 2.0_dp*del*sum(f_eval) + I_pme/3.0_dp
end select
pntmed_inf_rom= I_pme
```

```

return
CONTAINS
  pure function func(x)
  real(dp) :: func
  real(dp), intent(in) :: x
  func= f(1.0_dp/x - 1.0_dp)/(x*x)
  return
  end function func
end function pntmed_inf_rom

```

**Listing 4.7:** Subrotina para o cálculo da integral de Romberg de uma integral imprópria.

```

! ***** SUBROTINA QPMI_ROM *****
! Calcula a quadratura numerica de uma funcao f(x) usando o Metodo de Romberg
! no intervalo de integracao [a, Infinito).
! O integrando nao possui pontos singulares no intervalo de integracao.
!
! Argumentos:
! f: Funcao na forma f(x) a ser integrada (Entrada).
! a: Limite inferior de integracao (Entrada).
! errrel: Valor maximo admitido para o erro relativo (Entrada).
! result: Valor obtido para a quadratura numerica (Saida).
! errest: Valor estimado para o erro relativo (Saida).
!
! Rotinas usadas: pntmed_inf_rom.
!
! Autor: Rudi Gaelzer, IFM – UFPel.
! Data: Junho/2010.
!
  subroutine qpml_rom(f, a, errrel, result, errest)
  real(kind= dp), intent(in) :: a, errrel
  real(kind= dp), intent(out) :: result, errest
! Variaveis locais
  integer :: k, i, np
  real(kind= dp) :: novei, noveim1, errabs
  real(kind= dp), dimension(:), allocatable :: Ikml, Ik, temp1, temp2
  INTERFACE
    pure function f(x)
    use Modelos_Computacionais_Dados
    real(kind= dp) :: f
    real(kind= dp), intent(in) :: x
    end function f
  END INTERFACE
!
  np= 200
  allocate(Ikml(0:np), Ik(0:np))
  Ik(0)= pntmed_inf_rom(f, a, 0)
  Ikml(0)= pntmed_inf_rom(f, a, 1)
  Ikml(1)= (9.0_dp*Ikm1(0) - Ik(0))/8.0_dp
  k= 2
  do
    if(k > np) then !Dobrar a alocação atual dos vetores
      allocate(temp1(0:2*np), temp2(0:2*np))
      temp1(0:np)= Ikml
      temp2(0:np)= Ik
      call move_alloc(temp1, Ikml)
      call move_alloc(temp2, Ik)
      np= 2*np
    end if
! Realiza lacos ao longo das diagonais.

```

```

Ik(0)= pntmed_inf_rom(f, a, k)
novei= 1.0_dp
do i= 1, k
    novei= 9.0_dp*novei
    noveiml= novei - 1.0_dp
    Ik(i)= (novei*Ik(i-1) - Ikml(i-1))/noveiml
end do
! Calcula e compara erro
errabs= (Ik(k-1) - Ikml(k-1))/noveiml
errest= abs(errabs/Ik(k))
if(errest <= errrel)then
    result= Ik(k)
    exit
else
    k= k + 1
    Ikml= Ik
end if
end do
return
end subroutine qpml_rom

```

### 4.4.2 INTEGRAÇÃO AUTOMÁTICA USANDO QUADRATURAS GAUSSI- ANAS

A implementação de uma integração automática utilizando quadraturas gaussianas não é tão simples quanto com as regras newtonianas, principalmente porque para as fórmulas da seção 4.3, diferentes valores de  $N$  resultam em distintos valores das abcissas  $\{x_i\}$ , o que não permite o uso de cálculos prévios da quadratura, como acontece com o método de Romberg. Isto implica em um maior tempo de computação para o cálculo da quadratura.

Para tentar remediar este problema, diferentes técnicas de extensão da quadratura gaussiana foram elaboradas, baseadas na definição de *nodos pré-definidos*, ou seja, um conjunto de fixo de valores de abcissas que são sempre utilizados para distintos valores de  $N$ . O problema envolve então a escolha adequada de pesos e dos pontos  $\{x_i\}$  restantes que maximizam a exatidão do resultado no menor tempo de computação possível.

Um destes métodos denomina-se *Quadratura de Gauss-Radau*, onde um dos nodos fixos é um dos limites da integração. Outro método é a *Quadratura de Gauss-Lobatto*, onde ambos os extremos  $a$  e  $b$  são nodos fixos. Uma outra classe de métodos importantes são as fórmulas de *Gauss-Kronrod*, onde todas as abcissas utilizadas em um cálculo prévio da quadratura são aproveitadas para valores subsequentes de  $N$ . Se o cálculo inicial utilizou  $N = m$  pontos, então o próximo cálculo utilizará  $N = 2n + m$  pontos: os  $n$  novos pesos e abcissas mais os  $m$  pesos e abcissas anteriores. Kronrod mostrou que se  $n$  e  $m$  são escolhidos tais que  $n = m + 1$ , uma fórmula de quadratura automática pode ser estabelecida para a regra de Gauss-Legendre. Neste caso, a sequência de pontos novo utilizados é  $N = 10, 21, 43, 87, \dots$ . Bibliotecas de software numérico, tanto comerciais quanto gratuitas, sempre disponibilizam rotinas do tipo Gauss-Kronrod.

### 4.4.3 INTEGRAÇÃO ADAPTATIVA

Retornando à fórmula geral para quadratura numérica (4.1),

$$\int_a^b f(x) dx = \sum_{i=0}^N w_i f(x_i),$$

todos os métodos apresentados neste capítulo restringem-se a uma única regra utilizada para a determinação das abcissas  $\{x_i\}$  e os pesos  $\{w_i\}$ . Um *algoritmo de quadratura adaptativa*, por outro lado, escolhe os valores de  $\{w_i\}$  e  $\{x_i\}$  dinamicamente durante a computação, de forma a adaptar-se ao comportamento particular de  $f(x)$ .

Quando o integrando apresenta um comportamento que dificulta o cálculo da sua quadratura ( $f(x)$  oscila rapidamente, por exemplo), a regra em uso, aplicada a todo intervalo de integração, pode encontrar dificuldades para obter um resultado com a precisão solicitada. Nesta situação torna-se necessário o uso de uma rotina adaptativa. Contudo, bons algoritmos de quadratura adaptativa são bastante complexos do ponto de vista de cálculo numérico e custosos para ser desenvolvidos. Nesta situação, é recomendável que o programador busque uma rotina pronta em alguma biblioteca de software numérico, ao invés de tentar desenvolvê-la por si mesmo.

No método de Romberg, o valor de  $h$  é reduzido pela metade a cada iteração consecutiva do método, até que a precisão solicitada seja alcançada. Por se basear em uma regra Newtoniana, o método exige que o valor de  $h$  seja o mesmo ao longo de todo o intervalo de integração. Contudo, o comportamento de  $f(x)$  pode não requerer que  $h$  seja o mesmo ao longo de todo o intervalo para que a precisão solicitada seja atingida. Em regiões onde o integrando varia lentamente apenas uns poucos pontos podem ser suficientes; ao passo que nas regiões onde  $f(x)$  varia rapidamente, um número maior de pontos é requerido.

Considera-se, por exemplo, o integrando ilustrado na figura 4.7. Na região d-e,  $f(x)$  é essencialmente constante, e o incremento  $h$  pode ser grande. Contudo, na região a-d,  $f(x)$  varia rapidamente, significando que o incremento  $h$  deve ser pequeno. De fato, a região a-d pode ser dividida em três regiões distintas, como ilustrado. Inspeção visual do comportamento de  $f(x)$  pode identificar as regiões onde  $h$  deve ser pequeno ou grande. Porém, construir o gráfico de  $f(x)$  é um processo custoso e demorado, principalmente quando é necessário o cálculo de um número grande de integrais. Uma rotina adaptativa deve ser capaz de identificar as regiões onde  $h$  necessita ser maior ou menor e automaticamente dividir o intervalo de integração de acordo com essa identificação. Boas rotinas automáticas variam não somente os valores de  $h$  e dos pesos, mas tentam também diferentes regras de quadratura, sempre visando a otimização no cálculo numérico. Em determinados intervalos, uma rotina automática do tipo Romberg pode atingir a precisão solicitada rapidamente, enquanto que em outro intervalo o algoritmo pode optar por uma regra de quadratura gaussiana, por exemplo.

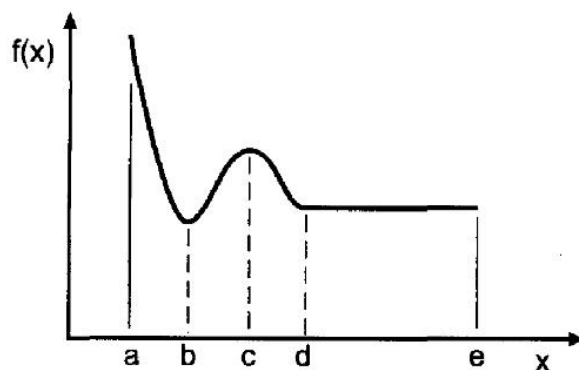


Figura 4.7: Função  $f(x)$  que apresenta distintos comportamentos ao longo do intervalo de integração.



# SOLUÇÕES DE EQUAÇÕES NÃO LINEARES

## 5.1 INTRODUÇÃO

Um problema que surge com muita frequência em computação científica consiste no cálculo das raízes de uma equação na forma

$$f(x) = 0. \quad (5.1)$$

Ou seja, é necessário calcular o conjunto de valores de  $\{x\}$  onde  $f(x)$  é nula. Em muitas situações, a função  $f(x)$  pode ser conhecida explicitamente, como é o caso de um polinômio ou de uma função transcendental. Às vezes, contudo,  $f(x)$  pode vir a ser conhecida somente de forma implícita, como ocorre quando  $f(x)$  é solução de uma equação diferencial ou integral.

Em raras circunstâncias é possível calcular-se analiticamente as raízes de  $f(x)$ . Situações onde isso ocorre restringem-se, por exemplo, a equações polinomiais do 1º ao 4º grau ou de um polinômio qualquer fatorável. Porém em geral somente soluções aproximadas para as raízes são possíveis, dependendo-se de alguma técnica computacional para calcular a aproximação. Dependendo do contexto, *solução aproximada* pode significar um ponto  $x^*$  para o qual (5.1) é *aproximadamente satisfeita*, isto é, para o qual  $|f(x^*)|$  é pequeno, ou um ponto  $x^*$  que está *próximo de* uma solução de (5.1). Infelizmente o conceito de *solução aproximada* é um tanto vago. Uma solução aproximada obtida por um computador conterá sempre um erro devido ao arredondamento, ou devido a uma instabilidade numérica ou devido ao truncamento gerado pelo método empregado. De fato, há sempre infinitas *soluções aproximadas*, todas igualmente válidas, embora a solução de (5.1) possa ser única.

Uma situação onde o efeito dos erros de arredondamento produzem falsas raízes pode ser vista na figura 1.4 à esquerda. Esta figura mostra o gráfico do polinômio  $p_6(x) = (x-1)^6$ , escrito na forma expandida, para valores próximos a  $x = 1$ . O gráfico foi gerado a partir de um programa de computador e, embora as 6 raízes de  $p_6(x)$  são únicas e iguais a 1, o gráfico mostra um número grande de pontos onde a curva cruza o eixo das abscissas. Estas falsas raízes foram produzidas pelos erros de arredondamento resultantes principalmente do cancelamento de quantidades próximas entre si. Este exemplo isolado já mostra algumas das dificuldades envolvidas no cálculo de raízes.

## 5.2 MÉTODOS ITERATIVOS PARA O CÁLCULO DE RAÍZES REAIS

Nesta seção serão apresentados os métodos iterativos elementares utilizados com maior frequência para o cálculo das **raízes reais** da função

$$f(x) = 0,$$

isto é, para uma função transcendental unidimensional. Estes métodos serão sempre exemplificados com o cálculo da raiz real da equação de 3º grau

$$p_3(x) = x^3 - x - 1 = 0. \quad (5.2a)$$

As raízes de  $p_3(x)$  podem ser obtidas analiticamente; elas consistem em uma raiz real e duas complexas conjugadas:

$$x_1 = \frac{1}{3} \left( \frac{27}{2} - \frac{3\sqrt{69}}{2} \right)^{1/3} + \frac{\left( \frac{1}{2} (9 + \sqrt{69}) \right)^{1/3}}{3^{2/3}} \approx 1.3247179572447460260 \quad (5.2b)$$

$$x_2 = -\frac{1}{6} (1 + i\sqrt{3}) \left( \frac{27}{2} - \frac{3\sqrt{69}}{2} \right)^{1/3} - \frac{(1 - i\sqrt{3}) \left( \frac{1}{2} (9 + \sqrt{69}) \right)^{1/3}}{23^{2/3}} \\ \approx -0.66235897862237301298 + 0.56227951206230124390i \quad (5.2c)$$

$$x_3 = -\frac{1}{6} (1 - i\sqrt{3}) \left( \frac{27}{2} - \frac{3\sqrt{69}}{2} \right)^{1/3} - \frac{(1 + i\sqrt{3}) \left( \frac{1}{2} (9 + \sqrt{69}) \right)^{1/3}}{23^{2/3}} \\ \approx -0.66235897862237301298 - 0.56227951206230124390i. \quad (5.2d)$$

Os métodos iterativos mais conhecidos serão agora apresentados e estes terão a sua capacidade de calcular a raiz  $x_1$  analisada.

### 5.2.1 MÉTODO DA BISECÇÃO

Boa parte dos métodos de cálculo de raízes necessita do conhecimento prévio de alguma informação a respeito da solução ou de  $f(x)$  para que possa convergir para a solução correta. Para que o método da biseção funcione, é necessário inicialmente cercar a raiz (ou raízes) entre dois valores de  $x$ . Assim, sabendo-se que  $p_3(1) = -1 < 0$  e que  $p_3(2) = 5 > 0$ , conclui-se que há um número ímpar de raízes dentro do intervalo  $[1, 2]$ . Assim, a informação inicial que é necessária fornecer ao método da biseção é um par de pontos  $x = a_0$  e  $x = b_0$  distintos tais que

$$f(a_0) f(b_0) < 0, \quad (5.3)$$

em cuja situação sempre haverá um número ímpar de raízes no intervalo  $[a_0, b_0]$ .

Se  $f(a_0) f(b_0) > 0$ , significa que há um número par de raízes no intervalo (zero inclusive), o que torna necessária a procura de um outro intervalo. Se a condição (5.3) for satisfeita, há um número ímpar de raízes em  $[a_0, b_0]$ . Contudo, se  $f(x)$  for contínua em  $[a_0, b_0]$  o seguinte teorema deve valer:

#### TEOREMA 5.1. TEOREMA DE ROLLE.

*Seja  $f(x)$  um função é contínua no intervalo finito  $[a, b]$  e diferenciável no intervalo  $(a, b)$ .*

*Se  $f(a) = f(b) = 0$ , então*

$$f'(\xi) = 0$$

*para algum  $\xi \in (a, b)$ .*

Este teorema implica em que se houver 3 ou mais raízes em  $[a_0, b_0]$ , a derivada de  $f(x)$  deve possuir uma ou mais raízes neste intervalo. Caso seja possível calcular analiticamente as raízes de  $f'(x)$ , este teorema pode ser útil. Assim,

$$p'_3(x) = 3x^2 - 1 = 0 \implies x = \pm \frac{1}{\sqrt{3}}.$$

Como não há raízes de  $p'_3(x)$  em  $[1, 2]$ , isto implica que  $p_3(x)$  possui somente uma raiz no intervalo.

O método consiste então em tomar como primeira aproximação para a raiz o valor médio:

$$\xi_1 = \frac{a_0 + b_0}{2},$$

sendo o erro absoluto igual ao valor do intervalo entre  $\xi_1$  e um dos pontos extremos:

$$EA_1 = |b_0 - \xi_1| = \frac{b_0 - a_0}{2}.$$

No caso,  $x_1 = \xi_1 \pm EA_1 = 1,5 \pm 0,5$ .



Comparando agora  $f(\xi_1)$  com os pontos extremos, necessariamente deve ocorrer

$$f(\xi_1)f(a_0) < 0 \text{ ou } f(\xi_1)f(b_0) < 0,$$

o que irá definir um novo intervalo,  $[a_0, \xi_1]$  ou  $[\xi_1, b_0]$  que contém a raiz de  $f(x)$ , reiniciando o ciclo.

No exemplo,

$$p_3(1,5)p_3(1) = -0,875 < 0 \text{ ao passo que } p_3(1,5)p_3(2) = 4,375 > 0.$$

Portanto, a raiz encontra-se no intervalo  $[1; 1,5]$ . Tomando a nova aproximação e o seu erro:

$$\begin{aligned}\xi_2 &= \frac{1+1,5}{2} = 1,25 \\ EA_2 &= 0,25.\end{aligned}$$

Verificando  $\xi_2$ :

$$p_3(1,25)p_3(1) > 0 \text{ ao passo que } p_3(1,25)p_3(1,5) < 0.$$

Portanto  $x_1 \in [1,25; 1,5]$ . A próxima iteração:

$$\begin{aligned}\xi_3 &= \frac{1,25+1,5}{2} = 1,375 \\ EA_3 &= 0,125.\end{aligned}$$

Verificando  $\xi_3$ :

$$p_3(1,25)p_3(1,375) < 0 \text{ ao passo que } p_3(1,375)p_3(1,5) > 0.$$

Portanto,

$$\begin{aligned}\xi_4 &= \frac{1,25+1,375}{2} = 1,3125 \\ EA_4 &= 0,0625.\end{aligned}$$

Verificando  $\xi_4$ :

$$p_3(1,25)p_3(1,3125) > 0 \text{ ao passo que } p_3(1,3125)p_3(1,375) < 0.$$

Portanto,  $x_1 \in [1,3125; 1,375]$ . Iterando novamente,

$$\begin{aligned}\xi_5 &= \frac{1,3125+1,375}{2} = 1,34375 \\ EA_5 &= 0,03125.\end{aligned}$$

Verificando  $\xi_5$ :

$$p_3(1,3125)p_3(1,34375) < 0 \text{ ao passo que } p_3(1,34375)p_3(1,375) > 0.$$

Portanto,  $\xi_5 \in [1,3125; 1,34375]$ .

As próximas 2 iterações produzem

$$\begin{aligned}x_1 &= 1,328125 \pm 0,015625 \\ x_1 &= 1,3203125 \pm 0,0078125.\end{aligned}$$

Portanto, pode-se observar que os resultados das iterações estão monotonicamente convergindo para a raiz  $x_1 \approx 1.3247179572447460260$ , mas após 7 iterações somente 2 casas decimais corretas foram obtidas. Esta é uma característica do método da biseção: uma vez que a raiz de uma função contínua foi cercada, ele certamente retornará o resultado correto, porém sua convergência é extremamente lenta. De uma forma mais rigorosa, como o comprimento do intervalo que sabidamente contém a raiz é dividido pelo fator 2 a cada iteração, o método da biseção produz um dígito binário correto a cada passo.

Um algoritmo que implementa o método da biseção deve iniciar com os dois valores  $a_0$  e  $b_0$  ( $b_0 > a_0$ ) para  $x$ , verificar se a raiz realmente está no intervalo fornecido e retornar os valores da raiz aproximada

$$\xi = \frac{a+b}{2} \text{ (sendo } a = a_0 \text{ e } b = b_0 \text{ na primeira iteração)}$$

**Algoritmo 5.1** Implementação do método da bisecção.

**Dados:**  $a_0, b_0$  ( $b_0 > a_0$ ),  $f(x)$ : função contínua em  $[a_0, b_0]$  e  $\text{tol}$  (tolerância máxima para o erro).

$a_n = a_0; b_n = b_0$

**Para**  $n = 0, 1, 2, \dots$ , **faça:**

$m = (a_n + b_n) / 2$

**Se**  $f(a_n) f(m) \leq 0$ :

$a_{n+1} = a_n; b_{n+1} = m$

$\text{erro} = |m - a_n| / 2$

**Senão:**

$a_{n+1} = m; b_{n+1} = b_n$

$\text{erro} = |b_n - m| / 2$

**Fim Se**

**Se**  $\text{erro} \leq \text{tol}$ : sai laço

**Fim laço.**

e o erro absoluto da aproximação

$$EA = \frac{|b - a|}{2}.$$

O erro absoluto deve então ser comparado com o valor máximo de erro tolerado, parâmetro que também deve ser fornecido ao algoritmo. Se  $EA$  é maior que a tolerância, o novo intervalo  $[a, b]$  que contém a raiz é determinado e o procedimento é repetido novamente. Se  $EA$  é menor ou igual que a tolerância, o algoritmo retorna a última aproximação para a raiz. O algoritmo 5.1 implementa este processo.

A subrotina 5.1 implementa o algoritmo 5.1 em Fortran 95. Deve-se notar que, além de implementar os passos contidos no algoritmo, a rotina controla também se realmente há pelo menos uma raiz no intervalo fornecido e também se a tolerância solicitada for exageradamente pequena, como seria o caso se fosse fornecido o valor  $\text{erro} = 10^{-20}$  ou menor para um resultado em precisão dupla, que contém somente cerca de 15 casas decimais de precisão. Este controle é realizado pela variável inteira de saída iflag.

**Listing 5.1:** Subrotina em Fortran 95 que implementa o método da bisecção.

```
! ***** SUBROTINA BISEC *****
! Busca uma raiz da função F(X) pelo Método da Bisecção.
! Argumentos:
!   F: Nome da função cuja raiz é desejada (Entrada).
!   A,B: Pontos extremos do intervalo onde a raiz é procurada (Entrada).
!   XTOL: Tolerância máxima para a aproximação da raiz (Entrada).
!   XM: Melhor resultado obtido para a raiz de F(X) (Saida).
!   IFLAG: Um inteiro: (Saida).
!           = -1, Método falhou, uma vez que F tem o mesmo sinal em A e B.
!           = 0, Encerrou, uma vez que ABS(A-B)/2 <= XTOL.
!           = 1, Encerrou, uma vez que ABS(A-B)/2 é tao pequeno que
!               novos valores para a raiz não são possíveis.
!
! Autor: Rudi Gaelzer, IF-UFRGS
! Data: Maio/2008.
!
subroutine bisec (f, a, b, xtol, xm, iflag)
real(kind= dp), intent(inout) :: a
real(kind= dp), intent(inout) :: b
real(kind= dp), intent(in)    :: xtol
real(kind= dp), intent(out)   :: xm
integer, intent(out)          :: iflag
INTERFACE
  function f(x)
  use Modelos_Computacionais_Dados
  real(kind= dp) :: f
  real(kind= dp), intent(in) :: x
```

```

    end function f
END INTERFACE
real(kind= dp) :: erro, fa, fm
!
iflag= 0
fa = f(a)
if (fa*f(b) > 0.0_dp) then
    iflag = -1
    print '( "f(x) tem o mesmo sinal nos dois pontos extremos:",2e15.7) ', a , b
    return
end if
erro = abs(b - a)
do ! Execute enquanto erro > xtol.
    erro = 0.5*erro
    if (erro <= xtol) exit
    xm = 0.5*(a + b)
    if (xm + erro == xm) then ! Teste para tolerância muito pequena.
        iflag = 1
        return
    end if
    fm = f(xm)
    if (fa*fm > 0.0_dp) then ! Determine novo intervalo.
        a = xm
        fa = fm
    else
        b = xm
    end if
end do
end subroutine bisec

```

### 5.2.2 MÉTODO DA FALSA POSIÇÃO

Uma modificação do método da biseção que permite acelerar a taxa de convergência no cálculo da raiz consiste em utilizar uma informação adicional de  $f(x)$ , qual seja, o quão próximos da raiz estão os pontos extremos do intervalo. No exemplo adotado: o cálculo da raiz real de  $p_3(x)$ , o intervalo inicialmente definido foi  $[1, 2]$ ; porém,  $p_3(1) = -1$ , ao passo que  $p_3(2) = 5$ . Isto significa que a raiz provavelmente está mais próxima de  $x = 1$  que  $x = 2$ . Portanto, ao invés de calcular  $\xi_1$  como o ponto médio entre 1 e 2, será calculada a média ponderada:

$$w_1 = \frac{p_3(2) \cdot 1 - p_3(1) \cdot 2}{p_3(2) - p_3(1)} = 1,1666\dots,$$

o qual está ligeiramente mais próximo de  $x_1$  que o ponto médio  $\xi_1 = 1,5$ .

Verificando agora em qual intervalo se encontra a raiz, descobre-se que ela está em  $[w_1, 2]$ , ao passo que  $p_3(w_1) = -0,578703704\dots$ . Repetindo o cálculo da média ponderada,

$$w_2 = \frac{p_3(2) \cdot w_1 - p_3(w_1) \cdot 2}{p_3(2) - p_3(w_1)} = 1,253112033\dots,$$

a qual é também ligeiramente mais próxima de  $x_1$  que  $\xi_2$ .

O método da falsa posição pode ser sistematizado da seguinte maneira. Partindo de um intervalo inicial que contenha pelo menos uma raiz de  $f(x)$ , a  $n + 1$ -ésima aproximação para a raiz, obtida dos valores da  $n$ -ésima aproximação,  $a_n$ ,  $f(a_n)$ ,  $b_n$  e  $f(b_n)$  é dada por:

$$w_n = \frac{f(b_n) a_n - f(a_n) b_n}{f(b_n) - f(a_n)}. \quad (5.4)$$

O ponto  $w_n$  é a raiz da reta secante que passa pelos pontos  $(a_n, f(a_n))$  e  $(b_n, f(b_n))$ . Se  $f(x)$  for côncava na raiz, ou seja, se  $f'(x) > 0$  na raiz, os pontos  $w_n$  estarão sempre à esquerda da raiz. Se  $f(x)$  for convexa ( $f'(x) < 0$  na raiz), os pontos  $w_n$  estarão sempre à direita da raiz. No caso

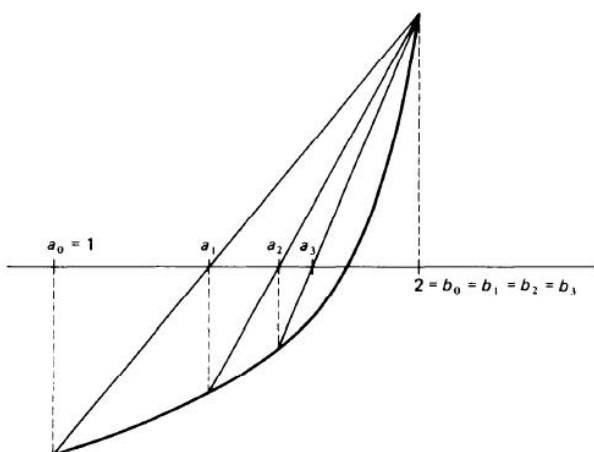


Figura 5.1: Método da falsa posição.

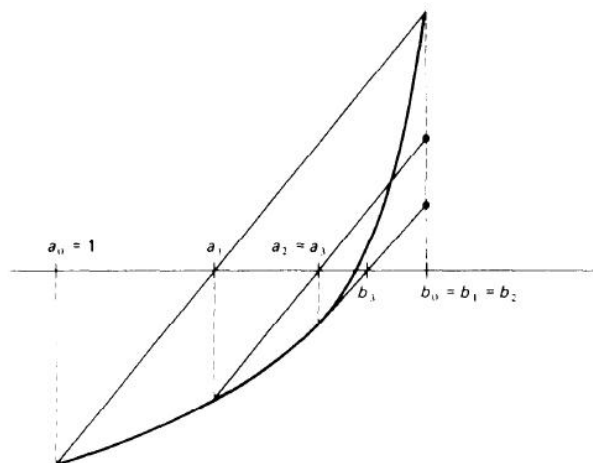


Figura 5.2: Método da falsa posição modificado.

**Algoritmo 5.2** Implementação do método da falsa posição modificado.

**Dados:**  $a_0, b_0$  ( $b_0 > a_0$ ),  $f(x)$ : função contínua em  $[a_0, b_0]$ ,  $\text{xtol}$ : tolerância máxima no tamanho de  $[a, b]$  e  $\text{ftol}$ : tolerância máxima no valor de  $|f(x)|$ .

$F = f(a_0)$ ;  $G = f(b_0)$ ;  $w_0 = a_0$

**Para**  $n = 0, 1, 2, \dots$ , **faça:**

**Se**  $|b_n - a_n| \leq \text{xtol}$  **ou**  $|w_n| \leq \text{ftol}$ : **sai laço**

$w_{n+1} = (Ga_n - Fb_n) / (G - F)$

**Se**  $f(a_n) f(w_{n+1}) \leq 0$ :

$a_{n+1} = a_n$ ;  $b_{n+1} = w_{n+1}$ ;  $G = f(w_{n+1})$

**Se**  $f(w_n) f(w_{n+1}) > 0$ :  $F = F/2$

**Senão:**

$a_{n+1} = w_{n+1}$ ;  $F = f(w_{n+1})$ ;  $b_{n+1} = b_n$

**Se**  $f(w_n) f(w_{n+1}) > 0$ :  $G = G/2$

**Fim Se**

**Fim laço**

de  $p_3(x)$ , este é côncavo em  $x = x_1$  e por consequência os valores de  $w_n$  irão se aproximar de  $x_1$  sempre pela esquerda, como se pode ver na figura 5.1.

Um aperfeiçoamento do método da falsa posição que permite acelerar a taxa de convergência à raiz é o chamado **Método da falsa posição modificado**. Neste método, as secantes são substituídas por retas de inclinações cada vez menores até que a raiz para uma determinada reta se encontre do lado oposto à aproximação  $w_n$  anteriormente obtida. Desta forma, as aproximações convergem à raiz pelo dois lados, ao invés de um lado somente, como no método da falsa posição. Este método é ilustrado na figura 5.2, o valor da ordenada em  $b_0$  é reduzido pela metade até que a raiz da reta se encontra do lado direito da raiz de  $f(x)$ .

O algoritmo 5.2 mostra como o método da falsa posição modificado pode ser implementado. Ao contrário da biseção, o presente método não pode determinar inequivocamente um valor mínimo para o intervalo onde a raiz se encontra. Na falta de uma melhor estimativa, o algoritmo toma como primeiro critério de parada um valor mínimo admissível para o intervalo que contém a raiz ( $\text{xtol}$ ). Como o valor de  $f(x)$  é continuamente reduzido à metade em um dos extremos do intervalo, um valor absurdamente pequeno para  $\text{xtol}$  pode inadvertidamente resultar em um valor numérico nulo para  $F$  ou  $G$ , devido à representação de ponto flutuante. Para evitar esta ocorrência, o algoritmo utiliza um segundo critério de parada ( $\text{ftol}$ ) que estabelece o menor valor admissível para  $|f(x)|$  em qualquer um dos pontos extremos do intervalo.

A subrotina 5.2 implementa o algoritmo 5.2. Além dos parâmetros de controle  $\text{xtol}$  e  $\text{ftol}$  já discutidos, introduz-se um parâmetro opcional  $\text{ntol}$  que controla o número total de iterações admitidas, quando presente. Este parâmetro pode ser importante quando o cálculo de  $f(x)$  é muito custoso do ponto de vista computacional. Ele pode servir também para indicar se o cálculo de  $f(x)$  está sendo feito corretamente ou se a obtenção da raiz é particularmente difícil.

**Listing 5.2:** Subrotina em Fortran 95 que implementa o método da falsa posição modificado.

```

! ***** SUBROTINA FAL_POS_MOD *****
! Busca uma raiz da função F(X) pelo Método da Falsa Posição Modificado.
! ***** Argumentos de entrada *****
! F: Nome da função cuja raiz é desejada
! A,B: Pontos extremos do intervalo onde a raiz é procurada.
! XTOL: Tolerância máxima para o intervalo que contém a raiz.
! FTOL: Tolerância máxima para o valor absoluto de F(W).
! NTOL: Número máximo de iterações admitidas (opcional).
! Se NTOL está ausente, permite infinitas iterações.
! ***** Argumentos de saída *****
! A,B: Pontos extremos do intervalo que contém a matriz.
! W: Melhor estimativa para a raiz.
! IFLAG: Um inteiro,
!      = -1, Método falhou, uma vez que F(x) tem o mesmo sinal em A e B.
!      = 0, Encerrou, porque ABS(A-B) <= XTOL.
!      = 1, Encerrou, porque ABS(F(W)) <= FTOL.
!      = 2, Encerrou, porque NTOL iterações foram realizadas.
! ***** Método *****
! O método da falsa posição modificado é empregado. Isto significa que
! a cada passo, interpolação linear entre os pontos (A,FA) e
! (B,FB) é empregada, com FA*FB < 0, para um novo ponto (W,F(W))
! que substitui um dos pontos A ou B de tal forma que novamente FA*FB < 0.
! Adicionalmente, a ordenada de um ponto que é repetido em mais de uma
! iteração é dividida por 2 a cada passo subsequente.
!
! Autor: Rudi Gaelzer, IF-UFRGS
! Data: Junho/2008.
!
subroutine fal_pos_mod(f, a, b, xtol, ftol, w, iflag, ntol)
real(kind= dp), intent(in out) :: a, b
real(kind= dp), intent(in) :: xtol, ftol
integer, intent(in), optional :: ntol
real(kind= dp), intent(out) :: w
integer, intent(out) :: iflag
INTERFACE
    function f(x)
    use Modelos_Computacionais_Dados
    real(kind= dp) :: f
    real(kind= dp), intent(in) :: x
    end function f
END INTERFACE
integer :: n
real(kind= dp) :: fa, fb, fw, signfa, prvsfw
!
fa= f(a)
signfa= sign(1.0_dp, fa)
fb= f(b)
if (signfa*fb > 0.0_dp) then
    print '("f(x) tem o mesmo sinal nos dois pontos extremos:", 2e15.7)',a,b
    iflag= -1
    return
end if
w= a
fw= fa
n= 1
do
    if (abs(a-b) <= xtol) then ! Verifica se intervalo é menor que xtol.
        iflag= 0

```

```

    return
end if
if (abs(fw) <= ftol) then      ! Verifica se ABS(f(w)) é menor que ftol.
    iflag= 1
    return
end if
w= (fa*b - fb*a)/(fa - fb)    ! Calcula novo w por interpolação.
prvsfw= sign(1.0_dp,fw)
fw= f(w)
if (signfa*fw > 0.0_dp) then ! Altera o intervalo.
    a= w
    fa= fw
    if (fw*prvsfw > 0.0_dp) fb = 0.5*fb
else
    b= w
    fb= fw
    if (fw*prvsfw > 0.0_dp) fa = 0.5*fa
end if
if((present(ntol)) .and. (n >= ntol)) then
    print '("Não houve convergência em ", i5, " iterações.")',ntol
    iflag= 2
    return
end if
n= n + 1
end do
return
end subroutine fal_pos_mod

```

Utilizando a rotina `fal_pos_mod` para o cálculo da raiz de  $p_3(x)$ , obteve-se os seguintes resultados:

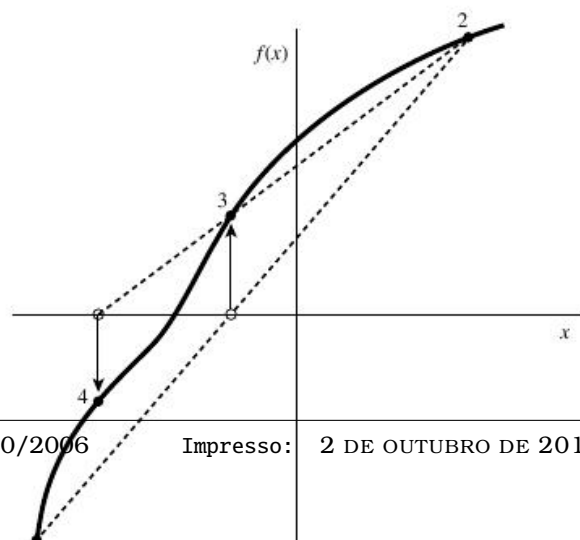
$$\begin{aligned}
 w_0 &= 1.000000000000000 \\
 w_1 &= 1.166666666666667 \\
 w_2 &= 1.32330827067669 \\
 w_3 &= 1.32654296624656 \\
 w_4 &= 1.32471556046769 \\
 w_5 &= 1.32471795317359.
 \end{aligned}$$

Ou seja, em 5 iterações, o resultado já concorda com  $x_1$  em 5 casas decimais, enquanto que com o método da biseção o resultado somente possuía 2 casas decimais corretas após o mesmo número de iterações.

### 5.2.3 MÉTODO DA SECANTE

O método da secante é um outro variante do método da falsa posição no qual, ao contrário da versão modificada, não se procura cercar a raiz entre dois pontos. Ao contrário, a fórmula (5.4) é empregada continuamente. O método da secante, por outro lado, não mais pressupõe que a raiz esteja dentro de um intervalo  $[a_0, b_0]$ . O método requer somente que sejam fornecidos dois valores iniciais para a raiz,  $x_{-1}$  e  $x_0$ , a partir dos quais novas aproximações para a raiz são obtidas a partir de

$$x_{n+1} = \frac{f(x_n)x_{n-1} - f(x_{n-1})x_n}{f(x_n) - f(x_{n-1})}, \quad (5.5)$$



para  $n = 0, 1, 2, \dots$ . Como agora  $f(x_{n-1})$  e  $f(x_n)$  não mais necessitam ter sinais opostos, a fórmula (5.5) está sujeita a erros de arredondamento quando ambos os valores forem próximos entre si. No caso mais extremo, pode até ocorrer que  $f(x_n) = f(x_{n-1})$ , em cuja situação o método falha completamente. Uma maneira de escrever (5.5) que pode mitigar a ocorrência dos erros de arredondamento é

$$x_{n+1} = x_n - f(x_n) \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}. \quad (5.6)$$

É fácil ver que (5.5) e (5.6) são idênticas. O comportamento das aproximações à raiz de  $f(x)$  no método da secante é ilustrado pela figura 5.3.

Como a raiz não mais permanece necessariamente cercada por dois valores extremos, não é possível garantir que o método da secante venha a convergir sempre. Caso o método convirja, os critérios usuais de parada são os seguintes. Para uma determinada iteração, identificada pelo índice  $n$ , o método será considerado bem sucedido se

$$|f(x_n)| \leq \text{ftol} \text{ ou } |x_n - x_{n-1}| \leq \text{xtol}. \quad (5.7a)$$

Ou seja, o valor absoluto da função ou a diferença absoluta entre duas aproximações consecutivas são considerados menores que um valor de tolerância. Quando não se conhece a ordem de grandeza do valor de  $f(x)$  em uma vizinhança em torno da raiz ou a ordem de grandeza da própria raiz, pode-se usar os seguintes valores *relativos* como critérios de parada:

$$\frac{|f(x_n)|}{\text{fmax}} \leq \text{ftol} \text{ ou } \left| \frac{x_n - x_{n-1}}{x_n} \right| \leq \text{xtol}. \quad (5.7b)$$

A subrotina 5.3 implementa o método da secante. Como critérios de parada foram escolhidos o erro absoluto no valor da função e o erro relativo entre dois resultados consecutivos. Para evitar um número excessivo de cálculos de  $f(x)$ , o parâmetro  $\text{ntol}$  é obrigatório para a rotina secante.

**Listing 5.3:** Subrotina em Fortran 95 que implementa o método da secante.

```
! ***** SUBROTINA SECANTE *****
! Busca uma raiz da função F(X) pelo Método da Secante.
! ***** Argumentos de entrada *****
! FUNC: Nome da função cuja raiz é desejada
! x1,x2: Dois valores iniciais para o início da iteração.
! ERRABS: Primeiro critério de parada. Se ABS(F(Xn)) <= FTOL,
!         então Xn é aceito como raiz
! ERRREL: Segundo critério de parada: erro relativo.
!         Se ABS(Xn - Xn-1) <= XTOL*ABS(Xn), então Xn é aceito como raiz.
! NTOL: Numero máximo de iterações admitidas.
! ***** Argumentos de saída *****
! X: Melhor estimativa para a raiz.
! IFLAG: Um inteiro,
!        = -1, Método falhou. Nenhuma raiz foi encontrada em NTOL
!          iterações. O ultimo valor encontrado para X é retornado.
!        = 0, Encerrou devido ao primeiro critério de parada.
!        = 1, Encerrou devido ao segundo critério de parada.
!
! Autor: Rudi Gaelzer, IF-UFRGS.
! Data: Junho/2008.
!
subroutine secante(func, x1, x2, errabs, errrel, ntol, x, iflag)
use Modelos_Computacionais_Extras, only: troca
integer, intent(in) :: ntol
real(kind=dp), intent(in) :: x1, x2, errabs, errrel
```

```

integer , intent(out)      :: iflag
real(kind=dp) , intent(out) :: x
INTERFACE
  function func(x)
  use Modelos_Computacionais_Dados
  real(kind=dp) , intent(in) :: x
  real(kind=dp) :: func
  end function func
END INTERFACE
integer :: j
real(kind=dp) :: dx, f, fl , x1
fl= func(x1)
f= func(x2)
if (abs(fl) < abs(f)) then ! Tome o valor inicial com o menor valor de
  x= x1                      ! func(x) como aproximação inicial.
  x1= x2
  call troca(fl , f)
else
  x1= x1
  x= x2
end if
do j= 1, ntol
  dx= (x1-x)*f/(f-fl)
  x1= x
  fl= f
  x= x + dx
  f= func(x)
  if(abs(f) <= errabs)then
    iflag= 0
    return
  end if
  if(abs(dx) <= errrel*abs(x))then
    iflag= 1
    return
  end if
end do
iflag= -1
return
end subroutine secante

```

Ao contrário dos métodos anteriores, o método da secante não exige que os dois pontos iniciais cerquem a raiz. na tabela abaixo, mostra-se um estudo das aproximações realizadas para a obtenção da raiz  $x_1$  de  $p_3(x)$ . Em todos os casos, foram tomados os seguintes valores:  $\text{errabs} = 0$ ,  $\text{errrel} = 10^{-7}$  e  $\text{ntol} = 50$ . Cada coluna corresponde a pares distintos de valores para  $x_1$  e  $x_2$ . Com exceção da primeira coluna, as demais partiram de valores que não cercam a raiz. Observa-se que o método da secante convergiu em todos os casos, embora no último a convergência tenha sido bastante lenta.



x1= 1, x2= 2	x1= 2, x2= 3	x1= -1, x2= -2	
1.0000000000000000	2.0000000000000000	-1.0000000000000000	4.75696684459274
1.1666666666666667	1.7222222222222222	-0.8333333333333333	0.722810860257192
1.39560439560440	1.46867825516563	-0.3454545454545454	0.775378136138015
1.31365666090990	1.36356158529402	6.49850961972721	2.68909924536253
1.32401611532221	1.32934949633224	-0.327662162639576	0.922191365232718
1.32472525004811	1.32488078110669	-0.309617266171994	1.04120189173659
1.32471795247273	1.32471865771829	-1.34519594172679	1.52276684371980
1.32471795724471	1.32471795735102	0.235096865426605	1.26997209752081
	1.32471795724475	2.46288916284303	1.31554258824252
		0.449496710807014	1.32520726281671
		0.662624843273948	1.32471374864206
		-21.8117328908561	1.32471795532635
		0.665601992393366	1.32471795724475
		0.668577447291536	

## 5.2.4 MÉTODO DE NEWTON-RAPHSON

Tomando novamente a fórmula do método da secante (5.6), o termo

$$\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

se aproxima de  $1/f'(x_n)$  quando a diferença entre  $x_n$  e  $x_{n-1}$  tende a zero. Portanto, é razoável que se realize esta substituição em (5.6), resultando a fórmula do **Método de Newton-Raphson**,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \quad (5.8)$$

Este resultado pode ser determinado de uma maneira um pouco mais rigorosa realizando o desenvolvimento da função  $f(x)$  em série de Taylor em torno do ponto  $a$ , supondo que  $|x - a| \ll 1$ :

$$f(x) \approx f(a) + f'(a)(x - a).$$

Se o ponto  $x$  é a raiz de  $f(x)$ , resulta

$$x = a - \frac{f(a)}{f'(a)}.$$

Escrevendo  $a$  como o valor prévio da iteração ( $a = x_n$ ) e  $x$  como o valor seguinte ( $x = x_{n+1}$ ), obtém-se a fórmula (5.8). De fato, esta fórmula fornece simplesmente o ponto onde a reta tangente a  $f(x_n)$  é nula.

O método de Newton-Raphson é um dos métodos mais utilizados para o cálculo de raízes porque fornece uma convergência rápida quando as formas analíticas de  $f(x)$  e  $f'(x)$  são conhecidas e não são custosas do ponto de vista computacional. Além disso, o método de Newton utiliza somente um ponto anterior, ao contrário do método da secante que necessita de dois pontos. Quando o cálculo de  $f'(x)$  se torna proibitivo, costuma-se utilizar o método da secante em seu lugar. Cabe aqui ressaltar que a fórmula (5.8) pode ser escrita na forma genérica

$$x_{n+1} = g(x_n), \quad n = 0, 1, 2, \dots, \quad (5.9)$$

onde  $g(x_n)$  é uma determinada função de  $x_n$ . Fórmulas do tipo (5.9) são conhecidas como **fórmulas de ponto fixo**, porque quando a seqüência  $x_0, x_1, x_2, \dots$  converge a um determinado ponto  $\xi$ , resulta que

$$\lim_{n \rightarrow \infty} x_n = \lim_{n \rightarrow \infty} x_{n+1} = \xi,$$

em cuja situação (5.9) se torna  $\xi = g(\xi)$ ; ou seja,  $\xi$  é um ponto fixo de  $g(x)$ .

O método de Newton-Raphson nem sempre irá convergir. De fato, freqüentemente ele irá divergir ou, quando converge, tende para uma outra raiz, caso exista, distinta daquela em consideração. Esta última situação pode ocorrer porque o método não permite um controle no sentido de convergência das iterações, ao contrário do que acontece com métodos como da biseção e da falsa posição. Busca-se, portanto, estabelecer condições que garantam a convergência do método para qualquer escolha de valor inicial  $x_0$  dentro de um dado intervalo. Estas condições são fornecidas pelo seguinte teorema [26].

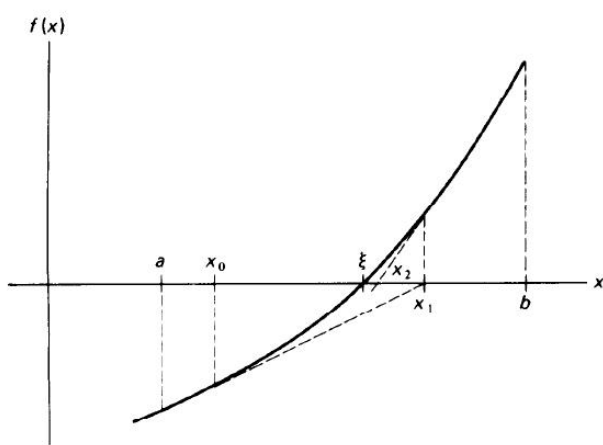
**TEOREMA 5.2.**

Seja  $f(x)$  diferenciável duas vezes no intervalo fechado  $[a, b]$ . Sendo as seguintes condições satisfeitas:

1.  $f(a)f(b) < 0$ ;
2.  $f'(x) \neq 0$ , para  $x \in [a, b]$ ;
3.  $f''(x) \leq 0$  ou  $f''(x) \geq 0$ ,  $\forall x \in [a, b]$ ;
4. nos pontos extremos  $a$  e  $b$ :

$$0 > \frac{f(a)}{f'(a)} > -(b-a) \text{ e } 0 < \frac{f(b)}{f'(b)} < b-a.$$

Então o método de Newton-Raphson converge para uma solução única  $\xi$  de  $f(x) = 0$  em  $[a, b]$  para qualquer escolha de  $x_0 \in [a, b]$ .



**Figura 5.4:** Convergência do método de Newton-Raphson. Dado  $x_0 \in [a, b]$ , o método rapidamente converge para  $\lim_{n \rightarrow \infty} x_n = \xi$ .

Alguns comentários a respeito destas condições são apropriados. Condições (1) e (2) garantem que há somente uma única solução em  $[a, b]$ . Condição (3) garante que o gráfico de  $f(x)$  possui somente uma única concavidade em  $[a, b]$  (côncava ou convexa) e que  $f'(x)$  é monótono neste intervalo. Finalmente, a condição (4) garante que a reta tangente a  $f(x)$  em cada ponto extremo do intervalo intercepta o eixo  $x$  dentro de  $[a, b]$ . Sucintamente, a condição (4) exige que  $f(x)$  seja suficientemente inclinada em  $[a, b]$  para que a raiz da reta tangente no ponto  $x_n$  esteja sempre dentro do intervalo considerado. A rápida convergência do método de Newton-Raphson, uma vez que as condições acima são satisfeitas, é ilustrada na figura 5.4.

As condições deste teorema são também suficientes para garantir a convergência do método da secante, desde que os dois valores iniciais  $x_0$  e  $x_1$  estejam ambos dentro de

$[a, b]$ . Neste caso, contudo a convergência à raiz pode se processar de duas formas distintas, dependendo da escolha feita para a ordem dos valores de  $x_0$  e  $x_1$ .

Um algoritmo que implemente o método de Newton-Raphson deve ter como parâmetros de entrada o(s) nome(s) da(s) rotina(s) que calcula(m)  $f(x)$  e  $f'(x)$ , o valor de  $x_0$  que garantidamente satisfaça as condições acima e o(s) parâmetro(s) de tolerância do erro de truncamento do método. Como critério de convergência, pode-se adotar novamente um ou mais dos critérios (5.7a) ou (5.7b).

Para que se possa escrever uma rotina robusta, portanto, seria necessário garantir as condições (1) – (4). Contudo, não há como verificar numericamente que as condições (2) e (3) estão sendo cumpridas para qualquer função  $f(x)$  apresentada à rotina. Para compensar esta deficiência, uma estratégia consiste em utilizar um algoritmo que mistura os métodos de Newton-Raphson e da bisecção. Sempre que a  $n + 1$ -ésima iteração esteja dentro do intervalo  $[a, b]$ , realiza-se nova iteração usando o método de Newton. Porém, se  $x_{n+1}$  estiver fora de  $[a, b]$  ou se  $|f(x_{n+1})|$  não estiver diminuindo rápido o suficiente, calcula-se a próxima iteração usando o método da bisecção. O programa 5.5 implementa justamente este tipo de algoritmo em Fortran 95.

**Listing 5.4:** Utiliza uma combinação dos métodos de Newton-Raphson e da bisecção para encontrar uma raiz dentro do intervalo fornecido.

```
! ***** SUBROTINA NEWTON_BISEC *****
! Busca uma raiz da função F(X) através de uma combinação dos métodos de
! Newton-Raphson e da bisecção.
! ***** Argumentos de entrada *****
```

```

! F_DFDX: Nome da subrotina que retorna os valores de F(X) e F'(X).
! x1,x2: Dois valores iniciais para o início da iteração.
! ERRABS: Primeiro critério de parada. Se ABS(F(Xn)) <= FTOL,
!         então Xn é aceito como raiz
! ERRREL: Segundo critério de parada: erro relativo.
!         Se ABS(Xn - Xn-1) <= XTOL*ABS(Xn), então Xn é aceito como raiz.
! NTOL: Numero máximo de iterações admitidas.
! ***** Argumentos de saída *****
! RAIZ: Melhor estimativa para a raiz.
! IFLAG: Um inteiro,
!        = -2, Método falhou. Não existe raiz em [x1,x2].
!        = -1, Método falhou. Nenhuma raiz foi encontrada em NTOL
!           iterações. O último valor encontrado para X é retornado.
!        = 0, Encerrou devido ao primeiro critério de parada ou por novas
!           iterações não alterarem o resultado.
!        = 1, Encerrou devido ao segundo critério de parada.
!
! Autor: Rudi Gaelzer, IF-UFRGS.
! Data: Junho/2008.
! (Baseado na função rtsafe do Numerical Recipes).
!
subroutine newton_bisec(f_dfdx, x1, x2, errabs, errrel, ntol, raiz, iflag)
integer, intent(in)      :: ntol
real(kind=dp), intent(in) :: x1, x2, errabs, errrel
integer, intent(out)     :: iflag
real(kind=dp), intent(out) :: raiz
INTERFACE
  subroutine f_dfdx(x, fx, dfdx)
  use Modelos_Computacionais_Dados
  real(kind=dp), intent(in)  :: x
  real(kind=dp), intent(out) :: fx, dfdx
  end subroutine f_dfdx
END INTERFACE
integer      :: j
real(kind=dp) :: df, dx, dxold, f, fh, fl, temp, xh, x1
call f_dfdx(x1, fl, df)
call f_dfdx(x2, fh, df)
if (fl*fh > 0.0_dp) then
  iflag = -2
  return
end if
if (fl == 0.0_dp) then
  raiz = x1
  iflag = 0
  return
else if (fh == 0.0_dp) then
  raiz = x2
  iflag = 0
  return
else if (fl < 0.0_dp) then ! Oriente o intervalo tal que f(x1) < 0.
  x1 = x1
  xh = x2
else
  xh = x1
  x1 = x2
end if
raiz = 0.5_dp*(x1 + x2)
dxold = abs(x2 - x1)
dx = dxold

```

```

call f_dfdx(raiz,f,df)
do j= 1, ntol                                ! Laço sobre o número permitido de iterações.
  if (((raiz-xh)*df-f)*((raiz-xl)*df-f) >= 0.0_dp & ! Se a iteração estiver
    .or. abs(2.0*f) > abs(dxold*df) ) then        ! fora de [x1,x2],
    dxold= dx                                    ! ou se não estiver
    dx= 0.5_dp*(xh - xl)                        ! convergindo rapidamente,
    raiz= xl + dx                               ! use bisecção.
    if (xl == raiz) then
      iflag= 0
      return
    end if
  else ! Iteração esta dentro de [x1,x2].
    dxold= dx
    dx= f/df
    temp= raiz
    raiz= raiz - dx
    if (temp == raiz) then
      iflag= 0
      return
    end if
  end if
  call f_dfdx(raiz,f,df)
  if(abs(f) <= errabs) then                    ! Primeiro critério de parada.
    iflag= 0
    return
  end if
  if(abs(dx) <= errrel*abs(raiz)) then ! Segundo critério de parada.
    iflag= 1
    return
  end if
  if (f < 0.0_dp) then
    xl=raiz
  else
    xh=raiz
  end if
end do
iflag= -1
return
end subroutine newton_bisec

```

Utilizando a rotina `newton_bisec` para encontrar a raiz  $x_1$  de  $p_3(x)$  dentro do intervalo  $[1, 2]$ , com `errabs= 0`, `errrel=  $10^{-10}$`  e `ntol= 50`, obteve-se as seguintes iterações:

$$\begin{aligned}
 w_0 &= 1.5000000000000000 \\
 w_1 &= 1.34782608695652 \\
 w_2 &= 1.32520039895091 \\
 w_3 &= 1.32471817399905 \\
 w_4 &= 1.32471795724479 \\
 w_5 &= 1.32471795724475.
 \end{aligned}$$

Pode-se notar que com somente 5 iterações, todas as casas decimais disponíveis para uma variável de dupla precisão foram obtidas.

## 5.3 RAÍZES COMPLEXAS DE FUNÇÕES ANALÍTICAS

Os métodos discutidos até este momento permitem a obtenção de uma raiz real isolada uma vez que uma aproximação prévia da raiz ou outra informação são conhecidas. A informação

prévia pode ser, por exemplo, o intervalo onde se sabe que um número ímpar de raízes reside, como no caso do método da bisecção.

Estes métodos não são muito satisfatórios quando todos os zeros de uma função são requeridos ou quando boas aproximações iniciais não estão disponíveis. Outra evidente limitação dos métodos até agora apresentados consiste na obtenção unicamente de raízes reais da função. Muitos problemas em física, engenharia, matemática ou outro campo de ciências naturais e exatas exigem o conhecimento também de raízes complexas de funções analíticas. Uma classe muito útil de funções onde todas estas limitações são evidentes é a dos polinômios, da qual a função  $p_3(x)$ , apresentada em (5.2a) faz parte. Das três raízes de  $p_3(x)$ , somente uma ( $x_1$ ) é real, enquanto as outras duas ( $x_2$  e  $x_3$ ) são complexas. Contudo, os métodos apresentados possibilitaram somente a obtenção de  $x_1$ .

Nesta seção, alguns métodos desenvolvidos para o cálculo numérico de raízes complexas de funções analíticas serão abordados. Particular ênfase será concedida ao Método de Müller e uma descrição sucinta será realizada acerca de métodos modernos que utilizam propriedades matemáticas oriundas da teoria de funções analíticas.

### 5.3.1 O MÉTODO DE MÜLLER

Este método relativamente recente, desenvolvido inicialmente por D. E. Müller [27], tem sido empregado em diversas aplicações distintas com bastante sucesso. Este método pode ser usado para descobrir qualquer número pré-fixado de raízes, reais ou complexas, de uma função analítica arbitrária. O método é iterativo, converge quase quadraticamente na vizinhança de uma raiz, não requer a forma analítica da derivada da função e obtém tanto raízes reais quanto complexas, mesmo quando estas são múltiplas.

Este método é global, no sentido de que o usuário não necessita fornecer uma aproximação inicial. Nesta seção o método será apresentado, omitindo qualquer discussão a respeito da sua convergência, e uma rotina que possibilita a obtenção de raízes tanto reais quanto complexas será também incluída. O problema de encontrar as todas as raízes de um polinômio terá uma atenção especial, uma vez que este problema surge com frequência em todos os ramos das ciências naturais e exatas.

O método de Müller é uma extensão do método da secante (seção 5.2.3). Para relembrar, no método da secante são fornecidas duas aproximações iniciais ( $x_i$  e  $x_{i-1}$ ) para a solução da equação  $f(x) = 0$ , sendo obtida uma terceira aproximação ( $x_{i+1}$ ) (Eq. 5.6). Esta aproximação consiste simplesmente na raiz da reta secante que passa pelos pontos  $\{x_i, f(x_i)\}$  e  $\{x_{i-1}, f(x_{i-1})\}$ . No método de Müller, são fornecidos 3 pontos:  $\{x_{i-2}, f(x_{i-2})\}$ ,  $\{x_{i-1}, f(x_{i-1})\}$  e  $\{x_i, f(x_i)\}$ , sendo a próxima aproximação à raiz,  $x_{i+1}$ , obtida como uma raiz da **parábola** que cruza os 3 pontos anteriores. Este método está ilustrado na figura 5.5.

Para interpolar a parábola  $p(x)$ :

$$p(x) = ax^2 + bx + c$$

com a função  $f(x)$  nos três pontos apresentados, basta determinar o valor das constantes  $a$ ,  $b$  e  $c$  de tal forma que  $p(x)$  corta  $f(x)$  nestes pontos, como apresentado na figura 5.5. Uma maneira equivalente de escrever  $p(x)$  é a seguinte:

$$p(x) = f(x_i) + f[x_{i-1}, x_i](x - x_i) + f[x_{i-2}, x_{i-1}, x_i](x - x_{i-1})(x - x_i), \quad (5.10a)$$

sendo

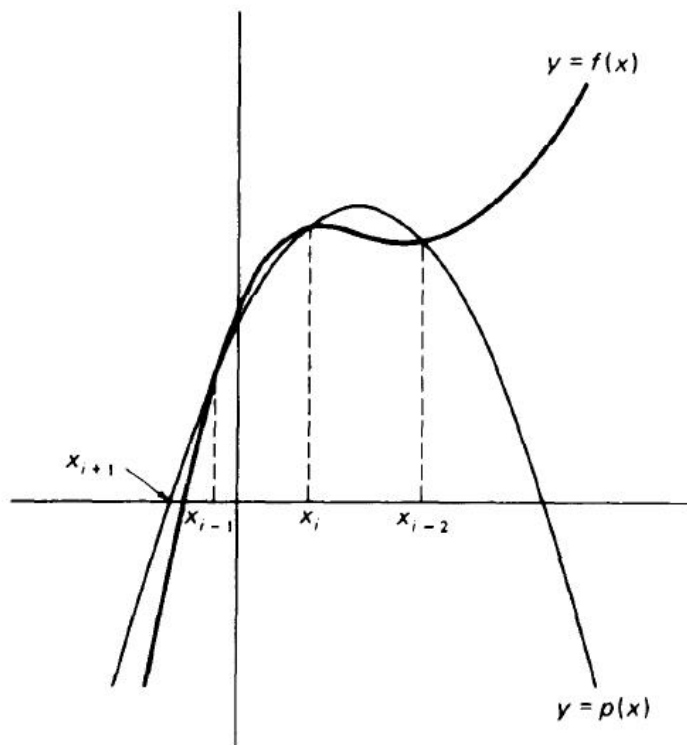
$$\begin{aligned} f[x_0, x_1] &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} \\ f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}, \end{aligned}$$

expressões particulares de

$$\begin{aligned} f[x_0] &= f(x_0) \\ f[x_0, \dots, x_k] &= \frac{f[x_1, \dots, x_k] - f[x_0, \dots, x_{k-1}]}{x_k - x_0}, \end{aligned}$$

onde  $f[x_0, \dots, x_k]$  é a  **$k$ -ésima diferença dividida de  $f(x)$  nos pontos  $x_0, \dots, x_k$** . Pode-se verificar facilmente que (5.10a) realmente interpola  $f(x)$  nos três pontos escolhidos. Uma vez que

$$(x - x_{i-1})(x - x_i) = (x - x_i)^2 + (x_i - x_{i-1})(x - x_i),$$



**Figura 5.5:** O método de Müller. Partindo de 3 pontos iniciais da função  $f(x)$ :  $\{x_{i-2}, f(x_{i-2})\}$ ,  $\{x_{i-1}, f(x_{i-1})\}$  e  $\{x_i, f(x_i)\}$ , a parábola  $p(x)$  é contruída, sendo a próxima aproximação para a raiz de  $f(x)$  tomada como a raiz  $x_{i+1}$  da parábola que mais se aproxima da raiz da função.

pode-se escrever  $p(x)$  também como

$$p(x) = f(x_i) + c_i(x - x_i) + f[x_{i-2}, x_{i-1}, x_i](x - x_i)^2, \quad (5.10b)$$

onde

$$c_i = f[x_{i-1}, x_i] + f[x_{i-2}, x_{i-1}, x_i](x_i - x_{i-1}).$$

A função  $p(x)$  apresentada em (5.10b) está escrita na forma de um polinômio do 2º grau de  $(x - x_i)$ . Buscando-se então uma raiz  $\xi$  de  $p(x)$ , tal que  $p(\xi) = 0$ , resulta

$$p(\xi) = f(x_i) + c_i(\xi - x_i) + f[x_{i-2}, x_{i-1}, x_i](\xi - x_i)^2 = 0 \implies \xi - x_i = \frac{-c_i \pm \sqrt{c_i^2 - 4f(x_i)f[x_{i-2}, x_{i-1}, x_i]}}{2f[x_{i-2}, x_{i-1}, x_i]},$$

a qual pode também ser escrita como

$$\xi - x_i = \frac{-2f(x_i)}{c_i \pm \sqrt{c_i^2 - 4f(x_i)f[x_{i-2}, x_{i-1}, x_i]}}. \quad (5.11)$$

Se o sinal de (5.11) for escolhido de tal forma que o denominador seja o maior possível<sup>1</sup> e o valor de  $\xi$  resultante for tomado como a aproximação  $x_{i+1}$  da raiz de  $f(x)$ , então a fórmula do método de Müller fica:

$$x_{i+1} = x_i - \frac{2f(x_i)}{c_i \pm \sqrt{c_i^2 - 4f(x_i)f[x_{i-2}, x_{i-1}, x_i]}}, \quad (5.12)$$

onde, conforme mencionado, o sinal do denominador de (5.12) deve ser escolhido de forma adequada.

Uma vez obtido  $x_{i+1}$ , o processo é então repetido utilizando-se  $x_{i-1}$ ,  $x_i$  e  $x_{i+1}$  em (5.12) para se obter  $x_{i+2}$  e assim consecutivamente. Se os zeros obtidos a partir de (5.12) forem reais, então a situação é ilustrada graficamente pela figura 5.5. Contudo, as raízes podem ser complexas mesmo que as aproximações iniciais sejam reais, bastando para isso que  $c_i^2 - 4f(x_i)f[x_{i-2}, x_{i-1}, x_i] < 0$ . Isto significa que mesmo que a raiz procurada seja real, aproximações intermediárias podem ser complexas. Porém, à medida que as iterações se aproximam da raiz real, a parte imaginária de  $x_{i+1}$  tende a zero. O algoritmo 5.3 apresenta a seqüência de passos necessária para implementar o método de Müller.

<sup>1</sup>Isto é, de tal forma que não possa ocorrer cancelamento, o que facilmente poderia gerar erros de arredondamento.

**Algoritmo 5.3** Implementação numérica do método de Müller.

1. Dadas  $x_0, x_1$  e  $x_2$ , três aproximações iniciais para a raiz  $\xi$  de  $f(x)$ . Calcule  $f(x_0)$ ,  $f(x_1)$  e  $f(x_2)$ .

2. Partindo de  $i = 2$ , calcule

$$\begin{aligned} h_i &= x_i - x_{i-1}; \quad h_{i-1} = x_{i-1} - x_{i-2} \\ f[x_{i-1}, x_i] &= (f(x_i) - f(x_{i-1})) / h_i \\ f[x_{i-2}, x_{i-1}] &= (f(x_{i-1}) - f(x_{i-2})) / h_{i-1}. \end{aligned}$$

3. Calcule

$$\begin{aligned} f[x_{i-2}, x_{i-1}, x_i] &= (f[x_{i-1}, x_i] - f[x_{i-2}, x_{i-1}]) / (h_i + h_{i-1}) \\ c_i &= f[x_{i-1}, x_i] + h_i f[x_{i-2}, x_{i-1}, x_i], \end{aligned}$$

4. Calcule

$$h_{i+1} = -2f(x_i) / \left\{ c_i \pm \sqrt{c_i^2 - 4f(x_i)f[x_{i-2}, x_{i-1}, x_i]} \right\},$$

escolhendo o sinal de modo a maximizar a magnitude do denominador.

5. Calcule

$$x_{i+1} = x_i + h_{i+1}.$$

6. Calcule

$$f(x_{i+1}).$$

7. Teste se um dos seguintes critérios for satisfeito:

- (a)  $|f(x_{i+1})| \leq \epsilon_1$  (Erro absoluto).
- (b)  $|x_{i+1} - x_i| \leq \epsilon_2 |x_{i+1}|$  (Erro relativo).
- (c) O número máximo de iterações é excedido.

8. Se o teste for verdadeiro, retorna a última aproximação obtida ( $x_{i+1}$ ). Se o teste for falso, calcule

$$f[x_i, x_{i+1}] = (f(x_{i+1}) - f(x_i)) / h_{i+1},$$

faça  $i = i + 1$  e recomece a partir do passo 3.



O programa 5.5 implementa o método de Müller, conforme delineado no algoritmo 5.3, em uma subrotina em Fortran 95. O método de Müller encontra uma raiz de cada vez. Para encontrar mais de uma raiz e evitar que as iterações venham a convergir para valores previamente encontrados, o programa 5.5 possui uma subrotina interna que implementa a técnica conhecida como **deflação**. Se, por exemplo, uma raiz  $\xi_1$  foi previamente obtida, a rotina calcula o próximo zero não a partir da função  $f(x)$  original, mas a partir da função *deflacionada* ou *reduzida*

$$f_1(x) = \frac{f(x)}{x - \xi_1}.$$

Desta forma, se a raiz  $\xi_1$  for única,  $\lim_{x \rightarrow \xi_1} f_1(x) = 1$  e o método deverá convergir para uma raiz  $\xi_2$  distinta. Pode acontecer de  $\xi_2 = \xi_1$  se esta raiz for dupla. Contudo, a função será novamente reduzida para cada nova raiz encontrada. Assim, se os zeros  $\xi_1, \xi_2, \dots, \xi_r$  foram previamente obtidos, o próximo zero será obtido a partir da função reduzida

$$f_r(x) = \frac{f(x)}{(x - \xi_1)(x - \xi_2) \cdots (x - \xi_r)}.$$

**Listing 5.5:** Implementação do método de Müller em Fortran 95.

```
! ***** SUBROTINA MULLER *****
! Encontra as raízes de uma função analítica unívoca pelo Método de Muller.
!
! Argumentos:
!   fn: Função analítica unívoca f(z) cujas raízes são procuradas (Entrada)
!   nnov: Número total de raízes novas a serem encontradas. (Entrada)
!   nprev: Número de raízes previamente conhecidas. (Entrada)
!   maxit: Número máximo de chamadas da função fn(z) por raiz. (Entrada)
!   errabs: Primeiro critério de parada. (Entrada)
!           Iterações são interrompidas se abs(fn(z)) .lt. errabs.
!   errrel: Segundo critério de parada. (Entrada)
!           Iterações são interrompidas se abs(h) .lt. errrel*abs(z).
!   zeros: Vetor que contém as raízes de fn(z). (Entrada/Saída)
!           zeros(1), ..., zeros(nprev): raízes previamente conhecidas.
!           zeros(nprev+1), ..., zeros(n): (ent.) aproximações iniciais
!                                   para raízes.
!                                   (sai.) raízes encontradas.
!   fator: (OPCIONAL) Fator multiplicativo para valores iniciais. (Entrada)
!           Para z= zeros(i), os 3 primeiros pontos para ajustar a parábola são
!           z, z + fator*h e z - fator*h, sendo h um valor fixo.
!   fnreal: (OPCIONAL) Variável lógica para raízes reais. (Entrada)
!           fnreal = .true. se todas as raízes são reais.
!           fnreal = .false. se há raízes complexas (valor padrão).
!   iterac: (OPCIONAL) Vetor com o número de iterações realizadas
!           para cada raiz. (saída)
!
! Autor: Rudi Gaelzer, IF-UFRGS.
! Data: Abril/2011 (versão 2).
!
subroutine muller (fn, nnov, nprev, maxit, errabs, &
                  errrel, zeros, fator, fnreal, iterac)
!
implicit none
! Variáveis mudas.
logical, optional, intent(in) :: fnreal
integer, intent(in) :: nnov, nprev, maxit
integer, dimension(nnov), optional, intent(out) :: iterac
real(kind= dp), intent(in) :: errabs, errrel
real(kind= dp), optional, intent(in) :: fator
complex(kind= dp), dimension(nnov+nprev), intent(inout) :: zeros
! Variáveis locais.
```



```

logical          :: tes_maxit, tes_g_mxit, tes_real, tes_iter
integer          :: ntot, i, n_it
real(kind= dp)   :: eps1, eps2, teste, h_ini= 0.1_dp, tiny1= tiny(1.0_dp)
real(kind= dp), parameter :: big1= huge(1.0_dp)
complex(kind= dp), parameter :: z0= (0.0_dp,0.0_dp), z1o10= (0.1_dp,0.0_dp)
complex(kind= dp), parameter :: z2= (2.0_dp,0.0_dp), z4= (4.0_dp,0.0_dp)
complex(kind= dp) :: c, den, divdf1, divdf2, dvdf1p, fzf, z_inc, zrprev
complex(kind= dp) :: fzrdf1, fzrprv, h, hprev, zero, sqr, him2, him1
INTERFACE
    function fn(z)
    use Modelos_Computacionais_Dados
    complex(kind= dp), intent(in) :: z
    complex(kind= dp)             :: fn
    end function fn
END INTERFACE
!
if(nnov < 1)then
    print*, 'O argumento nnov deve ser >= 1.'
    stop
end if
!                               Inicializações.
tes_real= .false. ; tes_iter= .false. ; tes_g_mxit= .false.
if(present(fnreal))tes_real= fnreal
if(present(fator))h_ini= fator*h_ini
if(present(iterac))tes_iter= .true.
if(big1*tiny1 < 1.0_dp)tiny1= 1.0_dp/big1
eps1 = max(errrel, 10.0_dp*epsilon(1.0_dp)) ! Erro de primeira espécie.
eps2 = max(errabs, 10.0_dp*tiny1)          ! Erro de segunda espécie.
ntot= nnov + nprev
!
l_raízes: do i = nprev + 1, ntot
    n_it = 0
    tes_maxit= .false.
    !                               Calcule os três primeiros valores da i-ésima raiz como
    !                               zeros(i) + h, zeros(i)- h, zeros(i)
    zero = zeros(i)
    h = z1o10*cmplx(h_ini, kind= dp)
    if(abs(zero) > h_ini) h= z1o10*zero
    him2= h
    call dflac(zero, him2, i, fzf, dvdf1p) ! f(zero + h)
    him1= -h
    call dflac(zero, him1, i, fzf, fzrprv) ! f(zero - h)
    hprev = him1 - him2
    zrprev= zero + him1
    dvdf1p = (fzrprv - dvdf1p)/hprev
    l_iter: do
        l_div: do
            z_inc= z0
            call dflac(zero, z_inc, i, fzf, fzrdf1)
            zero= zero + z_inc
            if (tes_maxit)then
                tes_g_mxit= .true.
                exit l_iter
            end if
            h= zero - zrprev
            !                               Testa convergência da segunda espécie.
            if (max(abs(fzf),abs(fzrdf1)) < eps2)exit l_iter
            !                               Testa convergência da primeira espécie.
            teste= abs(h) - eps1*abs(zero)

```

```

        if (teste < 0.0_dp) exit l_iter
!           Verifique se valor iterado diverge da raiz.
        if (abs(fzrdfl) < 10.0_dp*abs(fzrprv)) exit l_div
        h = cmplx(0.5_dp, kind= dp)*h
        zero = zero - h
    end do l_div
!           Inicia algoritmo principal.
    divdf1 = (fzrdfl - fzrprv)/h
    divdf2 = (divdf1 - dvdf1p)/(h + hprev)
    hprev = h ; zrprev = zero
    dvdf1p = divdf1
    c = divdf1 + h*divdf2
    sqr = c*c - z4*fzrdfl*divdf2
    if (tes_real .and. (real(sqr) < 0.0_dp)) sqr = z0
    sqr = sqrt(sqr)
    teste= sign(1.0_dp, real(c)*real(sqr)+aimag(c)*aimag(sqr))
    den = c + cmplx(teste, kind= dp)*sqr
    h = -z2*fzrdfl/den
    fzrprv = fzrdfl
    zero = zero + h
end do l_iter
zeros(i) = zero
if(tes_iter)iterac(i - nprev)= n_it
end do l_raízes
if(tes_g_mxit)print*, 'Método nao convergiu em', maxit, &
    'passos para 1 ou mais raízes.'
return
CONTAINS
    subroutine dflac(zero, z_inc, i, fzero, fzrdfl)
    integer, intent(in)          :: i
    complex(kind= dp), intent(in) :: zero
    complex(kind= dp), intent(out) :: fzero, fzrdfl
    complex(kind= dp), intent(inout) :: z_inc
    logical                      :: t_den= .true.
    integer                      :: j
    real(kind= dp), dimension(2) :: v_sft
    complex(kind= dp) :: root, den
!
    l_den: do
        if (n_it == maxit)then
            tes_maxit= .true.
            return
        end if
        n_it = n_it + 1
        root= zero + z_inc
        fzero = fn(root)
        fzrdfl = fzero
        l_deflac: do j = 2, i
            den = root - zeros(j-1)
! Teste para evitar singularidade ou overflow.
            if ((abs(den) < tiny1) .or. &
                (big1*min(abs(den),1.0_dp) <= abs(fzrdfl))) then
! Desloca o ponto aleatoriamente.
                call random_number(v_sft)
                v_sft= 10.0_dp*(v_sft - 0.5_dp)*eps1
                z_inc= z_inc + cmplx(v_sft(1), v_sft(2), kind= dp)
                t_den= .false.
                exit l_deflac
            else

```

```

        fzrdbl = fzrdbl/den
        t_den= .true.
    end if
end do l_deflac
if(t_den) exit l_den
end do l_den
return
end subroutine dflac
end subroutine muller

```

Usando a rotina muller para encontrar as três raízes de  $p_3(x)$ , dadas em (5.2b-d), obteve-se os seguintes resultados.

- Com errabs= 0 e errrel= 0.1d0, os resultados são:

$$\begin{aligned}
 x_1 &= (-0.662461653676260, -0.562201573489122) \\
 x_2 &= (-0.662358990777978, 0.562279503661044) \\
 x_3 &= (1.32471793173699, -6.241718792293410E - 008)
 \end{aligned}$$

com erros relativos respectivamente iguais a

$$1.483654014681657E - 004, 1.700701981827868E - 008, 5.089998023306140E - 008$$

- Com errabs= 0 e errrel= 1.0d-5, os resultados são:

$$\begin{aligned}
 x_1 &= (-0.662358978622395, -0.562279512062290) \\
 x_2 &= (-0.662358978622373, 0.562279512062301) \\
 x_3 &= (1.32471795724475, 9.491574411619214E - 027)
 \end{aligned}$$

com erros relativos iguais a

$$2.878544463935637E - 014, 0.000000000000000E + 000, 7.164977540849939E - 027$$

- Com errabs= 0 e errrel= 1.0d-8, os resultados são:

$$\begin{aligned}
 x_1 &= (-0.662358978622373, -0.562279512062301) \\
 x_2 &= (-0.662358978622373, 0.562279512062301) \\
 x_3 &= (1.32471795724475, 8.128422767397306E - 027)
 \end{aligned}$$

com erros relativos iguais a

$$0.000000000000000E + 000, 0.000000000000000E + 000, 6.135964808919362E - 027$$

Ou seja, observa-se que o método rapidamente converge para os valores exatos das raízes.



## INTERPOLAÇÃO, EXTRAPOLAÇÃO E AJUSTE DE FUNÇÕES



**[EM CONSTRUÇÃO]**



# PROBLEMAS DE VALOR INICIAL

## 7.1 INTRODUÇÃO

Neste capítulo serão discutidos alguns métodos de solução numérica de Equações Diferenciais Ordinárias (ODE) que fazem parte de Problemas de Valor Inicial (PVI).

Em ciências exatas ou naturais, grande parte dos problemas existentes são descritos por equações diferenciais, cujas soluções gerais devem ser particularizadas por condições iniciais e/ou condições de contorno. Quando ocorrem somente as primeiras, diz-se que o problema é de valor inicial.

Um problema de valor inicial pode ser definido da seguinte forma. Sendo  $x \geq x_0$  um parâmetro que varia de forma independente no problema e  $y(x)$  uma função da variável  $x$ , a função  $y(x)$  será determinada, em um problema de valor inicial, a partir da solução da equação diferencial ordinária de ordem  $n$

$$F\left(y^{(n)}(x), y^{(n-1)}(x), \dots, y'(x), y(x), x\right) = f(x), \quad (7.1a)$$

juntamente com as condições iniciais

$$g_0\left(y(x_0), y'(x_0), \dots, y^{(n-1)}(x_0)\right) = a_0 \quad (7.1b)$$

$$g_1\left(y(x_0), y'(x_0), \dots, y^{(n-1)}(x_0)\right) = a_1 \quad (7.1c)$$

$$\begin{aligned} & \vdots \\ g_{n-1}\left(y(x_0), y'(x_0), \dots, y^{(n-1)}(x_0)\right) &= a_{n-1}, \end{aligned} \quad (7.1d)$$

sendo  $F(\dots)$  um funcional qualquer de  $y(x)$  e suas derivadas até a ordem  $n$ ,  $f(x)$  uma função de  $x$ ,  $g_0(\dots), \dots, g_{n-1}(\dots)$  funcionais das condições iniciais  $y(x_0), \dots, y^{(n-1)}(x_0)$  e  $a_0, \dots, a_{n-1}$  constantes. Em problemas realísticos na física, geralmente os funcionais  $g_0, \dots, g_{n-1}$  são lineares em  $\{y(x_0), \dots, y^{(n-1)}(x_0)\}$  mas o funcional  $F$  pode ser não linear em  $\{y(x), \dots, y^{(n)}(x)\}$ .

## 7.2 EQUAÇÕES DE DIFERENÇAS FINITAS LINEARES

A solução numérica do problema (7.1) envolve a discretização da ODE, ou seja, a transformação da equação diferencial em uma equação de diferenças finitas. Para exemplificar, pode-se considerar o PVI linear

$$y' = y, \quad (7.2a)$$

$$y(x_0) = a_0, \quad (7.2b)$$

cuja solução é  $y(x) = a_0 e^{x-x_0}$ . Para discretizar este PVI de uma forma trivial, considera-se a definição de uma derivada e omite-se o símbolo de limite:

$$y'(x) \rightarrow \frac{y(x+h) - y(x)}{h},$$

sendo  $h$  um pequeno incremento em  $x$ . Desta forma, é possível afirmar que a solução no ponto  $x+h$  é obtida a partir do conhecimento da solução em  $x$  através de

$$y(x+h) = (1+h)y(x).$$

Partindo-se então do valor inicial  $y_0 = a_0$ , obtem-se  $y_1 = y(x+h) = (1+h)y_0 = (1+h)a_0$ ,  $y_2 = y(x+2h) = (1+h)y_1 = (1+h)^2 a_0$ , etc. Por indução, pode-se deduzir que  $y_n = (1+h)^n a_0$ . Escrevendo agora o  $n$ -ésimo valor de  $x$  como  $x_n = x_0 + nh$  e chamando  $x_n \equiv x$  e  $y_n \equiv y$ , temos

$$h = \frac{x - x_0}{n}. \text{ Portanto, } y = \left(1 + \frac{x - x_0}{n}\right)^n a_0.$$

Empregando agora a identidade

$$\lim_{n \rightarrow \infty} \left(1 + \frac{z}{n}\right)^n = e^z,$$

resulta que a solução discretizada do PVI (7.2), quando  $n \rightarrow \infty$ , reduz-se a  $y = a_0 e^{x-x_0}$ , a qual é justamente a sua solução. Contudo, neste limite o resultado deverá estar bastante contaminado pela propagação de erros de arredondamento; além disso, deseja-se buscar métodos que forneçam resultados mais acurados já para as primeiras iterações.

Alguns exemplos de equações de diferenças finitas e suas soluções são

$$\begin{aligned} y_{n+1} - y_n &= 1 & \implies & y_n = n + c \\ y_{n+1} - y_n &= n & \implies & y_n = \frac{n(n-1)}{2} + c \\ y_{n+1} - (n+1)y_n &= 0 & \implies & y_n = cn! \end{aligned}$$

Vamos considerar com algum detalhe uma equação de diferenças finitas linear de ordem  $N$  com coeficientes constantes

$$y_{n+N} + a_{N-1}y_{n+N-1} + \dots + a_1y_{n+1} + a_0y_n = 0. \quad (7.3)$$

Esta equação deve possuir  $N$  soluções linearmente independentes, as quais são da forma  $y_n = \beta^n$ ,  $\forall n$ , sendo  $\beta$  uma constante. Substituindo esta solução em (7.3) resulta

$$\beta^{n+N} + a_{N-1}\beta^{n+N-1} + \dots + a_1\beta^{n+1} + a_0\beta^n = 0.$$

Dividindo-se por  $\beta^n$ , resulta a *equação característica*

$$\beta^N + a_{N-1}\beta^{N-1} + \dots + a_1\beta + a_0 = 0, \quad (7.4)$$

a qual fornece as raízes de um polinômio de grau  $N$ . Assumindo que todas as raízes  $\beta_1, \beta_2, \dots, \beta_N$  são distintas, a solução geral de (7.3) pode ser finalmente escrita como

$$y_n = c_1\beta_1^n + c_2\beta_2^n + \dots + c_N\beta_N^n, \quad n = 0, 1, 2, \dots$$

Se os valores de  $y_n$   $n = 0, \dots, N-1$  forem dados, estes, juntamente com (7.3), formam um **problema de valor inicial de diferenças finitas**, o qual pode ser resolvido explicitamente resultando na solução particular para  $y_n$ .

Como um exemplo, a equação de diferenças

$$y_{n+3} - 2y_{n+2} - y_{n+1} + 2y_n = 0$$

possui a equação característica

$$\beta^3 - 2\beta^2 - \beta + 2 = 0,$$

cujas raízes são  $\beta_1 = 1$ ,  $\beta_2 = -1$  e  $\beta_3 = 2$ . Portanto, a solução geral é

$$y_n = c_1 1^n + c_2 (-1)^n + c_3 2^n.$$

Sendo agora dados  $y_0 = 0$ ,  $y_1 = 1$  e  $y_2 = 1$ , então

$$y_0 = c_1 + c_2 + c_3 = 0$$

$$y_1 = c_1 - c_2 + 2c_3 = 1$$

$$y_2 = c_1 + c_2 + 4c_3 = 1$$

o qual forma um sistema linear nas constantes, cuja solução é  $c_1 = 0$ ,  $c_2 = -1/3$  e  $c_3 = 1/3$ . Portanto, a solução particular é

$$y_n = -\frac{1}{3}(-1)^n + \frac{2^n}{3}.$$

Se alguma das raízes da equação característica (7.4) for dupla ( $\beta_1$ , por exemplo), então uma segunda solução da mesma é  $n\beta_1^n$ . No mesmo espírito, se algum par de raízes de (7.4) forem complexo conjugadas ( $\beta_1 = \beta_2^*$ , por exemplo), então estas podem ser escritas na sua forma polar e  $c_1\beta_1 + c_2\beta_2$  pode ser reescrita na forma  $r^n (C_1 \cos n\theta + C_2 \sin n\theta)$ , onde  $r = |\beta_1|$  e  $\theta = \arg \beta_1$ .

As propriedades das equações de diferenças finitas consideradas nesta seção serão úteis para os métodos desenvolvidos no restante deste capítulo.



## 7.3 INTEGRAÇÃO NUMÉRICA POR SÉRIE DE TAYLOR

Considerando-se inicialmente um PVI de primeira ordem na forma

$$y' = f(x, y) \quad (7.5a)$$

$$y(x_0) = y_0. \quad (7.5b)$$

A função  $f(x, y)$  pode ser linear ou não linear em  $y$ , mas é assumido que esta é diferenciável em qualquer ordem em  $x$  e  $y$ . Se  $\partial f / \partial y$  for contínua no domínio de interesse, então a solução de (7.5) é única.

Sendo então  $y(x)$  a solução exata de (7.5), pode-se desenvolver  $y(x)$  em uma série de Taylor em torno do ponto  $x = x_0$ :

$$y(x) = y_0 + (x - x_0) y'(x_0) + \frac{1}{2!} (x - x_0)^2 y''(x_0) + \dots \quad (7.6)$$

O valor de  $y_0$  é suposto dado, mas as derivadas na série acima não são conhecidas uma vez que  $y(x)$  é desconhecido. Contudo, dada a hipótese de  $f(x, y)$  ser diferenciável, as derivadas de (7.6) podem ser obtidas tomando-se a derivada total de (7.5a) em relação a  $x$ , lembrando sempre que  $y$  é função de  $x$ . Assim, obtem-se para as primeiras derivadas:

$$\begin{aligned} y' &= f \\ y'' &= \frac{df}{dx} = f_x + f_y y' \\ &= f_x + f_y f \\ y''' &= \frac{d^2 f}{dx^2} = f_{xx} + f_{xy} f + f_{yx} f + f_{yy} f^2 + f_y f_x + f_y^2 f \\ &= f_{xx} + 2f_{xy} f + f_{yy} f^2 + f_y f_x + f_y^2 f. \end{aligned}$$

Continuando desta maneira, pode-se expressar qualquer derivada de  $y$  em termos de  $f(x, y)$  e suas derivadas parciais. Contudo, para derivadas de mais alta ordem a expressão resultante torna-se cada vez mais extensa.

Por razões práticas, portanto, deve-se limitar o número de termos em (7.6) a um valor pequeno e esta limitação restringe o valor de  $x$  para o qual a série (7.6) truncada resulta em um valor para  $y(x)$  razoavelmente acurado. Assumindo que a série (7.6) truncada fornece uma boa aproximação para um passo de comprimento  $h$ , isto é,  $x - x_0 = h$ , pode-se calcular  $y$  em  $x_0 + h$ , recalculer suas derivadas  $y'$ ,  $y''$ , etc, em  $x = x_0 + h$  e então usar (7.6) novamente para calcular  $y$  em  $x_0 + 2h$ ; e assim sucessivamente. Prosseguindo desta maneira, obtem-se uma conjunto discreto de valores  $\{y_n\}$  que são aproximações da solução correta nos pontos  $x_n = x_0 + nh$  ( $n = 0, 1, 2, \dots$ ). No restante deste capítulo, a solução exata de (7.5) no ponto  $x_n$  será denotada por  $y(x_n)$ , enquanto que a solução aproximada será denotada por  $y_n$ .

Para formalizar este procedimento, introduz-se o operador

$$T_k(x, y) = f(x, y) + \frac{h}{2!} f'(x, y) + \frac{h^2}{3!} f''(x, y) + \dots + \frac{h^{k-1}}{k!} f^{(k-1)}(x, y), \quad k = 1, 2, \dots, \quad (7.7)$$

onde  $f^{(j)}(x, y)$  denota a  $j$ -ésima derivada total de  $f(x, y)$  com relação a  $x$ . Assim, truncando-se a série (7.6) até o  $k$ -ésimo termo, pode-se escrever:

$$y(x_0 + h) \approx y_0 + h \{T_k(x_0, y(x_0))\}. \quad (7.8a)$$

O erro local cometido ao se tomar o passo de  $x_n$  para  $x_{n+1}$  utilizando o método de Taylor na ordem  $k$ , é fornecido pelo próximo termo da série de Taylor truncada:

$$E = \frac{h^{k+1}}{(k+1)!} y^{(k+1)}(\xi) = \frac{h^k}{(k+1)!} f^{(k)}(\xi, y(\xi)), \quad x_n < \xi < x_n + h. \quad (7.8b)$$

Neste caso, diz-se que o algoritmo de Taylor é de **ordem  $k$** . O algoritmo 7.1 implementa o método de Taylor.

**Algoritmo 7.1 Algoritmo de Taylor de ordem  $k$ .**

Para encontrar uma solução aproximada do PVI

$$\begin{aligned} y' &= f(x, y) \\ y(a) &= y_0 \end{aligned}$$

sobre o intervalo  $[a, b]$ :

1. Escolha um passo  $h = (b - a) / N$ . Defina

$$x_n = a + nh, \quad n = 0, 1, \dots, N.$$

2. Obtenha as aproximações  $y_n$  de  $y(x_n)$  a partir da fórmula de recorrência

$$y_{n+1} = y_n + hT_k(x_n, y_n), \quad n = 0, 1, \dots, N - 1,$$

onde  $T_k(x_n, y_n)$  está definido em (7.7).

**7.3.1 O MÉTODO DE EULER**

Arbitrando  $k = 1$  em (7.8a,b), obtem-se o **método de Euler** e o seu erro local. Seguindo a representação apresentada no algoritmo 7.1, a fórmula para o método de Euler fica:

$$y_{n+1} = y_n + f(x_n, y_n), \quad (7.9a)$$

$$E = \frac{1}{2} f'(x_n, y_n) h^2, \quad x_n < \xi < x_{n+1}. \quad (7.9b)$$

Para este método, existe uma outra estimativa de erro que pode ser denominada de *erro global* ou *erro de convergência*. Trata-se de um limite superior no erro cometido ao se utilizar repetidas vezes este método com um passo fixo  $h$ , variando  $x$  entre  $x_0$  e algum limite superior  $x = b$ . Sendo  $x_n = x_0 + nh$ , este erro é mensurado como

$$e_n = y(x_n) - y_n,$$

isto é, o erro realizado no processo de discretização empregado pelo método de Euler. Nesta definição,  $y_n$  é o valor aproximado dado pelo método de Euler (Eq. 7.9a), enquanto que  $y(x_n)$  é a solução exata do PVI no ponto  $x_n$ . Uma estimativa máxima para  $e_n$  é dada pelo Teorema 7.1 abaixo.

**Teorema 7.1.** *Seja  $y_n$  a solução aproximada de (7.5) obtida pelo Método de Euler (7.9). Se a solução exata de (7.5),  $y = y(x)$ , possui sua derivada segunda contínua no intervalo  $[x_0, b]$  e se neste intervalo as desigualdades*

$$|f_y(x, y)| \leq L, \quad |y''(x)| < Y$$

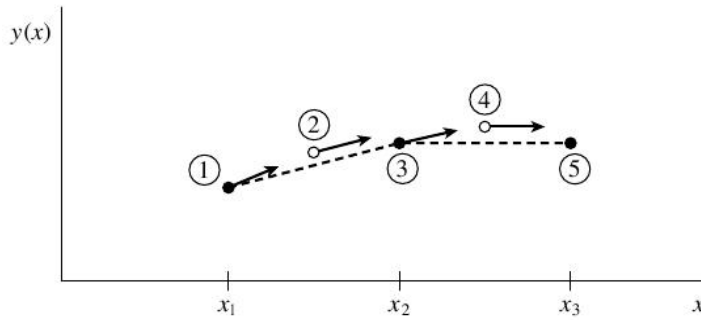
*são satisfeitas para certas constantes positivas  $L$  e  $Y$ , o erro de convergência  $e_n = y(x_n) - y_n$  do Método de Euler no ponto  $x_n = x_0 + nh$  tem seu valor máximo dado por*

$$|e_n| \leq \frac{hY}{2L} \left[ e^{(x_n - x_0)L} - 1 \right].$$

O teorema 7.1 mostra que o erro é  $\mathcal{O}(h)$ , isto é,  $e_n \rightarrow 0$  proporcionalmente a  $h$  se  $x = x_n$  é mantido fixo. Por outro lado, para um  $h$  fixo, o erro aumenta na ordem  $e^{xL}$  quando  $x = x_n$  se afasta de  $x_0$ .

**7.4 O MÉTODO DE RUNGE-KUTTA**

O Método de Euler não é muito útil para a solução de problemas que demandam uma maior acurácia; a sua utilidade surge quando o programador necessita ter somente uma idéia da



**Figura 7.1:** Método do ponto médio ou Método de Runge-Kutta de segunda ordem. Acurácia em segunda ordem é obtida usando a derivada no início do intervalo para encontrar uma solução intermediária no ponto médio do intervalo e, então, usando a derivada no ponto médio ao longo de todo o intervalo.

ordem de grandeza e da tendência da solução de (7.5). Por outro lado, o algoritmo de Taylor (7.1) para uma ordem  $k$  alta é também impraticável, pois necessita do conhecimento das derivadas de ordem  $k - 1$  de  $f(x, y)$ .

O Método de Runge-Kutta foi desenvolvido com o intuito de obter maior acurácia que o Método de Euler e, ao mesmo tempo, evitar a necessidade de se conhecer derivadas de ordens altas. Para tanto, o método faz uso da estratégia de calcular os valores de  $f(x, y)$  em pontos intermediários para cada passo da integração de (7.5).

A deficiência no Método de Euler se deve ao fato de que a fórmula (7.9a) avança a solução por um intervalo  $h$  usando somente informações somente no início do intervalo, isto é, no ponto  $x = x_n$ . O método não utiliza nenhuma outra informação sobre a variação de  $f(x, y)$  no intervalo  $[x_n, x_{n+1}]$ . O mesmo pode ser dito do algoritmo de Taylor em qualquer ordem.

### 7.4.1 O MÉTODO DE RUNGE-KUTTA DE SEGUNDA ORDEM OU O MÉTODO DO PONTO MÉDIO

Numa tentativa de remediar esta deficiência, pode-se realizar primeiramente um passo tentativo até o ponto médio no intervalo  $[x_n, x_{n+1}]$  e então utilizar os valores de  $x$  e  $y$  neste ponto médio para computar o passo real ao longo de todo o intervalo de comprimento  $h$ . Esta sequência de 2 passos intermediários para um passo completo é quantitativamente descrito pelo sistema de equações

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + h/2, y_n + k_1/2) \\ y_{n+1} &= y_n + k_2 + \mathcal{O}(h^3). \end{aligned}$$

Como indicado no termo de erro, o uso do ponto médio torna o método acurado em segunda ordem. A figura 7.1 ilustra a aplicação deste método. Já o algoritmo 7.2 mostra como este método pode ser implementado em um programa de computador.

O erro local do Método de Runge-Kutta de ordem 2 é  $\mathcal{O}(h^3)$ , ao passo que o erro local do Método de Euler é  $\mathcal{O}(h^2)$ . Isto significa que é possível usar um passo  $h$  com o primeiro método para se obter a mesma acurácia do segundo. O preço que se paga é que para cada passo  $h$  o funcional  $f(x, y)$  é calculado 2 vezes, no início e no ponto médio do intervalo. Expressões com erros ainda menores podem ser obtidos utilizando-se informações de derivadas de ordens mais altas no ponto médio. Contudo, em vez de se utilizar esta complicação adicional, o uso prático recomenda o emprego do Método de Runge-Kutta de ordem 4 (seção 7.4.2).

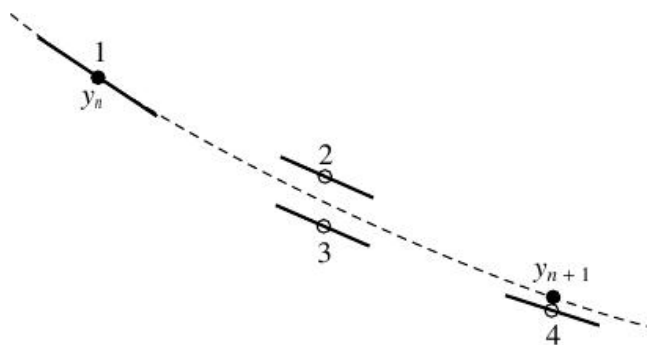
### 7.4.2 O MÉTODO DE RUNGE-KUTTA DE QUARTA ORDEM

Talvez o método mais empregado para a solução de PVI's, o Método de Runge-Kutta de quarta ordem faz uso das informações fornecidas por  $f(x, y)$  em 3 pontos (ou fórmulas) intermediários antes de calcular a aproximação para  $y_{n+1}$ .

Sem demonstração, as fórmulas envolvidas neste método são:

$$k_1 = hf(x_n, y_n) \quad (7.10a)$$

$$k_2 = hf(x_n + h/2, y_n + k_1/2) \quad (7.10b)$$



**Figura 7.2:** Método de Runge-Kutta de quarta ordem. Em cada passo a derivada é calculada 4 vezes: uma vez no ponto inicial, duas vezes no ponto médio e uma vez no ponto final. Destas derivadas o valor final da solução do PVI é calculado.

$$k_3 = hf(x_n + h/2, y_n + k_2/2) \quad (7.10c)$$

$$k_4 = hf(x_n + h, y_n + k_3) \quad (7.10d)$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(h^5). \quad (7.10e)$$

O método de ordem 4 requer 4 cálculos de  $f(x, y)$  por passo  $h$ . Isto significa que este método deve ser superior que o método de ordem 2 se o valor de  $h$  neste caso puder ser, pelo menos, 2 vezes maior que o valor para o método de ordem 2, para se obter a mesma acurácia. Caso contrário, é melhor usar o algoritmo 7.2 ou algum outro método de solução de um PVI. A figura 7.2 ilustra a aplicação do método e o algoritmo 7.3 mostra a sua implementação.

## 7.5 SISTEMAS DE EQUAÇÕES DIFERENCIAIS

As fórmulas (7.10a-e) para a aplicação do Método de Runge-Kutta de quarta ordem supõe a existência de um PVI simples do tipo (7.5) o qual consiste em uma equação diferencial de primeira ordem (linear ou não linear) com uma condição inicial simples. Contudo, grande parte dos problemas que surgem em ciências exatas e naturais envolvem PVI's compostos por uma ou mais equações diferenciais de segunda ordem ou ordens mais altas, com um correspondente número de condições iniciais. Desejamos então estender o método apresentado na seção 7.4.2 (ou qualquer outro) para esta situação mais geral.

Para exemplificar a generalização do método, vamos considerar o caso de uma ODE de ordem  $N$  com  $N$  condições iniciais. A extensão para o caso onde há mais de uma equação diferencial, inclusive de diferentes ordens, segue diretamente do exemplo apresentado. O PVI a ser

---

### Algoritmo 7.2 O Método de Runge-Kutta de ordem 2.

---

Dado o PVI

$$y' = f(x, y), \quad y(x_0) = y_0,$$

aproximações  $y_n$  para  $y(x_n)$ , sendo  $x_n = x_0 + nh$  para um passo  $h$  fixo e  $n = 0, 1, \dots$ , são obtidas usando-se a seguinte sequência de passos:

1. Calcule  $k_1$  dado por

$$k_1 = hf(x_n, y_n).$$

2. A partir de  $k_1$ , calcule  $k_2$  dado por

$$k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right).$$

3. A partir de  $k_2$ , calcule a solução  $y_{n+1}$  dada por

$$y_{n+1} = y_n + k_2.$$


---

**Algoritmo 7.3 O Método de Runge-Kutta de ordem 4.**

Dado o PVI

$$y' = f(x, y), \quad y(x_0) = y_0,$$

aproximações  $y_n$  para  $y(x_n)$ , sendo  $x_n = x_0 + nh$  para um passo  $h$  fixo e  $n = 0, 1, \dots$ , são obtidas usando-se a seguinte sequência de passos:

1. Calcule  $k_1$  dado por

$$k_1 = hf(x_n, y_n).$$

2. A partir de  $k_1$ , calcule  $k_2$  dado por

$$k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right).$$

3. A partir de  $k_2$ , calcule  $k_3$  dado por

$$k_3 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right).$$

4. A partir de  $k_3$ , calcule  $k_4$  dado por

$$k_4 = hf(x_n + h, y_n + k_3).$$

5. A partir de  $k_1, k_2, k_3$  e  $k_4$ , calcule a aproximação  $y_{n+1}$  dada por

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

considerado pode ser escrito a partir de (7.1) como

$$y^{(N)} = f\left(x, y(x), y'(x), \dots, y^{(N-1)}(x)\right), \quad (7.11a)$$

juntamente com as condições iniciais

$$g_0\left(y(x_0), y'(x_0), \dots, y^{(N-1)}(x_0)\right) = a_0 \quad (7.11b)$$

$$g_1\left(y(x_0), y'(x_0), \dots, y^{(N-1)}(x_0)\right) = a_1 \quad (7.11c)$$

$$\vdots \quad \vdots \quad (7.11d)$$

$$g_{N-1}\left(y(x_0), y'(x_0), \dots, y^{(N-1)}(x_0)\right) = a_{N-1}. \quad (7.11e)$$

Definindo inicialmente  $y_1(x) = y(x)$ , podemos escrever:

$$y'_1(x) = y_2(x) \quad (7.12a)$$

$$y'_2(x) = y_3(x) \quad (7.12b)$$

$$y'_3(x) = y_4(x) \quad (7.12c)$$

$$\vdots$$

$$y'_{N-1}(x) = y_N(x), \quad (7.12d)$$

finalmente, fazendo uso de (7.11a),

$$y'_N(x) = f\left(x, y_1(x), y_2(x), \dots, y_N(x)\right), \quad (7.12e)$$

com as condições iniciais (7.11b–e) escritas

$$g_0\left(y_1(x_0), y_2(x_0), \dots, y_N(x_0)\right) = a_0 \quad (7.12f)$$

$$g_1(y_1(x_0), y_2(x_0), \dots, y_N(x_0)) = a_1 \quad (7.12g)$$

$$\vdots \quad \vdots$$

$$g_{N-1}(y_1(x_0), y_2(x_0), \dots, y_N(x_0)) = a_{N-1}. \quad (7.12h)$$

Ou seja, ao invés procurarmos uma forma do método de Runge-Kutta para resolver 1 ODE de ordem  $N$ , o que iremos fazer é resolver  $N$  equações de ordem 1.

Quando o PVI for composto por mais de uma ODE de diferentes ordens, busca-se reduzir este sistema sempre a um sistema de primeira ordem. Neste caso, ao invés de somente uma equação de primeira ordem do tipo (7.12e), teremos um sistema de  $N$  equações do tipo:

$$y'_1 = f_1(x, y_1, y_2, \dots, y_n) \quad (7.13a)$$

$$y'_2 = f_2(x, y_1, y_2, \dots, y_n) \quad (7.13b)$$

$$\vdots \quad \vdots$$

$$y'_N = f_N(x, y_1, y_2, \dots, y_n), \quad (7.13c)$$

onde neste sistema já estão incluídas as equações auxiliares (7.12a-d). É muitas vezes conveniente pensar este sistema na forma vetorial,

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \quad (7.13d)$$

onde  $\mathbf{y}$  e  $\mathbf{f}$  são vetores com  $N$  componentes cada.

A subrotina rk4 (programa 7.1) implementa o Método de Runge-Kutta de quarta ordem, dado pelo algoritmo 7.3, em Fortran 95. Nota-se que a subrotina resolve um sistema de EDO's de primeira ordem do tipo (7.13a-c) ou (7.13d). O programador deve fornecer à subrotina o valor da variável independente  $x$ , o valor do passo  $h$  e, no vetor  $y$ , os valores das soluções no ponto  $x$ . A rotina retorna com a solução numérica do sistema de EDO's no vetor  $ysai$ , o qual pode ser o próprio vetor  $y$ , no ponto  $x + h$  posterior. A subrotina não atualiza o valor da variável independente.

**Exemplo 7.1. Movimento harmônico amortecido.** Suponhamos um corpo de massa  $m$  pendurado do teto por uma mola que exerce uma força restauradora  $f_R = -ky$ , oscilando sob a ação da gravidade, mas imerso em um fluido viscoso tal que a força de resistência à passagem do corpo seja proporcional ao quadrado da velocidade do mesmo,  $f_v = Cv^2$ . Este problema pode ser escrito na forma de um PVI como:

$$\ddot{y} = -g - \frac{k}{m}y - \frac{C}{m}\dot{y}|\dot{y}|, \\ y(0) = y_0, \quad \dot{y}(0) = v_0.$$

Definindo  $y_1(t) = y(t)$  e  $\dot{y}_1(t) = y_2(t)$ , o PVI pode ser escrito a partir de (7.12) como:

$$\dot{y}_1 = y_2 \\ \dot{y}_2 = -g - \frac{k}{m}y_1 - \frac{C}{m}y_2|y_2|, \\ y_1(0) = y_0, \quad y_2(0) = v_0,$$

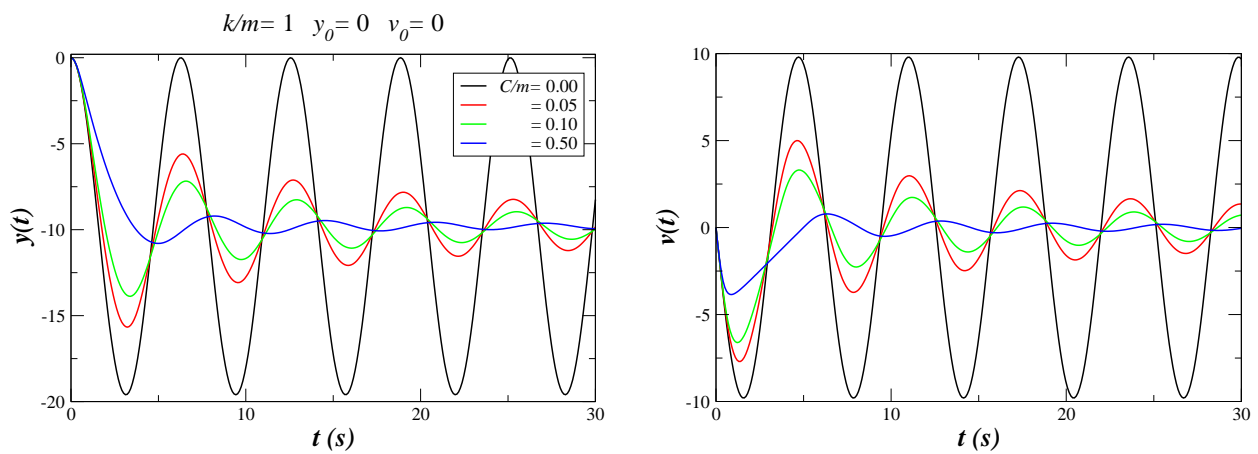
ou seja, em vez de resolvermos 1 equação de 2ª ordem, vamos resolver 2 equações de 1ª ordem. Resultados deste PVI são mostrados em ambos os painéis da figura 7.3, os quais mostram a evolução temporal da posição ( $y(t)$ ) e da velocidade ( $v(t)$ ) do oscilador harmônico. Quando  $C \neq 0$  pode-se observar claramente o amortecimento na oscilação. O programa que gerou os dados para os gráficos da figura 7.3 pode ser obtido em [www.ufpel.edu.br/~rudi/grad/ModComp/Progs/Mov\\_Har\\_Amor.f90](http://www.ufpel.edu.br/~rudi/grad/ModComp/Progs/Mov_Har_Amor.f90).

**Listing 7.1:** Resolve um Problema de Valor Inicial usando o Método de Runge-Kutta de quarta ordem.

```

! ***** SUBROTINA RK4 *****
! Resolve um Problema de Valor Inicial pelo Método de Runge-Kutta
! de quarta ordem com passo fixo.
! Dados o vetor y(:) que contém as variáveis e o vetor das derivadas
! dydx(:) no ponto x, a rotina invoca a subrotina derivs(y,x,dydx)
! que será usada para avançar o vetor das soluções ysai(:) até o
! ponto x + h.
!
! Argumentos de entrada:
! x:      Ponto inicial do intervalo.
! y:      Vetor de forma assumida contendo as soluções do PVI no ponto x.
! h:      Tamanho do passo.
! derivs: Subrotina que calcula as derivadas dydx no ponto x.
! Argumento de saída:
! ysai:   Vetor de forma assumida contendo as soluções do PVI no ponto x + h.
!         O vetor ysai pode ser o próprio vetor y.
!
! Autor: Rudi Gaelzer, IF-UFRGS.
! Data: Julho/2010.
! Obs: Baseada na subrotina RK4 do Numerical Recipes.
!
subroutine rk4(x,y,h,ysai,derivs)
real(kind= dp), intent(in) :: x, h
real(kind= dp), dimension(:), intent(in) :: y
real(kind= dp), dimension(:), intent(out) :: ysai
INTERFACE
  subroutine derivs(x,y,dydx)
  use Modelos_Computacionais_Dados
  real(kind= dp), intent(in) :: x
  real(kind= dp), dimension(:), intent(in) :: y
  real(kind= dp), dimension(:), intent(out) :: dydx
  end subroutine derivs
END INTERFACE
real(kind= dp) :: h6, hh, xh
real(kind= dp), dimension(size(y)) :: dydx, dym, dyt, yt
!
call verifica_tamanho(size(y),size(ysai),'rk4')
hh= h*0.5_dp
h6= h/6.0_dp
call derivs(x,y,dydx)
xh= x + hh
yt= y + hh*dydx
call derivs(xh,yt,dyt)
yt= y + hh*dyt
call derivs(xh,yt,dym)
yt= y + h*dym
dym= dyt + dym
call derivs(x+h,yt,dyt)
ysai= y + h6*(dydx + dyt + 2.0_dp*dym)
return
end subroutine rk4

```



**Figura 7.3:** Soluções numéricas do PVI do exemplo 7.1 utilizando a rotina *rk4*. Esquerda:  $y(t) \times t$ . Direita:  $v(t) \times t$ .



## OUTROS MÉTODOS

Neste capítulo serão abordados alguns métodos computacionais que são úteis em diversas aplicações distintas, mas que não se enquadram nos assuntos discutidos nos capítulos anteriores.

### CLASSIFICAÇÃO (*sorting*)

Em muitas situações, o programador está de posse de um conjunto de objetos de dados que não possui uma organização ou ordenamento evidente. Neste caso, com frequência é necessário ou desejável organizar esses dados a partir de algum critério definido. Por exemplo, dado um vetor contendo quantidades numéricas que se distribuem de forma aleatória ou de uma forma não pré-determinada ao longo do vetor, deseja-se então organizar esses dados em ordem crescente ou decrescente. Este processo de organização de objetos de dados de acordo com algum critério é denominado *classificação* (*sorting*).

Existem diversos algoritmos para implementar *sorting*. Dado um conjunto de  $N$  objetos de dados, os algoritmos mais eficientes que realizam *sorting* irão demandar um número máximo da ordem de  $N \log_2 N$  operações de ponto flutuante para realizar a classificação dos dados. Dois dos algoritmos mais eficientes conhecidos são o *quicksort* e o *heapsort*. Aqui será discutido o algoritmo de *inserção direta* (*straight insertion*), o qual é o mais simples de ser implementado, mas pode demandar até  $N^2$  operações para a sua realização.

### INSERÇÃO DIRETA (*straight insertion*)

O algoritmo de inserção direta, como mencionado, pode demandar até  $N^2$  operações para realizar o *sorting*, mas, em compensação, é o mais simples de ser implementado.

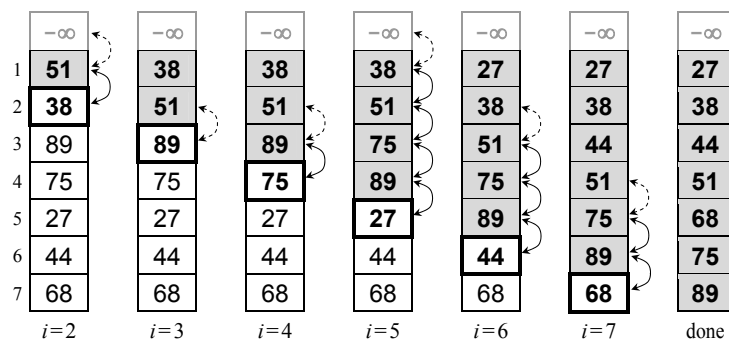
Imagine um baralho de cartas embaralhadas. Para organizar as cartas em ordem crescente, por exemplo, primeiro organiza-se as mesmas em uma linha; então compara-se a segunda com a primeira e organiza-se as mesmas na ordem escolhida. Depois compara-se a terceira com as duas primeiras e realiza-se novamente a organização entre as três. Este processo é então repetido até o final do baralho.

A figura 8.1 exemplifica a aplicação do algoritmo para um vetor contendo números inteiros positivos. Neste exemplo, um conjunto de 7 números inteiros necessitou de um total de 11 permutações para a classificação, mas poderiam ser necessárias até  $7^2 = 49$  permutações.

O algoritmo 8.1 apresenta uma implementação da inserção direta para organizar um vetor em ordem crescente. Os dados são de algum tipo numérico ou de algum outro tipo para o qual possam ser definidas as expressões relacionais.<sup>1</sup> O algoritmo permite a classificação de uma seção do vetor ou do vetor inteiro, sendo que  $\inf(A)$  e  $\sup(A)$  são, respectivamente, os limites inferior e superior do vetor  $A$ .

A sub-rotina `sort_pick` (programa 8.1) implementa o algoritmo 8.1 em Fortran para classificar todo o vetor real `arr` em ordem crescente. Na saída, o vetor é retornado à unidade invocadora com os elementos de `arr` classificados.

<sup>1</sup>Isto é, para o qual expressões como  $a > b$  ( $a, b \in A$ ) façam sentido.



**Figura 8.1:** Aplicação do algoritmo de inserção direta para organizar um vetor de números inteiros em ordem crescente.

---

**Algoritmo 8.1** Método da inserção direta para classificar um vetor de números em ordem crescente.

---

**Entrada:** Vetor  $A$ ; inteiros  $p$  e  $r$ , tais que  $\inf(A) \leq p \leq r \leq \sup(A)$ .

**Saída:** Vetor  $A$ , com os elementos da seção  $A(p:r)$  classificados. Os demais elementos de  $A$  não são alterados.

**Para**  $i = p + 1, p + 2, \dots, r$  **faça**

$temp = A(i)$

$j = i$

**Enquanto que**  $(j > p \text{ e } A(j-1) > temp)$  **faça**

$A(j) = A(j-1)$

$j = j - 1$

**Fim**

$A(j) = temp$

**Fim Laço**

---

**Listing 8.1:** Sub-rotina `sort_pick`: recebe como entrada um vetor real `arr` e usa inserção direta para classificar o vetor em ordem crescente. Na saída, o vetor `arr` é retornado com seus elementos classificados.

```

subroutine straight_sort(arr)
implicit none
real, dimension(:), intent(inout) :: arr
integer :: i, j, n
real :: a
n= size(arr)
do j= 2, n
  a= arr(j)
  do i= j - 1, 1, -1
    if (arr(i) <= a) exit
    arr(i + 1)= arr(i)
  end do
  arr(i + 1)= a
end do
return
end subroutine straight_sort

```

# REFERÊNCIAS BIBLIOGRÁFICAS

- [1] Intel® Fortran Compiler 19.0 Developer Guide and Reference. Acesso em setembro de 2019. URL: <https://software.intel.com/en-us/fortran-compiler-developer-guide-and-reference-language-reference>.
- [2] David M. Young and Robert Todd Gregory. *A Survey of Numerical Mathematics*, volume I. Dover, New York, 1988. 548 + x pp.
- [3] D. Goldberg. What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23:5–48, March 1991. doi:10.1145/103162.103163.
- [4] John R. Rice. *Numerical Methods, Software, and Analysis*. McGraw-Hill, New York, 1983. 483 + xii pp.
- [5] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in FORTRAN: The Art of Scientific Computing*. Cambridge, New York, second edition, 1992. 999 pp.
- [6] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in Fortran 90: The Art of parallel Scientific Computing*, volume 2 of *Fortran Numerical Recipes*. Cambridge, New York, second edition, 1997. 552 pp.
- [7] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, third edition, 2007. 1235 + xxi pp. URL: <http://books.google.com.br/books?id=1aA0dzK3FegC>.
- [8] R. Roy, F. W. J. Olver, R. A. Askey, and R. Wong. Algebraic and Analytic Methods. In Olver et al. [16], chapter 1, page 1–39. URL: <https://dlmf.nist.gov/1>.
- [9] R. Roy and F. W. J. Olver. Elementary Functions. In Olver et al. [16], chapter 4, page 103–134. URL: <https://dlmf.nist.gov/4>.
- [10] G. B. Arfken, H. J. Weber, and F. E. Harris. *Mathematical Methods for Physicists: A Comprehensive Guide*. Elsevier Science, New York, seventh edition, 2013. 1205 + xiii pp. URL: [https://books.google.com.br/books?id=qLFo\\_Z-PoGIC](https://books.google.com.br/books?id=qLFo_Z-PoGIC).
- [11] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Texts in Applied Mathematics. Springer, New York, third edition, 2013. URL: <https://books.google.com.br/books?id=fh3vBwAAQBAJ>.
- [12] G. Dahlquist and A. Bjorck. *Numerical Methods in Scientific Computing*. Other Titles in Applied Mathematics. SIAM, Philadelphia, 2008. URL: <https://books.google.com.br/books?id=1nwbGQN4jcC>.
- [13] R. L. Burden and J. D. Faires. *Numerical Analysis*. Cengage Learning, Boston, ninth edition, 2010. URL: <https://books.google.com.br/books?id=Dbw8AAAAQBAJ>.
- [14] Rudi Gaelzer. Física-Matemática. Porto Alegre, July 2019. URL: <http://professor.ufrgs.br/rgaelzer/pages/fismat-apost>.
- [15] Amparo Gil, Javier Segura, and Nico M. Temme. *Numerical Methods for Special Functions*. SIAM, Philadelphia, 2007. 417 + xiv pp. URL: <http://functions.djvu>.

- [16] Frank W. J. Olver, Daniel W. Lozier, Ronald F. Boisvert, and Charles W. Clark, editors. *NIST Handbook of Mathematical Functions*. Cambridge, New York, 2010. URL: <http://dlmf.nist.gov/>.
- [17] T. H. Koornwinder, R. Wong, R. Koekoek, and R. F. Swarttouw. Orthogonal Polynomials. In Olver et al. [16], chapter 18, page 435–484. URL: <https://dlmf.nist.gov/18>.
- [18] Annie Cuyt, Vigdis Brevik Petersen, Brigitte Verdonk, Haakon Waadeland, and William B. Jones. *Handbook of Continued Fractions for Special Functions*. Springer, New York, 2008. 431 + xvi pp.
- [19] G. Blanch. Numerical Evaluation of Continued Fractions. *SIAM Review*, 6(4):383–421, 1964. doi:10.1137/1006092.
- [20] Tao Pang. *An Introduction to Computational Physics*. Cambridge University Press, New York, second edition, 2006. 385 + xvi pp.
- [21] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions*. Dover, New York, 1970.
- [22] N. M. Temme. Numerical Methods. In Olver et al. [16], chapter 3, page 71–101. URL: <https://dlmf.nist.gov/3>.
- [23] Rubin H. Landau and Manuel José Páez Mejiá. *Computational Physics. Problem Solving with Computers*. John Wiley & Sons, New York, 1997. 511 + xviii pp.
- [24] Joe D. Hoffman. *Numerical Methods for Engineers and Scientists*. Marcel Dekker, New York, second edition, 2001. 823 + xi pp.
- [25] P. L. DeVries. *A First Course in Computational Physics*. John Wiley & Sons, New York, 1994.
- [26] S. D. Conte and C de Boor. *Elementary Numerical Analysis. An Algorithmic Approach*. International series in pure and applied mathematics. McGraw-Hill, New York, third edition, 1980. 432 + xii pp.
- [27] D. E. Müller. A method of solving algebraic equations using automatic computer. *Mathematical Tables and Other Aids to Computation*, 10:208–215, 1956.