Fundamentos de Cálculo Numérico

Notas de aula PPGEM

Prof. Eduardo Lenz Cardoso

Conteúdo

Fundamentos de Cálculo Numérico

1. Introdução

Este texto compila minhas notas de aula associadas a métodos numéricos, principalmente no contexto da disciplina de Fundamentos de Matemática do PPGEM/UDESC.

Neste material iremos revisar/apresentar os conceitos básicos de cálculo numérico. Para isto, o leitor deve primeiro entender como os números são representados internamente em um computador, para então compreender a natureza dos erros de aproximação que são inerentes a esta representação. De posse destes conceitos básicos, iremos apresentar, de forma simplificada, uma representação para algoritmos para, depois, estudarmos os métodos numéricos de interesse.

É importante salientar que na norma brasileira a vírgula é utilizada para separar a parte inteira de um número real da parte fracionária (1,23), sendo que o ponto é o separador de milhar. No entanto, em países de língua inglesa, é o ponto que deve ser utilizado para separar a parte inteira da parte fracionária (1.23). Neste texto iremos utilizar o procedimento internacional, que é mais comum, principalmente se formos utilizar alguma linguagem de programação.

Outras definições importantes para compreender o texto são:

- Matrizes são indicadas em letras maiúsculas e em negrito, $\mathbf{A}$. Um valor em uma linha i e coluna j é indicada como a_{ij}

$$\mathbf{A}_{m\times n} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{12} & \dots & a_{nm} \end{bmatrix}$$

- Vetores são indicados em minúsculo e em negrito, $\mathbf{v}$. Um valor em uma linha i é indicado como v_i

$$\mathbf{v}_m = \begin{Bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{Bmatrix}$$

Por definição, vetores serão sempre no formato "coluna", isto é, tem dimensão de m linhas por apenas uma coluna: $m \times 1$.

- Uma notação alternativa para vetor, que será adotada por ser mais compacta (ocupar apenas uma linha de texto) é

$$\mathbf{v}_m = \begin{pmatrix} v_1 & v_2 & \dots & v_m \end{pmatrix}$$

sendo que o vetor continua sendo "coluna", só que mostrado em uma linha.

- A multiplicação de dois valores escalares será indicada, em algumas situações, por $\cdot$ ao invés de $*$, por uma questão de estética.
- O produto interno entre dois vetores $\mathbf{u}$ e $\mathbf{v}$ será indicado sempre por $\langle \mathbf{u}, \mathbf{v} \rangle$;

2. Conceitos Básicos de Cálculo Numérico

2.1 Representação de números naturais em diferentes bases

Um número natural como 215 é representado por valores que indicam a quantidade de (neste caso) centenas + dezenas + unidades, sendo que neste caso (base 10) cada uma das posições pode assumir valores entre $[0,9]$ (dez possibilidades). Diferentes bases seguem a mesma lógica, mas alteram o número valores que podem ser utilizado em cada posição. As bases mais comuns são

Base (b)	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
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

e, de forma geral, podemos representar um número natural por uma combinação linear

$$N = a_n b^n + a_{n-1} b^{n-1} + ... + a_1 b + a_0,$$

ou na forma compacta

$$(a_n a_{n-1} a_{n-2} .... a_1 a_0)_b$$

como por exemplo

$$(215)_{10} = 2 \cdot 10^2 + 1 \cdot 10 + 5$$

ou

$$(0111)_2 = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0.$$

Partindo-se de um número natural $(M)_{10}$, podemos obter a representação em qualquer base, bastando para isto realizarmos uma série de divisões pelo valor da base desejada. Por exemplo:

$$\begin{array}{cccccccc} 110 & \lfloor 2 & & & & & & \\ 0 & 55 & \lfloor 2 & & & & & \\ & 1 & 27 & \lfloor 2 & & & & \\ & & 1 & 13 & \lfloor 2 & & & \\ & & & 1 & 6 & \lfloor 2 & & \\ & & & & 0 & 3 & \lfloor 2 & \\ & & & & & 1 & 1 & \end{array}$$

e, portanto, $(110)_{10} = (1101110)_2$, ou seja, os restos das divisões irão indicar os coeficientes $a_0, a_1 .... a_{n-1}, a_n$ na sequência em que aparecerem. O último valor é o divisor final. O mesmo número na base 8 seria descrito por

$$\begin{array}{ccc} 110 & \lfloor 8 & \\ 6 & 13 & \lfloor 8 \\ & 5 & 1 \end{array}$$

ou seja, $(110)_{10} = (156)_8$. Finalmente, embora seja óbvio,

$$\begin{array}{ccc} 110 & \lfloor 10 & \\ 0 & 11 & \lfloor 10 \\ & 1 & 1 \end{array} .$$

2.2 Representação da parte fracionária de um número real

Um número real X pode ser visto como uma parte inteira X_i somado a uma parte fracionária X_f, tal que $X_f = X - X_i$. Desta forma, X_f estará sempre na faixa $[0,1)$ e poderá ser descrito como uma combinação linear

$$X_f = a_1 \cdot 2^{-1} + a_2 \cdot 2^{-2} + ... + a_n \cdot 2^{-n}$$

onde deve-se observar que esta série é convergente para 1 quando o número de termos tender a infinito, pois

$$\sum_{n=1}^{\infty} \frac{1}{2^n} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... \rightarrow 1.0$$

ou seja, quanto maior o número de bits (termos de base), maior a precisão da representação. A forma de converter uma parcela fracionária escrita em base decimal para uma representação em base 2 é o inverso da parte inteira, pois aqui iremos multiplicar a parte

fracionária por 2. Por exemplo 0.7 é representado por

$$\begin{aligned} 0.7 \cdot 2 &= 1.4 \\ 0.4 \cdot 2 &= 0.8 \\ 0.8 \cdot 2 &= 1.6 \\ 0.6 \cdot 2 &= 1.2 \\ 0.2 \cdot 2 &= 0.4 \\ 0.4 \cdot 2 &= 0.8 \\ 0.8 \cdot 2 &= 1.6 \\ &.... \end{aligned}$$

ou $(1011001.....)_2$. Observem que a representação não tem fim, estando limitada pelo número de bits que são utilizados para representar o número. Por exemplo, se utilizarmos 7 bits, obteremos o número

$$1 \cdot 2^{-1} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-7} = 0.6953125.$$

2.3 Representação de Números em Computadores

Um número inteiro pode ser representado exatamente por uma representação binária utilizada em computadores, desde que sejam utilizados o número suficiente de bits. Uma questão importante é a representação do sinal, que não é necessária em um número natural. Para isto, é reservado o bit (posição) mais a esquerda (ou mais significativo) da representação binária. Se este bit for 0 o número é dito positivo e se for 1 o número é dito negativo, pois $-1^0 = 1$ e $-1^1 = -1$. Com isto, restam $n-1$ bits para a representação do número. Por exemplo

$$(00001)_2 = -1^0 (0001)_2 = +2^0 = +(1)_{10}$$

e

$$(10101)_2 = -(0101)_2 = -1^1 \left(2^0 + 2^2\right) = -(5)_{10}.$$

Assim, se um inteiro for representado com 8 bits, teremos apenas 7 para representar efetivamente os valores e um para o sinal. Com isto estaremos trabalhando na faixa $\pm[127]$.

Por sua vez, um numero real, também chamado de ponto flutuante (float) , pode ser representado de diferentes formas. A mais utilizada é a definida pela norma IEEE 754 (1985, com revisão em 2008) onde

$$N = (-1)^S \cdot (1 + Mantissa) \cdot 2^{Expoente - PESO}$$

onde a Mantissa $\in [0,1)$ indica a parte fracionária do número 1, S indica o sinal da Mantissa, Expoente é o expoente da base 2 e PESO é um fator que define o sinal do expoente. O uso do PESO, embora pareça uma complicação desnecessária, permite a economia de um bit de sinal para aumentar a precisão da representação do expoente.

■ **Exemplo 2.1** Em precisão simples, utilizamos 32 bits para descrever um número.

Destes, 1 bit (S) é utilizado para descrever o sinal da Mantissa, 8 bits são utilizados para descrever o Expoente e 23 bits são utilizados para descrever a Mantissa. Como utilizamos 8 bits para o expoente, então podemos (idealmente) obter valores na faixa $(00000000)_2 = (0)_{10}$ e $(11111111)_2 = (255)$ que são todos positivos. Neste caso, o valor *PESO* definido na norma é 127, pois se Expoente=$(00000000)_2$, então estaremos trabalhando com $2^{0-127} = 2^{-127}$ e se Expoente= $(11111111)_2 = (255)$ então $2^{255-127} = 2^{128}$.

■

■ **Exemplo 2.2** Em precisão dupla, *double*, utilizamos 64 bits para representar um número em ponto flutuante. Neste caso, continuamos utilizando 1 bit para o sinal da Mantissa, mas agora utilizamos 11 bits para o expoente e 52 bits para a Mantissa.

Da mesma forma, em precisão dupla, teremos um $PESO = 1023$. No entanto, a norma reservou os valores extremos da faixa para representação de números especiais e, portanto, temos um limite real de $1 \leq Expoente \leq 254$ o que nos leva aos extremos -126 e 127 para o expoente efetivo. ■

■ **Exemplo 2.3** O decimal 0.1 não tem representação exata. A representação em precisão simples é

$$0\,01111011\,10011001100110011001101$$

que, na verdade, corresponde ao decimal

$$0.100000001490116119384765625.$$

■

■ **Exemplo 2.4** Em qualquer caso (precisão adotada), a Mantissa representa a parte fracionária a direita do 1 e, portanto, segue a notação apresentada na Seção (2.2). Considere a representação binária em precisão simples:

$$1\,01001011\,10100101101100011101011$$

que equivale a

$$N = (-1)^1 1.(10100101101100011101011)_2 \cdot 2^{(01001011)_2 - 127}$$

com Expoente

$$Expoente = 2^0 + 2^1 + 2^3 + 2^6 = 75$$

e Mantissa

$$Mantissa = 2^{-1} + 2^{-3} + 2^{-6} + 2^{-8} + 2^{-9} + 2^{-11} + 2^{-12} + 2^{-16} + 2^{-17} + 2^{-18} + 2^{-20} + 2^{-22} + 2^{-23}$$

ou 0.6472448. Portanto

$$N = -1.6472448 \cdot 2^{-52} = -3.6576182 \times 10^{-16}.$$

Aqui é importante salientarmos que todas as operações matemáticas foram realizadas retendo apenas os 7 dígitos significativos que são "garantidos"pela precisão simples. No entanto, se cálculos como 2^{-23} forem realizados em uma precisão maior, poderemos obter mais dígitos no resultado (que não serão significativos). O resultado, neste caso, seria $N = -3.65761823286191319580529 \times 10^{-16}$ ■

Exercício 2.1 Obtenha o valor da representação binária (precisão simples)

$$0\,10001000\,00100101100000011101011$$

resp: Expoente = 136, Mantissa = 0.14651239, resultado = 587.01434. ■

Assim, qualquer número real terá um limite de representação que depende da precisão utilizada. Em precisão simples observamos que o maior número que pode ser representado (em módulo) será $1.99999988 \cdot 2^{127} = 3.402823466385289 \times 10^{38}$, portando, se um número for maior do que este valor, será interpretado como ∞ (*overflow*). Da mesma forma, o menor número que pode ser representado será $1.0 \cdot 2^{-126} = 1.175494350822287 \times 10^{-38}$. Assim, se $-1.175494350822287 \times 10^{-38} < x < 1.175494350822287 \times 10^{-38}$, então x será considerado como 0.0 (*underflow*).

■ **Exemplo 2.5** O cálculo da hipotenusa de um triângulo com catetos x e y é dado por $h = \sqrt{x^2 + y^2}$. Se x e/ou y forem números muito grandes, então existe a possibilidade de o resultado ser *overflow*, mesmo que o valor esperado seja menor do que o limite da representação. Como exemplo, se $x = y = 0.5DMAX$, onde $DMAX$ é o maior número que pode ser representado, então o cálculo direto resulta em *overflow*,

Geralmente isto é evitado com a seguinte estratégia

- $m = \max(|x|, |y|)$;
- $n = \min(|x|, |y|)$;
- $r = n/m$;
- $h = m\sqrt{1 + r^2}$.

que permite obter o resultado correto sem *overflow*. ■

2.4 Valores especiais reservados pela norma

Conforme comentado anteriormente, alguns valores são reservados pela norma. Estes valores são $+\infty$, $-\infty$, *NaN* (*not a number*) e ± 0, pois a mantissa define somente a parte fracionária, sendo que o número 0.0 não pode ser obtido diretamente na representação definida pela norma.

Valor	Representação binária		
∞	0	1111 1111 1	0000 0000 0000 0000 0000 000
$-\infty$	1	1111 1111 1	0000 0000 0000 0000 0000 000
NaN	1	1111 1111 1	1000 0000 0000 0000 0000 000
$+0$	0	0000 0000 1	0000 0000 0000 0000 0000 000
-0	1	0000 0000 1	0000 0000 0000 0000 0000 000

Assim, temos que $\frac{1}{+0} = \infty$, $\frac{1}{-0} = -\infty$, $\log(0) = -\infty$ e $\log(x) = NaN$, $x < 0$ (em teoria, pois isto depende do compilador). Por definição da norma, as operações que definem *NaN* são: $0 * \infty$, $0/0$, ∞/∞, $\infty + (-\infty)$, resto das operações $x/0$ e ∞/x e raiz de um número real negativo. **Toda a operação que envolver um *NaN* irá resultar em *NaN*.**

2.5 Epsilon da Máquina

Outro conceito importante em cálculo numérico é o de *epsilon* da máquina, que é o menor número que se for somado a unidade não será perceptível (não altera o valor da operação). Para entendermos melhor este comportamento, vamos lembrar que em precisão simples o número 1.0 é representado por o por

$$(1.0)_{10} = 1.(00000000000000000000000)_2 \cdot 2^0$$

e o próximo número será representado por

$$(1 + 1.1920928955078125 \times 10^{-7})_{10} = 1.(00000000000000000000001)_2 \cdot 2^0$$

ou seja, mudamos o bit menos significativo da Mantissa. No entanto, devemos considerar um número menor do que este, para que seja imperceptível. Desta forma, a definição da norma é

$$\varepsilon_m = \frac{1}{2}2^{-\text{Número de bits da Mantissa}}$$

tal que

$$\left|\frac{f_l(x)-x}{x}\right| \leq \varepsilon_m$$

em que $f_l(x)$ é a representação em ponto flutuante de um dado número real x. Desta forma, em precisão simples teremos $\varepsilon_m = 0.5 \cdot 2^{-23} = 5.96 \times 10^{-8}$ e, em precisão dupla, aproximadamente 1.1×10^{-16}. Este conceito é de fundamental importância, devendo ser sempre cuidadosamente verificado em cálculos que utilizem pequenos incrementos.

■ **Exemplo 2.6** Ao aproximarmos a derivada de uma função em torno de um ponto x, podemos utilizar

$$\frac{df(x)}{dx} \backsimeq \frac{f(x+\delta)-f(x)}{\delta}.$$

Essa derivada irá retornar zero se $\delta < \varepsilon_m$, pois a subtração não fará sentido $(x+\varepsilon = x)$.

Considerando $x = 0.2$. A representação binária do número 2, em precisão dupla, é exata e igual a

$$0\,10000000000000000000000000000000000000000000000000000000000000000$$

a alteração do último bit (mais a direita) faz com que tenhamos o número

$$2.00000000000000044408920985006$$

ou seja, qualquer adição entre $[0, 4.4 \times 10^{-16}]$ será interpretada como o mesmo número pelo computador e $f(x) - f(x+\delta) = 0$. É importante salientar que essas diferenças dependem do número que estamos tentando aproxima, sendo que a referência para a definição do ε é a perturbação da unidade. ■

2.6 Precisão e Exatidão/Acurácia

Dois conceitos fundamentais em cálculo numérico são o de precisão e o de Exatidão, ou acurácia (do inglês *accuracy*).

Definição 2.6.1 Precisão

Refere-se ao quão próximo um número representado pelo computador (ponto flutuante) representa o número que ele ambiciona representar. A precisão de um número é governada pelo número de dígitos empregados na representação. Assim, a constante π será representada com maior precisão utilizando precisão dupla do que precisão simples.

Definição 2.6.2 Exatidão

Refere-se a quão próximo um número representado pelo computador (como resultado de uma série de operações) está do valor correto do número que ele almeja representar. A acurácia é governada pelos erros (de truncamento e arredondamento) do método numérico empregado.

■ **Exemplo 2.7** Sejam dois valores em ponto flutuante (precisão dupla) $\pi_1 = 3.141604958$ e $\pi_2 = 3.1415809485$. Ambos almejam representar o número π. Neste caso, o número π_2 possui maior acurácia do que π_1, embora ambos possuam a mesma precisão (pois estão sendo representados com o mesmo número de bits.

■

Da mesma forma, um conceito importante é o de quantos dígitos são realmente significativos na representação decimal que desejamos obter.

Definição 2.6.3 Precisão: número de dígitos significativos

Outra maneira de pensar no conceito de precisão é que não conseguimos representar um número real de forma exata no computador (pois um número real tem precisão infinita). Desta forma, para deixar muito claro que não estamos lidando com números reais mas sim com uma aproximação, chamamos os ponto flutuante, ou *floating point*. Assim, um número como $0.3333\bar{3}$, onde o 3 se repete infinitas vezes, precisaria de um número infinito de termos de base 2^{-n}, para ser representado corretamente, o que sabemos que é inviável.

Considerando que usamos uma base 2, se um número for representado com b bits de Mantissa, teremos que o número de dígitos significativos da representação é dado por

$$p = 1 + (b-1)\log(2).$$

■ **Exemplo 2.8** Se estivermos trabalhando em precisão simples, $b = 23$, então

$$p = 1 + (22)0.3010 = 7.622$$

ou seja, 7 dígitos. Para precisão dupla, $b = 52$,

$$p = 1 + (51)0.3010 = 16.351$$

ou 16 dígitos.

■

2.7 Propagação de Erros

Ao realizarmos uma ou mais operações matemáticas com pontos flutuantes em um computador, iremos sempre ter um erro de representação em cada um dos números, causado pela imprecisão na representação binária. Outra fonte de erros está na própria natureza dos algoritmos numéricos iterativos ou até mesmo na entrada dos dados de um problema. De forma geral, podemos definir os valores exatos de uma dada sequência de operações matemáticas como x e y e seus valores numéricos como $\tilde{x}$ e $\tilde{y}$. Assim, podemos definir os erros absolutos como

$$\begin{aligned} E_x &= x - \tilde{x} \\ E_y &= y - \tilde{y} \end{aligned}$$

onde para simplificar a notação assumimos que x e y são maiores do que os valores aproximados. Assim, teremos que $x = E_x + \tilde{x}$ e $y = E_y + \tilde{y}$ e, se estes valores forem utilizados para realizarmos outros cálculos, podemos agora estimar a propagação destes erros absolutos pois:

Definição 2.7.1 Erro absoluto na soma

$$x+y=(E_x+\tilde{x})+(E_y+\tilde{y})=(\tilde{x}+\tilde{y})+(E_x+E_y)$$

tal que

$$E_+ = E_x + E_y$$

Definição 2.7.2 Erro absoluto na diferença

$$x-y=(E_x+\tilde{x})-(E_y+\tilde{y})=(\tilde{x}-\tilde{y})+(E_x-E_y)$$

tal que

$$E_- = E_x - E_y.$$

Definição 2.7.3 Erro absoluto na multiplicação

$$xy=(E_x+\tilde{x})(E_y+\tilde{y})=\tilde{x}\tilde{y}+E_xE_y+E_x\tilde{y}+E_y\tilde{x}$$

Assim,

$$E_* = E_xE_y + E_x\tilde{y} + E_y\tilde{x}$$

e, como o primeiro termo é de segunda ordem

$$E_* \approx E_x\tilde{y} + E_y\tilde{x}.$$

Definição 2.7.4 Erro absoluto na divisão

$$\frac{x}{y}=\frac{(E_x+\tilde{x})}{(E_y+\tilde{y})}=\frac{(E_x+\tilde{x})}{\tilde{y}\left(1+\frac{E_y}{\tilde{y}}\right)}$$

e, como $(1+z)^{-1}=1-z+z^2-z^3+z^4-...$, podemos escrever

$$\frac{x}{y}=\frac{(E_x+\tilde{x})}{(E_y+\tilde{y})}=\frac{(E_x+\tilde{x})}{\tilde{y}\left(1+\frac{E_y}{\tilde{y}}\right)}=\frac{(E_x+\tilde{x})}{\tilde{y}}\left(1-\frac{E_y}{\tilde{y}}+\frac{E_y^2}{\tilde{y}^2}-\frac{E_y^3}{\tilde{y}^3}+....\right)$$

e, como E_y é pequeno, podemos abandonar os termos de alta ordem (potências de E_y) tal que

$$\frac{x}{y}\approx\frac{(E_x+\tilde{x})}{\tilde{y}}\left(1-\frac{E_y}{\tilde{y}}\right)=\frac{(E_x+\tilde{x})}{\tilde{y}}-\frac{(E_x+\tilde{x})E_y}{\tilde{y}^2}=\frac{E_x}{\tilde{y}}+\frac{\tilde{x}}{\tilde{y}}-\frac{E_xE_y}{\tilde{y}^2}-\frac{\tilde{x}E_y}{\tilde{y}^2}$$

e, novamente, podemos descartar o termo que contém E_xE_y, obtendo uma estimativa

$$E_/ \approx \frac{E_x}{\tilde{y}}-\frac{\tilde{x}E_y}{\tilde{y}^2}$$

De posse dos erros absolutos, podemos definir o erro relativo

Definição 2.7.5 Erro Relativo
Definimos o erro relativo de uma operação *op* como a razão entre o erro absoluto e

o valor aproximado obtido no cálculo

$$Er_{op} = \frac{E_{op}}{op}$$

tal que

Definição 2.7.6 Erro relativo na soma

$$Er_{+} = \frac{E_x + E_y}{\tilde{x}+\tilde{y}} = \frac{\tilde{x}}{\tilde{x}+\tilde{y}}\left(\frac{E_x}{\tilde{x}}\right) + \frac{\tilde{y}}{\tilde{x}+\tilde{y}}\left(\frac{E_y}{\tilde{y}}\right)$$

Definição 2.7.7 Erro relativo na diferença

$$Er_{-} = \frac{E_x - E_y}{\tilde{x}-\tilde{y}} = \frac{\tilde{x}}{\tilde{x}-\tilde{y}}\left(\frac{E_x}{\tilde{x}}\right) - \frac{\tilde{y}}{\tilde{x}-\tilde{y}}\left(\frac{E_y}{\tilde{y}}\right)$$

Definição 2.7.8 Erro relativo na multiplicação

$$Er_{*} \approx \frac{E_x\tilde{y} + E_y\tilde{x}}{\tilde{x}\tilde{y}} = \left(\frac{E_x}{\tilde{x}}\right) + \left(\frac{E_y}{\tilde{y}}\right)$$

Definição 2.7.9 Erro relativo na divisão

$$Er_{/} \approx \frac{\frac{E_x}{\tilde{y}} - \frac{\tilde{x}E_y}{\tilde{y}^2}}{\tilde{x}/\tilde{y}} = \frac{E_x\tilde{y} - E_y\tilde{x}}{\tilde{x}\tilde{y}}$$

Assim, se em um dado instante de um cálculo tivermos $x = 100$, $\tilde{x} = 99.99991$, $y = 50$ e $\tilde{y} = 49.99995$, por exemplo, então teremos

$$\begin{aligned} E_x &= 100 - 99.99991 = 9.000000000014552 \times 10^{-5} \\ E_y &= 50 - 49.99995 = 5.000000000165983 \times 10^{-5} \end{aligned}$$

tal que os erros absolutos serão

$$\begin{aligned} E_{+} &= 1.4000000000180535 \times 10^{-4} \\ E_{-} &= 3.999999999848569 \times 10^{-5} \\ E_{*} &= 0.009499995500173258 \\ E_{*} &\approx 0.009499991000173258 \\ E_{/} &\approx -2.0000040006408287 \times 10^{-7} \end{aligned}$$

que devem ser comparados diretamente aos valores aproximados que seriam obtidos sem a propagação deste erro

$$\begin{aligned} \tilde{x}+\tilde{y} &= 149.99986 \pm 1.4000000000180535 \times 10^{-4} \\ \tilde{x}-\tilde{y} &= 49.99996 \pm 3.999999999848569 \times 10^{-5} \\ \tilde{x}\cdot\tilde{y} &= 4999.9905000045 \pm 0.009499991000173258 \\ \frac{\tilde{x}}{\tilde{y}} &= 2.0000002000002 \pm 2.0000040006408287 \times 10^{-7} \end{aligned}$$

sendo que o erro relativo permite uma melhor avaliação da severidade dos erros associados

as estas operações matemáticas básicas, pois

$$\begin{aligned} Er_{+} &= 9.333342044572931 \times 10^{-7} \\ Er_{-} &= 8.000006399702259 \times 10^{-7} \\ Er_{*} &= 1.900001810036381 \times 10^{-6} \\ Er_{/} &= -1.0000019003201259 \times 10^{-7}. \end{aligned}$$

É importante salientar que os computadores e compiladores atuais propagam erro de forma mais lenta do que o visto aqui, pois utilizam um dígito de guarda (definido pela norma), além do fato de que os erros com sinais contrários tendem a se compensar de forma aleatória. **No entanto, devemos sempre lembrar que a medida que o número de operações matemáticas aumenta em um dado procedimento numérico, maior será a diferença entre o valor exato e o obtido.**

2.8 Dicas Importantes

Como as operações com ponto flutuante estão sujeitas a erros, nunca devemos utilizar a comparação direta entre valores (se $x = y$), pois para o computador os números só serão iguais se **todos os bits forem iguais**. Desta forma, o correto é avaliar se a distância entre os números esta abaixo de uma determinada tolerância (se $|x-y| < tol$).

■ **Exemplo 2.9** Se avaliarmos $y = \cos(\pi/3)$ e $z = \sin(\pi/3+\pi/2)$, ambos deveriam dar 0.5. No entanto, tanto a representação de π quando as operações matemáticas utilizadas para o cálculo do cos e do sin fazem com que $y = 0.5 + 1 \times 10^{-15}$ e $z = 0.5 + 3 \times 10^{-15}$, que embora sejam erros absolutos muito pequenos, inviabilizam a comparação direta. Se convertemos esses números para uma representação em precisão dupla (no julia, comando bitstrings), obteremos

$$0.5 = 0011111111100000000000000000000000000000000000000000000000000000,$$

$$y = 0011111111100000000000000000000000000000000000000000000000000001$$

e

$$z = 0011111111100000000000000000000000000000000000000000000000000011$$

de onde podemos notar as diferenças nos bits bem da direita. ■

Outra questão importante é a associatividade da soma e da multiplicação de números reais. Em um número real com precisão infinita, a operação $(x+y)+z$ deve ser igual a $x+(y+z)$. No entanto, devido aos erros de arredondamento da representação de números reais em computadores, esta associatividade pode não se verificar.

■ **Exemplo 2.10** Sejam $x = 10^{30}$, $y = -10^{30}$ e $z = 1$. Se usarmos uma representação com precisão dupla, obteremos $(x+y)+z = 1$ mas $x+(y+z) = 0$. Isso ocorre pois no primeiro caso realizamos a primeira operação entre os parênteses e, como os números tem a mesma ordem de grandeza, $x+y = 0$, como esperado. No segundo caso, a primeira operação é realizada entre dois números com ordens de grandeza completamente diferentes, tal que o resultado é da ordem de grandeza do maior.

Vamos entender do ponto de vista binário. O número 1×10^{30} tem representação (em dupla precisão)

$$0100011000101001001111100101100100111001101000001000110011101010$$

e o número -1×10^{30} só muda o bit de sinal

$$1\,10001100010100100111110010110010011100110100000100011001110101 0$$

tal que subtração será exata. O interessante é que o número binário tem um erro associado a representação do número real "exato", tal que se convertermos do binário para o real mais próximo, veremos que na verdade ambos os números são

$$\pm 1.00000000000000001988462483866 \times 10^{30}$$

e a subtração de dois números iguais e sinal oposto (mesmo com o erro de representação) será zero, tal que $(x+y)+z = 0+1 = 0$. No entanto, se observarmos a representação binária de $y+z$

$$1\,10001100010100100111110010110010011100110100000100011001110101 0$$

que corresponde ao número real (mais próximo)

$$-1.00000000000000001988462483866 \times 10^{30}$$

tal que $y+z = y$ e $x+y = 0$.

■

3. Descrição Básica de um Algoritmo

Um dos objetivos deste estudo é descrever o comportamento de algoritmos, isto é, uma sequência de operações bem definidas e não ambíguas, cada qual podendo ser executada em um tempo finito. A palavra algoritmo é derivada do nome Persa *Muhammad ibn Mũsã Al-Khwãrizmĩ* do século 9. Para isto, iremos definir neste capítulo uma pseudo-linguagem que irá nos ajudar a descrever os passos necessários para a implementação da sequência de operações associada a cada um dos tópicos a serem estudados nestas notas de aula.

3.1 Definição da notação

As operações básicas são descritas por:

Definição 3.1.1 Notação básica

- Variáveis: $a, b, c...$
- Atribuição de valor: $a \leftarrow$ valor
- Vetores e Matrizes: A(i,j) e B(i)
- Cria uma matriz $m \times n$ nula: $\mathbf{A} \leftarrow \mathbf{0}_{m \times n}$
- Cria um vetor $n \times 1$ nulo: $\mathbf{B} \leftarrow \mathbf{0}_n$
- Texto: "Texto"
- Escrever na tela: escrever "texto"
- Sequência de valores: $\{1, ..., n\}$ ou $\{1 : n\}$
- Sequência de valores com passo p: $\{1 : p : n\}$
- Retorna um ou mais valores: return

E operações lógicas e de execução são definidas a seguir.

3.1.1 Estrutura condicional

A estrutura condicional testa se uma condição lógica é verdadeira *true* ou falsa *false*. Se ela for verdadeira, então somente o bloco correspondente é executado. Opcionalmente, pode-se adicionar um bloco final que será executado se nenhuma condição for atendida.

Algoritmo 1: Estrutura Condicional

```
1 if Condição 1 then
2 |   Comandos
3 else if Condição 2 then
4 |   Comandos
5 else
6 |   Comandos
```

3.1.2 Laços de execução

Existem diversas maneiras de definirmos laços de execução, sendo que a mais comum é iterar uma variável sobre um conjunto de valores, ou sequência. Para iterarmos uma variável de um valor i_0 até um valor i_f de s em s, usamos

Algoritmo 2: Laço de execução

```
1 for i ← i_0 to i_f step s do
2 |   Comandos
```

ou

Algoritmo 3: Laço de execução

```
1 for i ∈ {i_0 : s : i_f} do
2 |   Comandos
```

A outra maneira é executar uma sequência de comandos enquanto uma determinada condição lógica é satisfeita.

Algoritmo 4: Repetição com teste no início

```
1 while Condição do
2 |   Comandos
```

3.1.3 Sub Rotinas

Do ponto de vista de programação, o ideal é sempre encapsularmos uma sequência de operações em uma sub rotina, que terá um conjunto de entradas, suas operações e um conjunto de saída.

Algoritmo 5: Sub rotinas

```
1 Subrotina(Entradas)
3 |   Comandos
5 |   return Resultados
```

3.2 Exemplos

Algoritmo 6: Calcula o valor de um polinômio em um ponto x

```
1  Polinomio(x)
       Entrada: x número real
2      Calcula o valor no ponto
3      valor ← 2 * x^2 − 2 * x + 2
4      Retorna o valor
6      return valor
```

Algoritmo 7: Calcula a média de uma lista de valores com dimensão n

```
1  Media(lista,n)
       Entrada: lista vetor de inteiros
       Entrada: n dimensão do vetor
2      Inicializa a variável do somatório
3      soma ← 0
4      Laço pela dimensão da lista
5      for i ∈ {1,...,n} do
6          Acumula o valor
7          soma ← soma + lista(i)
8      Retorna a média
10     return soma/n
```

Algoritmo 8: Soma duas matrizes de dimensão $m \times n$

```
1  Soma(A,B,m,n)
       Entrada: A matriz m × n
       Entrada: B matriz m × n
       Entrada: m número de linhas
       Entrada: n número de colunas
2      Inicializa a variável do somatório
3      C ← 0_{m×n}
4      Laço pelas linhas das matrizes
5      for i ∈ {1,,...,m} do
6          Laço pelas colunas das matrizes
7          for j ∈ {1,...,n} do
8              Soma os valores na linha e coluna
9              C(i,j) ← A(i,j) + B(i,j)
10     Retorna a matriz com a soma
12     return C
```

Algoritmo 9: Retorna *verdade* se o número for par ou *falso* do contrário

```
1  Par(a)
       Entrada: a número
2      Inicializa a variável lógica
3      saida ← verdade
4      Repita o laço enquanto a > 2
5      while a > 2 do
6          Descontamos 2 da variável
7          a ← a − 2
8      Se a = 1, o número é ímpar
9      if a = 1 then
10         saida ← falso
11     Retorna verdade se a for par. Do contrário, retorna falso
13     return saida
```

Algoritmo 10: Soma, de forma separada, os números pares e ímpares de uma lista de valores inteiros de dimensão n

```
1  SomaPares(lista,n)
       Entrada: lista vetor de inteiros
       Entrada: n dimensão do vetor
2      Inicializa as variáveis dos somatórios
3      soma_p ← 0
4      soma_i ← 0
5      Laço pelas posições da lista for i ∈ {1,...,n} do
6          Recupera o valor na posição i
7          a ← lista(i)
8          Testa se o valor é par
9          cond ← Par(a)
10         Soma de acordo com par ou ímpar
11         if cond then
12             soma_p ← soma_p + a
13         else
14             soma_i ← soma_i + a
15     Retorna os dois somatórios
17     return soma_p,soma_i
```

4. Raízes de Equações

Dada uma equação de uma variável $f(x) = 0$ chamamos de raiz o valor (ou valores) de x que satisfazem esta equação. Essas funções podem ser algébricas se são definidas em termos de operações básicas ou fundamentais, tais como soma, multiplicação, subtração, divisão, raiz quadrada, etc...) ou transcendentes se são definidas a partir de outras funções, como por exemplo e^x. Existem diversos métodos para determinação de raízes, sendo que iremos abordar os mais utilizados. Para cada método iremos apresentar um algoritmo com caráter didático, sem qualquer preocupação com eficiência computacional.

4.1 Método da Bisecção

Seja $f(x)$ uma função de uma variável e contínua entre dois pontos $x = a$ e $x = b$, com $b > a$. Se os valores da função em a e b tiverem sinal diferentes, então com certeza a função passa pelo zero pelo menos uma vez (ao menos uma raiz no intervalo $[a,b]$) e podemos então reduzir o intervalo, até que obtenhamos uma estimativa aceitável para o ponto $x = \alpha$, onde $f(\alpha) = 0$ (raiz). A lógica deste método é extremamente simples e robusta, pois iniciamos com um intervalo $[a,b]$ de dimensão $I_0 = b - a$ e reduzimos o intervalo pela metade a cada nova iteração, de acordo com o algoritmo 11.

É interessante notarmos que a cada iteração, o intervalo será dividido por 2. Assim, em uma iteração n

$$|I_n - I_1| \leq \frac{I_0}{2^n}$$

4.2 Método *Regula Falsi*

Este método é uma modificação do método da Biseção, onde ao invés de utilizamos o ponto médio do intervalo utilizamos a interseção de uma reta que une a e b com o eixo x. A

Algoritmo 11: Método da bisecção

```
1  Bisecao(a,b,tol,nmax)
       Entrada: a e b extremos do intervalo inicial
       Entrada: tol tolerância do tamanho do intervalo
       Entrada: nmax número máximo de reduções de intervalo
2      Verifica se temos raiz no intervalo
3      if f(a) * f(b) > 0 then
4          return erro
5      Calcula o intervalo inicial
6      I ← b − a
7      Executa o laço até que a raiz seja obtida OU até que o contador
        atinja o limite de iterações
8      for n ∈ {0,...,nmax} do
9          Calcula o ponto médio do intervalo
10         c ← (a+b)/2
11         Testa pela tolerância do intervalo
12         if I < tol then
13             escrever "Atingimos a tolerância", I
14             return c
15         Testa se a raiz está entre a e c
16         if f(a) * f(c) < 0 then
17             b ← c
18         else
19             a ← c
20     Se chegamos aqui, então não atingimos a tolerância
21     escrever "Não atingimos a tolerância: cuidado"
22     return c
```

equação desta reta é

$$f(c) = 0 = f(a) + \frac{f(b) - f(a)}{b - a}(c - a)$$

e, isolando o c

$$c = a - \frac{(b-a)f(a)}{f(b) - f(a)}$$

ou, de modo a minimizar error numéricos

$$c = a - \frac{(b-a)\left(\frac{f(a)}{f(b)}\right)}{\left[1 - \frac{f(a)}{f(b)}\right]}.$$

O resto do método é igual ao da bisecção, conforme ilustrado no algoritmo 12.

4.3 Método de Newton-Raphson

Considere uma função $f(x)$ contínua e diferenciável. No entorno de um ponto x_n, podemos representar esta função por uma série de Taylor, na forma

$$f(x_{n+1}) = f(x_n) + f'(x_n) \cdot (x_{n+1} - x_n) + \mathscr{O}$$

onde $\mathscr{O}$ representa os termos de alta ordem. Se x_{n+1} for um ponto de passagem por zero, isto é, $f(x_{n+1}) = 0$, então

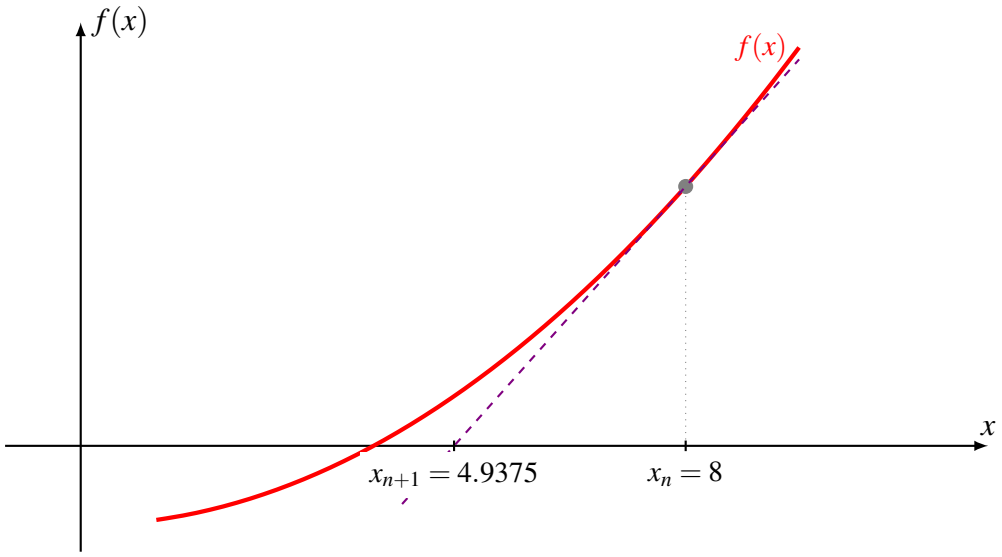
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{4.1}$$

que dá uma estimativa para um ponto cada vez mais próximo da raiz verdadeira e próxima de x_n.

■ **Exemplo 4.1** Um passo desse procedimento pode ser ilustrado utilizando a função $f(x) = 0.1x^2 - 1.5$ em uma posição arbitrária $x_n = 8$. Como a derivada da função é $f'(x) = 0.2x$, temos para o ponto

$$x_{n+1} = 8 - \frac{0.1(8)^2 - 1.5}{0.2(8)} = 4.9375$$

com ilustração



Algoritmo 12: Método da *Regula Falsi*

```
1  RegulaFalsi(a,b,tol,nmax)
       Entrada: a e b extremos do intervalo inicial
       Entrada: tol tolerância do tamanho do intervalo
       Entrada: nmax número máximo de iterações
2      Verifica se temos raiz no intervalo
3      if f(a) * f(b) > 0 then
4          return erro
5      Inicializa c
6      c ← a
7      Executa o laço até que a raiz seja obtida OU até que o contador
        atinja o limite de iterações
8      for n ∈ {0,...,nmax} do
9          Calcula a relação entre f(a) e f(b)
10         fab ← f(a)/f(b)
11         Calcula a intersecção da reta com o eixo x
12         c_n ← a − ((b − a) * fab/(1 − fab))
13         Testa pela convergência de c
14         if |c − c_n| < tol then
15             escrever "Atingimos a tolerância"
16             return c_n
17         Copia c_n para c
18         c ← c_n
19         Testa se a raiz está entre a e c
20         if f(a) * f(c) < 0 then
21             b ← c
22         else
23             a ← c
24     Se chegamos aqui, então não atingimos a tolerância
25     escrever "Não atingimos a tolerância: cuidado"
26     return c
```

A próxima iteração do método será

$$x_{n+1} = 4.375 - \frac{0.1(4.375)^2 - 1.5}{0.2(4.375)} = 3.9017857142857144$$

e, neste ponto, observamos que $f(3.9017857142857144) = 0.022$. Mais uma iteração nos leva para $x = 3.87308965348153$ cujo valor de f está na casa de 1×10^{-5}.

■

O interessante sobre este método, é que se a função e suas derivadas satisfizerem algumas condições, então a convergência (taxa de redução do intervalo) é quadrática. Para entendermos o motivo, voltamos à série de Taylor, mas agora retendo os termos de segunda ordem e usando uma variável intermediária α

$$f(\alpha) = f(x_n) + f'(x_n) \cdot (\alpha - x_n) + \frac{1}{2} f''(x_n) * (\alpha - x_n)^2$$

e, assumindo que α é um ponto de passagem por zero, então

$$0 = f(x_n) + f'(x_n) \cdot (\alpha - x_n) + \frac{1}{2} f''(x_n) \cdot (\alpha - x_n)^2$$

e, se dividirmos por $f'(x_n)$, obteremos

$$\frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) = -\frac{f''(x_n)(\alpha - x_n)^2}{2f'(x_n)}$$

e, substituindo x_n pela relação obtida na Eq, (4.1), obtemos

$$\alpha - x_{n+1} = -\frac{f''(x_n)}{2f'(x_n)} (\alpha - x_n)^2$$

tal que se considerarmos os módulos dos dois lados desta equação, obteremos

$$\varepsilon_{n+1} \leq M \varepsilon_n^2$$

onde $\varepsilon_{n+1} = |\alpha - x_{n+1}|$, $\varepsilon_n = |\alpha - x_n|$ e $M = \left| -\frac{f''(x_n)}{2f'(x_n)} \right|$. Assim, de acordo com o que vimos na primeira parte da matéria, o intervalo ε_{n+1} é limitado por um fator M, que será finito se f' for diferente de zero e se f" for limitado. Podemos ver ainda que o algoritmos converge para a raiz se o ponto x_n for próximo de α.

4.4 Método da Secante

Um das dificuldades na utilização do método de Newton-Raphson está justamente no cálculo da derivada. Para evitar este cálculo, podemos utilizar uma aproximação por diferenças finitas, na forma

$$f'(x_n) \simeq \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

de tal forma que a Eq. (4.1) pode ser escrita como

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n) - f(x_{n-1})} (x_n - x_{n-1}) \tag{4.2}$$

Algoritmo 13: Método de Newton-Raphson

```
1  NewtonRaphson(x,tol,nmax)
       Entrada: x ponto incial
       Entrada: tol tolerância do cálculo da raiz
       Entrada: nmax número máximo de iterações
2      Executa o laço até que a raiz seja obtida OU até que o contador
        atinja o limite de iterações
3      for n ∈ {0,...,nmax} do
4          Calcula a estimativa de raiz
5          x_n ← x − f(x)/df(x)
6          Testa pela tolerância da raiz
7          if |x − x_n| < tol then
8              escrever "Atingimos a tolerância"
9              return x_n
10         Copia x_n para x
11         x ← x_n
12     Se chegamos aqui, então não atingimos a tolerância
13     escrever "Não atingimos a tolerância: cuidado"
14     return x
```

e, portanto, devemos iniciar o método com dois pontos, ao invés do método de Newton-Raphson que necessita de apenas um ponto. Uma outra interpretação para esta abordagem consiste em traçar uma reta entre dois pontos (secante), na forma

$$f(x_{n+1}) = f(x_n) + \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}(x_{n+1} - x_n)$$

e, assumindo que x_{n+1} é uma estimativa de raiz, tal que $f(x_{n+1}) = 0$, obtemos novamente a Eq. (4.2).

É interessante comentar que o método, por utilizar uma aproximação da derivada, tem taxa de convergência igual a razão áurea, 1.618 quando não temos raizes repetidas na região do ponto incial. A prova deste resultado, bem como uma discussão bem completa sobre o procedimento discutido na dedução da taxa de convergência de Newton-Raphson pode ser obtida no artigo "A note on the Convergence of the Secant Methods for Simple and Multiple Roots", de P. Díez (Applied Mathematics Letters, 16(2003), pp. 1211-1215).

4.5 Exemplos

Vamos considerar a equação $f(x) = x - \cos(x)$ para $x \in [0,1]$, que tem uma raiz em $x = 0.7390851332151607$. Utilizaremos uma tolerância de 1×10^{-6} e $nmax = 100$. Os resultados serão mostrados com 7 casas decimais por uma questão de formatação.

■ **Exemplo 4.2** Método da Bisecção com $a = 0.2$ e $b = 1.0$

Algoritmo 14: Método da Secante

```
1  Secante(x_n,x_m,tol,nmax)
       Entrada: xn e xm pontos inciais
       Entrada: tol tolerância do cálculo da raiz
       Entrada: nmax número máximo de iterações
2      Executa o laço até que a raiz seja obtida OU até que o contador
        atinja o limite de iterações
3      for n ∈ {0,...,nmax} do
4          Calcula a estimativa de raiz
5          x ← x_n − (f(x_n) ∗ (x_n − x_m))/(f(x_n) − f(x_m))
6          Testa pela tolerância da raiz
7          if |x_n − x_m| < tol then
8              escrever "Atingimos a tolerância"
9              return x
10         Copia os pontos
11         x_m ← x_n
12         x_n ← x
13     Se chegamos aqui, então não atingimos a tolerância
14     escrever "Não atingimos a tolerância: cuidado"
15     return x_n
```

Iteração	a	b	$b-a$
0	2.000000e-01	1.000000e+00	8.000000e-01
1	6.000000e-01	1.000000e+00	4.000000e-01
2	6.000000e-01	8.000000e-01	2.000000e-01
...	....	....	....
20	7.390846e-01	7.390854e-01	7.629395e-07

■

■ **Exemplo 4.3** Método da *Regula Falsi* com $a = 0.2$ e $b = 1.0$

Iteração	a	b	$c_k - c_{k-1}$
0	2.000000e-01	1.000000e+00	2.966355e-01
1	7.033645e-01	1.000000e+00	3.389621e-02
2	7.372607e-01	1.000000e+00	1.732984e-03
3	7.389936e-01	1.000000e+00	8.690052e-05
4	7.390806e-01	1.000000e+00	4.353298e-06
5	7.390849e-01	1.000000e+00	2.180684e-07

■

■ **Exemplo 4.4** Método de Newton-Raphson com $x_0 = 0.2$

Iteração	x	$x_k - x_{k-1}$
0	2.000000e-01	
1	8.507771e-01	6.507771e-01
2	7.415302e-01	1.092469e-01
3	7.390864e-01	2.443744e-03
4	7.390851e-01	1.316662e-06
5	7.390851e-01	3.828049e-13

■

■ **Exemplo 4.5** Método da Secante, com $x_0 = 0.2$ $x_1 = 1.0$

Iteração	x_n	x_{n-1}	$x_n - x_{n-1}$
0	2.000000e-01	1.000000e+00	8.000000e-01
1	7.033645e-01	2.000000e-01	5.033645e-01
2	7.447825e-01	7.033645e-01	4.141808e-02
3	7.390395e-01	7.447825e-01	5.743044e-03
4	7.390851e-01	7.390395e-01	4.559160e-05
5	7.390851e-01	7.390851e-01	5.725747e-08

■

4.6 Exercícios

Exercício 4.1 Obtenha as raízes da função $f(x) = \sin(\pi x)$, $x \in [-\pi, \pi]$ para uma tolerância de 1×10^{-4} e um número máximo de 6 iterações, utilizado Regula-Falsi, Newton-Raphson e Método da Secante. Para isto, descreva em detalhes todas as etapas do cálculo e solucione passo a passo. ■

Exercício 4.2 A equação $x^3 - 2x^2 - 11x + 12$ tem como raízes -3, 1 e 4. O interessante é que pequenas diferenças nos pontos iniciais podem fazer com que o método de Newton-Raphson tenda para uma raiz diferente. Teste o método com $x_0 = 2.35287527$, $x_0 = 2.35284172$ e $x_0 = 2.352836323$. ■

Exercício 4.3 A equação $x^3 - 2x + 2$ com $x_0 = 0$ tem um comportamento muito interessante quando solucionada por Newton-Raphson. Verifique e explique em detalhes o que está ocorrendo. Para isto, investigue o gráfico da função e o comportamento da derivada. A melhor maneira é resolver graficamente, acompanhando as iterações do método.

■

5. Operações Básicas com Matrizes

Do ponto de vista de programação, uma matriz é uma estrutura de dados que contém uma série de valores. Existem diversas formas de armazenar uma matriz, sendo que nesta primeira parte do texto iremos tratar de armazenamento "denso", onde todas as posições são armazenadas em uma forma retangular. Uma matriz $\mathbf{A}$ com m linhas e n colunas é representada na forma,

$$\mathbf{A}_{[m\times n]} = A_{ij} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

sendo que, dependendo do padrão como os valores se distribuem no interior da matriz, adotamos as seguintes nomenclaturas:

- Matriz Simétrica: $a_{ij} = a_{ji}$
- Matriz Anti-simétrica: $a_{ij} = -a_{ji}$
- Matriz Identidade: $a_{ij} = \delta_{ij}$
- Matriz Nula: $a_{ij} = 0$
- Matriz Quadrada: $m = n$
- Matriz Triangular Superior: $a_{ij} = 0, \forall i > j$
- Matriz Triangular Inferior: $a_{ij} = 0, \forall i < j$
- Matriz Tridiagonal: Apenas a diagonal principal e as diagonais acima e abaixo da principal contém elementos não nulos.
- Matriz Pentadiagonal: Apenas a diagonal principal e duas diagonais acima e abaixo da principal contém elementos não nulos.

No que segue, iremos apresentar os algoritmos básicos para operações com matrizes retangulares sem impor qualquer tipo de particularidade quanto ao padrão de armazenamento.

5.1 Multiplicação por um escalar

Esta é a operação mais simples, onde multiplicamos toda uma matriz $\mathbf{A}_{[m\times n]}$ por um escalar e, na forma

$$e * \mathbf{A}_{[m\times n]} = \begin{bmatrix} e*a_{11} & \dots & e*a_{1n} \\ \vdots & \vdots & \vdots \\ e*a_{m1} & \dots & e*a_{mn} \end{bmatrix}$$

e o algoritmo referente a esta operação é ilustrado na listagem (15).

Algoritmo 15: Multiplicação de uma matriz $m \times n$ por um escalar

```
1  MatrizXescalar(A,e,m,n)
       Entrada: A matriz m × n
       Entrada: e escalar
2      Aloca a matriz de saída
3      C ← 0_{m×n}
4      Laço pelas linhas da matriz
5      for i ∈ {1,...,m} do
6          Laço pelas colunas da matriz
7          for j ∈ {1,...,n} do
8              Multiplica e por A(i,j) e armazena em C(i,j)
9              C(i,j) ← e*A(i,j)
10     Retorna a matriz C
11     return C
```

5.2 Soma de Matrizes

A soma de duas matrizes $\mathbf{A}_{[m\times n]}$ e $\mathbf{B}_{[m\times n]}$ só é possível se as matrizes tem exatamente o mesmo número de linhas e colunas. Neste caso, o resultado é uma matriz de mesmas dimensões das matrizes de entrada, na forma

$$\mathbf{C}_{[m\times n]} = \mathbf{A}_{[m\times n]} + \mathbf{B}_{[m\times n]} = \begin{bmatrix} a_{11}+b_{11} & \dots & a_{1n}+b_{1n} \\ \vdots & \vdots & \vdots \\ a_{m1}+b_{m1} & \dots & a_{mn}+b_{mn} \end{bmatrix}$$

e a operação é ilustrada no algoritmo (16)

5.3 Transposta de uma Matriz

Esta operação é definida pela relação

$$\mathbf{A}^T = (a_{ij})^T = a_{ji}$$

e a rotina que realiza esta operação é ilustrada no algoritmo (17)

Algoritmo 16: Soma de duas matrizes $m \times n$

```
1  MatrizSmatriz(A,B,m,n)
       Entrada: A matriz m × n
       Entrada: B matriz m × n
2      Aloca a matriz de saída
3      C ← 0_{m×n}
4      Laço pelas linhas das matrizes
5      for i ∈ {1,...,m} do
6          Laço pelas colunas das matrizes
7          for j ∈ {1,...,n} do
8              Soma as posições (i, j) e armazena em C(i, j)
9              C(i, j) ← A(i, j) + B(i, j)
10     Retorna a matriz C
11     return C
```

Algoritmo 17: Transposta de uma matriz $m \times n$

```
1  MatrizT(A,m,n)
       Entrada: A matriz m × n
2      Aloca a matriz de saída
3      C ← 0_{n×m}
4      Laço pelas linhas da matriz
5      for i ∈ {1,...,m} do
6          Laço pelas colunas da matriz
7          for j ∈ {1,...,n} do
8              Troca i e j em C(j,i)
9              C(j,i) ← A(i, j)
10     Retorna a matriz C
11     return C
```

5.4 Multiplicação de Matriz por um vetor

A multiplicação de uma matriz $\mathbf{A}_{[m\times n]}$ por um vetor coluna $\mathbf{v}_n$ só é possível se o número de colunas da matriz for igual ao número de linhas do vetor e resulta em um vetor coluna com m linhas. A operação é definida por

$$\mathbf{A}_{[m\times n]}\mathbf{v}_{[n]} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{n} a_{1j}v_j \\ \vdots \\ \sum_{j=1}^{n} a_{mj}v_j \end{bmatrix}.$$

O procedimento está listado no Algoritmo (18).

Algoritmo 18: Produto de uma matriz $m \times n$ por um vetor $n \times 1$

```
1  MatrizXvetor(A,v,m,n)
       Entrada: A matriz m × n
       Entrada: v vetor n × 1
2      Aloca o vetor de saída
3      u ← 0_m
4      Laço pelas linhas da matriz
5      for i ∈ {1,...,m} do
6          Inicializa o somatório da linha pelo vetor
7          soma ← 0
8          Laço pelas colunas da matriz e pelo vetor
9          for j ∈ {1,...,n} do
10             Acumula o somatório
11             soma ← soma + A(i, j) ∗ v(j)
12         Armazena o somatório no vetor de saída
13         v(i) ← soma
14     Retorna o vetor u
15     return u
```

5.5 Produto Interno

O produto interno de dois vetores $\mathbf{u}$ e $\mathbf{v}$, ambos de dimensão $n \times 1$ é obtido por

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{n} u_i v_i$$

resultando em um escalar. O procedimento é ilustrado no algoritmo 19

5.6 Produto de duas Matrizes

A multiplicação de uma matriz $\mathbf{A}_{[m\times n]}$ por outra matriz $\mathbf{B}_{[n\times o]}$ só é possível se o número de colunas da matriz $\mathbf{A}$ for igual ao número de linhas da matriz $\mathbf{B}$. Esta operação resulta em uma matriz $\mathbf{C}_{[m\times o]}$, tal que

$$\mathbf{A}_{[m\times n]}\mathbf{B}_{[n\times o]} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} b_{11} & \dots & b_{1o} \\ \vdots & \vdots & \vdots \\ b_{n1} & \dots & b_{no} \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{n} a_{1j}b_{j1} & \dots & \sum_{j=1}^{n} a_{1j}b_{jo} \\ \vdots & \vdots & \vdots \\ \sum_{j=1}^{n} a_{mj}b_{j1} & \dots & \sum_{j=1}^{n} a_{mj}b_{jo} \end{bmatrix}.$$

Algoritmo 19: Produto interno de dois vetores de dimensão $n \times 1$

```
1 Dotuv(u,v,n)
    Entrada: u vetor n × 1
    Entrada: v vetor n × 1
2   Inicializa o somatório
3   soma ← 0
4   Laço pelas posições dos vetores
5   for i ∈ {1,...,n} do
6       Acumula o somatório
7       soma ← soma + u(i) * v(i)
8   Retorna o escalar
9   return soma
```

O procedimento está listado no Algoritmo (20).

Algoritmo 20: Produto de uma matriz $m \times n$ por um uma matriz $n \times o$

```
1  MatrizXmatriz(A,B,m,n,o)
     Entrada: A matriz m × n
     Entrada: B matriz n × o
2    Aloca a matriz de saída
3    C ← 0_{m×o}
4    Laço pelas linhas da matriz
5    for i ∈ {1,...,m} do
6        for k ∈ {1,...,o} do
7            soma ← 0
8            for j ∈ {1,...,n} do
9                soma ← soma + A(i,j) * B(j,k)
10           C(i,j) ← soma
11   Retorna a matriz C
12   return C
```

5.7 Norma p de um Vetor

A norma p de um vetor é obtida por

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} x_i^p \right)^{\frac{1}{p}}$$

e o procedimento é ilustrado no Alg. (21) .

5.8 Traço de Uma Matriz

O traço é a soma dos termos da diagonal de uma matriz quadrada A, na forma

$$tr(\mathbf{A}) = \sum_{i=1}^{n} A_{ii}$$

Algoritmo 21: Norma p de um vetor de dimensão $n \times 1$

```
1 NormaPu(u,n,p)
    Entrada: u vetor n × 1
    Entrada: p expoente da norma
2   Inicializa o somatório
3   soma ← 0
4   Laço pelas posições do vetor
5   for i ∈ {1,...,n} do
6       Acumula o somatório
7       soma ← soma + u(i)^p
8   Calcula a raiz p e retorna o escalar
9   return (soma)^(1/p)
```

e o procedimento é ilustrado no Alg. (22) .

Algoritmo 22: Traço de uma matriz quadrada $n \times n$

```
1 TracoA(A,n)
    Entrada: A matriz n × n
2   Inicializa o somatório
3   soma ← 0
4   Laço pelas posições da diagonal
5   for i ∈ {1,...,n} do
6       Acumula o somatório
7       soma ← soma + A(i,i)
8   Retorna o escalar
9   return soma
```

5.9 Extrai uma coluna de uma matriz

Essa operação será definida para podermos utilizar nos algoritmos dos próximos capítulos.

Algoritmo 23: Devolve um vetor com os elementos da coluna i da matriz $\mathbf{A}$ $m \times n$

```
1 Coluna(A,i,m)
    Entrada: A matriz m × n
    Entrada: i <= n coluna
2   Inicializa um vetor de saída
3   v ← 0_m
4   Laço pelas posições da coluna
5   for j ∈ {1,...,m} do
6       Copia
7       v(j) ← A(j,i)
8   Retorna o vetor
9   return v
```

5.10 Complexidade de um Algoritmo

Um conceito importante em um algoritmo é o de complexidade, ou seja, o número de operações que são necessárias para realizar a tarefa, dada as dimensões das matrizes e vetores envolvidos. Desta forma, diz-se que um algoritmo tem complexidade $\mathcal{O}(n)$, por exemplo, quando depende linearmente do número de dimensões das matrizes e vetores envolvidos. Para os algoritmos apresentados neste capítulo, podemos verificar facilmente que:

- Multiplicação de escalar por matriz: $\mathcal{O}(m*n)$, operações de multiplicação;
- Soma de matrizes: $\mathcal{O}(m*n)$, operações de soma;
- Transposta de uma matriz: $\mathcal{O}(m*n)$, operações de à memória;
- Multiplicação de uma matriz por um vetor: $\mathcal{O}(m*n)$, operações de soma e de multiplicação;
- Multiplicação de duas matrizes $A_{[m\times n\}}$ e $B_{[n\times o]}$:$\mathcal{O}(m*o*n)$, operações de soma e de multiplicação;

Outra informação importante está associada ao erro propagado em cada uma destas operações, sendo que com isto podemos ter uma noção da acurácia esperada na resposta.

Deve-se salientar que a complexidade não mostra de forma detalhada os diferentes tipos de operações que são realizadas, como multiplicação, divisão, soma e subtração (por exemplo) e, muito relevante em problemas grandes, acesso a memória. Cada operação tem um custo diferente em diferentes processadores. Além disso, alguns processadores podem agrupar diversas operações em algumas operações vetorizadas (que são processadas em paralelo no processador), o que aumenta muito o desempenho.

Uma maneira muito utilizada de se medir a performance de um algoritmo é utilizando o número de operações de ponto flutuante por segundo (FLOPS) em função do tamanho do problema. Nos computadores atuais atingimos GFLOPS ou bilhões de operações por segundo em operações como multiplicação de matrizes, por exemplo.

5.11 BLAS - Basic Linear Algebra System

Quando se trata de implementação eficiente de rotinas básicas de álgebra linear, como as vistas neste capítulo, é sempre preferível utilizar as rotinas da biblioteca BLAS (Basic Linear Algebra System) em https://www.netlib.org/blas/. Essas rotinas são referência para diversas bibliotecas que utilizam informações específicas sobre o tipo de processador utilizado, tais como operações vetoriais, número de núcleos e memória cache. Exemplos de implementações eficientes das rotinas BLAS são a biblioteca MKL (Math Kernel Library) da Intel e OpenBLAS (https://www.openblas.net/), que é de código aberto. Essas rotinas são, originalmente, escritas em Fortran ou C, mas são utilizadas em diversas linguagens de alto nível, como Julia, Scilab, Matlab e Octave, por exemplo.

6. Triangularização de uma Matriz Quadrada

Um conceito fundamental para o estudo dos algoritmos básicos de álgebra linear está associado ao conceito de triangularização de uma matriz quadrada. Neste procedimento, transformamos uma matriz quadrada $\mathbf{A}_{[n\times n]}$, de estrutura geral, em uma matriz triangular superior, isto é, com termos não nulos armazenados somente da diagonal superior para "cima" . Este procedimento é utilizado em diversos algoritmos que servem de base para a solução de sistemas de equações lineares e cálculos de determinantes, além de prover fundamentos para aplicações mais complexas. Devido a este fato, iremos abordar a triangularização como um tópico separado, sendo que suas aplicações serão apresentadas em seções posteriores.

6.1 Método de Gauss

No método de Gauss, operamos com uma matriz quadrada $\mathbf{A}_{[n\times n]}$, partindo da primeira linha, $i = 1$ e aplicando duas etapas sucessivamente:

- $L_i = L_i/a_{ii}$;
- $L_k = L_k - a_{ki}L_i$, $k = i+1..n$

■ **Exemplo 6.1** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 2 & 4 & 1 \\ 3 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

Começamos com a primeira linha, $i = 1$

$$L_1 = \frac{1}{2}[\ 2 \quad 4 \quad 1\ ] = [\ 1 \quad 2 \quad 0.5\ ]$$

Com isso, realizamos um laço em k de $i+1 = 2$ até $n = 3$:

$$\begin{aligned} L_2 &= [\ 3 \quad 1 \quad -1\ ] - 3[\ 1 \quad 2 \quad 0.5\ ] = [\ 0 \quad -5 \quad -2.5\ ] \\ L_3 &= [\ 1 \quad 1 \quad 1\ ] - 1[\ 1 \quad 2 \quad 0.5\ ] = [\ 0 \quad -1 \quad 0.5\ ] \end{aligned}$$

tal que agora a matriz tem a forma

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 2 & 0.5 \\ 0 & -5 & -2.5 \\ 0 & -1 & 0.5 \end{bmatrix}$$

e observamos que toda a coluna abaixo da primeira posição diagonal foi zerada. Partindo agora da segunda linha, $i = 2$

$$L_2 = \frac{1}{-5}[\ 0 \quad -5 \quad -2.5\ ] = [\ 0 \quad 1 \quad 0.5\ ]$$

e realizamos um laço para considerar as linhas abaixo da atual, que no caso é a linha 3

$$L_3 = [\ 0 \quad -1 \quad 0.5\ ] - (-1)[\ 0 \quad 1 \quad 0.5\ ] = [\ 0 \quad 0 \quad 1\ ]$$

Com isso, realizamos todo o procedimento, obtendo

$$\mathbf{A}_2 = \begin{bmatrix} 1 & 2 & 0.5 \\ 0 & 1 & 0.5 \\ 0 & 0 & 1 \end{bmatrix}$$

que é uma matriz na forma triangular superior. ∎

6.1.1 Pivotamento

Pode-se notar que o procedimento visto acima não pode ser aplicado se a matriz possui algum zero em sua diagonal, pois neste caso a etapa 1, onde dividimos a linha pelo valor da diagonal,não será definido. Isto pode ser facilmente visualizado no seguinte exemplo:

∎ Exemplo 6.2

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 2 \\ 3 & 6 & 1 \\ 2 & 6 & -1 \end{bmatrix}$$

onde o procedimento de triangularização da matriz, já na primeira etapa, resulta em um zero na diagonal, pois

$$\begin{aligned} L_1 &= \frac{1}{1}[\ 1 \quad 2 \quad 2\ ] = [\ 1 \quad 2 \quad 2\ ] \\ L_2 &= [\ 3 \quad 6 \quad 1\ ] - 3[\ 1 \quad 2 \quad 2\ ] = [\ 0 \quad 0 \quad -5\ ] \\ L_3 &= [\ 2 \quad 6 \quad -1\ ] - 2[\ 1 \quad 2 \quad 2\ ] = [\ 0 \quad 2 \quad -5\ ] \end{aligned}$$

resultando em

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 2 & 2 \\ 0 & 0 & -5 \\ 0 & 2 & -5 \end{bmatrix}$$

de tal forma que na próxima etapa teremos uma divisão por zero. Uma alternativa consiste em trocar as linhas 2 e 3 (pivotamento) antes de prosseguir, de modo a colocar o 2 na posição da diagonal. Com isto, teremos

$$\mathbf{A}_{1p} = \begin{bmatrix} 1 & 2 & 2 \\ 0 & 2 & -5 \\ 0 & 0 & -5 \end{bmatrix}$$

e, com isto,

$$\begin{aligned} L_2 &= \frac{1}{2}[\ 0 \quad 2 \quad -5\ ] = [\ 0 \quad 1 \quad -\tfrac{5}{2}\ ] \\ L_3 &= [\ 0 \quad 0 \quad -5\ ] - 0\,[\ 0 \quad 1 \quad -\tfrac{5}{2}\ ] = [\ 0 \quad 0 \quad -5\ ] \end{aligned}$$

resultado em

$$\mathbf{A}_2 = \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & -\frac{5}{2} \\ 0 & 0 & -5 \end{bmatrix}$$

e

$$\mathbf{A}_3 = \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & -\frac{5}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

se dividirmos a ultima linha por -5 (não seria necessário, mas com isto obtemos uma diagonal unitária), obtemos uma matriz que é triangular superior.

■

■ **Exemplo 6.3** Uma outra maneira de evitarmos divisão por zero na diagonal consiste em realizar o pivotamento prévio na matriz, garantindo que as maiores posições de cada coluna fiquem nas posições diagonais. No exemplo

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 2 \\ 3 & 6 & 1 \\ 2 & 6 & -1 \end{bmatrix}$$

e verificamos que $a_{21} > a_{31} > a_{11}$. Portanto trocamos as posições da linha 1 com a linha 2, obtendo

$$\mathbf{A} = \begin{bmatrix} 3 & 6 & 1 \\ 1 & 2 & 2 \\ 2 & 6 & -1 \end{bmatrix}$$

e por fim verificamos que $a_{32} > a_{22}$, e trocamos as linhas 2 e 3, obtendo

$$\mathbf{A} = \begin{bmatrix} 3 & 6 & 1 \\ 2 & 6 & -1 \\ 1 & 2 & 2 \end{bmatrix}.$$

Agora, podemos verificar que os maiores valores de cada coluna estão na diagonal principal. Este procedimento faz com que a matriz possa ser triangularizada sem problemas. Assim,

$$\begin{aligned} L_1 &= \frac{1}{3}[\ 3 \quad 6 \quad 1\ ] = [\ 1 \quad 2 \quad \tfrac{1}{3}\ ] \\ L_2 &= [\ 2 \quad 6 \quad -1\ ] - 2\,[\ 1 \quad 2 \quad \tfrac{1}{3}\ ] = [\ 0 \quad 2 \quad -\tfrac{5}{3}\ ] \\ L_3 &= [\ 1 \quad 2 \quad 2\ ] - 1\,[\ 1 \quad 2 \quad \tfrac{1}{3}\ ] = [\ 0 \quad 0 \quad \tfrac{5}{3}\ ] \end{aligned}$$

resultado em

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & \frac{1}{3} \\ 0 & 2 & -\frac{5}{3} \\ 0 & 0 & \frac{5}{3} \end{bmatrix}$$

que já é triangular superior. Continuando com o algoritmo

$$L_2 = \frac{1}{2}[\ 0 \quad 2 \quad -\tfrac{5}{3}\ ] = [\ 0 \quad 1 \quad -\tfrac{5}{6}\ ]$$
$$L_3 = [\ 0 \quad 0 \quad \tfrac{5}{3}\ ] - 0\,[\ 0 \quad 1 \quad -\tfrac{5}{6}\ ] = [\ 0 \quad 0 \quad \tfrac{5}{3}\ ]$$

e, finalmente,

$$L_3 = \frac{3}{5}[\ 0 \quad 0 \quad \tfrac{5}{3}\ ] = [\ 0 \quad 0 \quad 1\ ]$$

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & \frac{1}{3} \\ 0 & 1 & -\frac{5}{6} \\ 0 & 0 & 1 \end{bmatrix}$$

■

Deve-se salientar que o pivotamento global, realizado antes do procedimento de triangularização, é desejável numericamente. Isto se deve ao fato de as vezes podermos ter números muito pequenos na diagonal principal, que mesmo se não resultarem em divisão por zero poderão acarretar grandes erros numéricos. Desta forma, o pivotamento deixa a matriz em uma forma mais apropriada para os cálculos a serem realizados.

6.2 Cálculo de Determinante de uma matriz obtida por triangularização

O cálculo do determinante pode ser realizado facilmente durante a triangularização de uma matriz. Para isto, aplicamos a equação

$$\det(\mathbf{A}) = (-1)^p \prod_{i=1}^{n} F_i$$

onde F_i são os fatores utilizados para normalizar cada linha L_i no procedimento de triangularização e p é o número de pivotamentos realizados.

■ **Exemplo 6.4** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 2 & 4 & 1 \\ 3 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Como não realizamos pivotamento, $p = 0$. Os fatores utilizados na triangularização foram $F_1 = 2$ e $F_2 = -5$ tal que

$$\det(\mathbf{A}) = (-1)^0 (2 \cdot -5) = -10$$

■

■ **Exemplo 6.5** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 2 \\ 3 & 6 & 1 \\ 2 & 6 & -1 \end{bmatrix}.$$

Realizamos dois pivotamentos, tal que $p = 2$. Os fatores de normalização de cada linha foram $F_1 = 3$, $F_2 = 2$ e $F_3 = \frac{5}{3}$, tal qe

$$\det(\mathbf{A}) = (-1)^2 \left(2 \cdot 2 \cdot \frac{5}{3}\right) = 10.$$

■

6.3 Solução de um Sistema de Equações Lineares

Um sistema de equações lineares pode ser descrito na forma

$$\mathbf{A}_{[n\times n]}\mathbf{x}_{[n]} = \mathbf{b}_{[n]}$$

onde $\mathbf{A}$ é uma matriz quadrada de coeficientes com determinante não nulo, $\mathbf{b}$ é um vetor com termos conhecidos e $\mathbf{x}$ é um vetor de incógnitas. Embora analiticamente possamos escrever a solução do sistema de equações como

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

isto **geralmente** não é viável do ponto de vista numérico, uma vez que a operação de inversão tem complexidade extremamente alta. Uma abordagem consiste em triangularizar o sistema, na forma

$$\mathbf{U}\mathbf{x} = \bar{\mathbf{b}}$$

onde $\mathbf{U}$ é triangular superior e $\bar{\mathbf{b}}$ é modificado no processo de triangularização. A maneira mais simples de apresentar o método é trabalharmos com uma matriz aumentada

$$\tilde{\mathbf{A}} = [\mathbf{A}\ \mathbf{b}]$$

onde o vetor $\mathbf{b}$ entra como uma coluna adicional. Com isso, podemos obter, após o processo de triangularização de $\tilde{\mathbf{A}}$

$$\tilde{\mathbf{U}} = \left[\mathbf{U}\ \tilde{\mathbf{b}}\right]$$

tal que

$$\mathbf{U}\mathbf{x} = \bar{\mathbf{b}}$$

tem solução muito simples, pois começando da última linha ($i = n$)

$$x_n = \bar{b}_n$$

(lembrando que, neste material, estamos garantindo que a diagonal é unitária). Tendo x_n, podemos utilizar a penúltima linha ($i = n-1$)

$$x_{n-1} + u_{n-1,n}x_n = \bar{b}_{n-1}$$

com solução

$$x_{n-1} = \bar{b}_{n-1} - u_{n-1,n}x_n.$$

Essa mesma lógica pode ser utilizada até que $i = 1$

$$x_i = \bar{b}_i - \sum_{k=i+1}^{n} u_{ik}x_k,\ i = n-1..1$$

onde deve-se observar que o contador i inicia em n e decrementa até 1 (por isto chamamos de retro substituição).

O algoritmo para solução de um sistema de equações e calculo de determinante é ilustrado no Algoritmo (24).

■ **Exemplo 6.6** Seja o sistema de equações lineares

$$\begin{bmatrix} 2 & 4 & 1 \\ 3 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 13 \\ 2 \\ 6 \end{Bmatrix}$$

Montando a matriz aumentada

$$\tilde{\mathbf{A}} = \begin{bmatrix} 2 & 4 & 1 & 13 \\ 3 & 1 & -1 & 2 \\ 1 & 1 & 1 & 6 \end{bmatrix}$$

podemos realizar a triangularização da matriz aumentada com o procedimento padrão. Começando pela primeira linha ($i = 1$)

$$L_1 = \frac{1}{2}\begin{bmatrix} 2 & 4 & 1 & 13 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0.5 & 6.5 \end{bmatrix}$$

e realizamos um laço da linha 2 para a linha 3 da matriz aumentada

$$L_2 = \begin{bmatrix} 3 & 1 & -1 & 2 \end{bmatrix} - 3 \cdot \begin{bmatrix} 1 & 2 & 0.5 & 6.5 \end{bmatrix} = \begin{bmatrix} 0 & -5 & -2.5 & -17.5 \end{bmatrix}$$

e

$$L_3 = \begin{bmatrix} 1 & 1 & 1 & 6 \end{bmatrix} - 1 \cdot \begin{bmatrix} 1 & 2 & 0.5 & 6.5 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0.5 & -0.5 \end{bmatrix}.$$

Com isso, terminamos a primeira etapa, que anula a primeira coluna da matriz, obtendo

$$\tilde{\mathbf{A}}_1 = \begin{bmatrix} 1 & 2 & 0.5 & 6.5 \\ 0 & -5 & -2.5 & -17.5 \\ 0 & -1 & 0.5 & -0.5 \end{bmatrix}$$

A segunda iteração do algoritmo inicia na linha $i = 2$

$$L_2 = \frac{1}{-5}\begin{bmatrix} 0 & -5 & -2.5 & 6.5 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0.5 & 3.5 \end{bmatrix}$$

e modificamos a terceira linha para anularmos o termo abaixo da segunda coluna

$$L_3 = \begin{bmatrix} 0 & -1 & 0.5 & -0.5 \end{bmatrix} - (-1) \cdot \begin{bmatrix} 0 & 1 & 0.5 & 3.5 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 3 \end{bmatrix}.$$

tal que

$$\tilde{\mathbf{U}} = \begin{bmatrix} 1 & 2 & 0.5 & 6.5 \\ 0 & 1 & 0.5 & 3.5 \\ 0 & 0 & 1 & 3 \end{bmatrix}.$$

Com isso, obtemos o sistema

$$\begin{bmatrix} 1 & 2 & 0.5 \\ 0 & 1 & 0.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 6.5 \\ 3.5 \\ 3 \end{Bmatrix}.$$

Realizando a retro substituição, começamos pela última linha

$$x_3 = 3$$

vamos para a penúltima linha

$$x_2 + 0.5 \cdot x_3 = 3.5 \implies x_2 = 3.5 - 0.5 \cdot 3 = 2$$

e terminamos na primeira linha

$$x_1 + 2 \cdot x_2 + 0.5 \cdot x_3 = 6.5 \implies x_1 = 6.5 - 2 \cdot 2 + 0.5 \cdot 3 = 1$$

tal que a solução é o vetor $\mathbf{x} = (1, 2, 3)$.

■

Algoritmo 24: Solução o sistema linear $\mathbf{Ax} = \mathbf{b}$, com n equações, utilizando triangularização de Gauss. A matriz $\mathbf{A}$ é modificada para a matriz $\mathbf{U}$ e a solução $\mathbf{x}$ é devolvida no vetor $\mathbf{b}$. A rotina também devolve o determinante da matriz $\mathbf{A}$.

```
1  Gauss(A,b,n)
       Entrada: A matriz n × n
       Entrada: b vetor n × 1
2      Inicializa o determinante
3      detA ← 1
4      Laço pelas linhas da matriz
5      for i ∈ {1,...,n} do
6          Extrai o termo da diagonal
7          aii ← A(i,i)
8          produtório do determinante
9          detA ← detA · aii
10         Laço pelas colunas da linha
11         for j ∈ {i,...,n} do
12             A(i, j) ← A(i, j)/aii
13         b(i) ← b(i)/aii
14         Laço pelas linhas abaixo da atual
15         for k ∈ {i+1,...,n} do
16             Termo a zerar nesta linha/coluna
17             aki ← A(k,i)
18             Laço pelas colunas da linha
19             for j ∈ {i,...,n} do
20                 A(k, j) ← A(k, j) − aki · A(i, j)
21             b(k) ← b(k) − aki · b(i)
22     Retro substituição
23     for i ∈ {n−1,n−2,...,1} do
24         Obtém a solução da linha i
25         for k ∈ {i+1,...,n} do
26             b(i) ← b(i) − A(i,k) · b(k)
27     Retorna o vetor b e o determinante
28     return b,detA
```

6.3.1 Inversa de uma Matriz

Conforme já foi comentado, o cálculo da inversa de uma matriz é extremamente custoso do ponto de vista computacional. No entanto, da definição da inversa de uma matriz

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}.$$

Essa relação pode ser vista como a solução simultânea de n sistemas de equações lineares com a forma

$$\mathbf{A}\mathbf{x}_i = \mathbf{1}_i$$

onde $\mathbf{x}_i$ corresponde a $i-$*ésima* coluna da matriz inversa e $\mathbf{1}_i$ é um vetor com $n-1$ termos nulos e 1 na posição i. Como a triangularização da matriz independe do vetor do lado direito, podemos estender o método de Gauss para considerarmos a solução simultânea de n sistemas de equação. No final, teremos como resposta a matriz inversa, com um custo computacional mais baixo do que se tivéssemos utilizado a abordagem tradicional. O procedimento é ilustrado no Algoritmo (25).

6.3.2 Análise de Complexidade

Uma análise grosseira do algoritmo 24 mostra que temos uma complexidade aproximada de $\mathcal{O}(n^3)$, pois temos 3 laços encadeados. Além disso, observamos que as operações básicas realizadas são divisão, subtração e multiplicação (essas últimas realizadas sempre em conjunto). Quando o sistema tem $n = 3$ equações, o Algoritmo 24 necessita de: 9 divisões, 14 subtrações e 14 multiplicações, totalizando 37 operações. Para um sistema com $n = 100$, teremos 5150 divisões, 343200 subtrações e 343200 multiplicações, totalizando 691550 operações. Como podemos notar, o número efetivo de operações é maior do que o previsto pela complexidade.

Se quisermos somente calcular a inversa, Alg. 25, termos, naturalmente, um número maior de cálculos. Para inverter uma matriz com $n = 3$, teremos 15 divisões, 26 subtrações e 26 multiplicações. Para $n = 100$, 15050 divisões, 1323300 subtrações e 1323300 multiplicações. No entanto, pela lógica dos laços, também observamos uma complexidade $\mathcal{O}(n^3)$.

Algoritmo 25: Inverte a matriz $\mathbf{A}_{n\times n}$ utilizando eliminação de Gauss

```
1  InvGauss(A,n)
       Entrada: A matriz n × n
2      Inicializa a matriz identidade
3      I ← 0_{n×n}
4      for i ∈ {1,...,n} do
5          I(i,i) ← 1
6      Laço pelas linhas da matriz
7      for i ∈ {1,...,n} do
8          Extrai o termo da diagonal
9          aii ← A(i,i)
10         Laços pelas colunas da linha
11         for j ∈ {i,...,n} do
12             A(i,j) ← A(i,j)/aii
13         for j ∈ {1,...,n} do
14             I(i,j) ← I(i,j)/aii
15         Laço pelas linhas abaixo da atual
16         for k ∈ {i+1,...,n} do
17             Termo a zerar nesta linha/coluna
18             aki ← A(k,i)
19             Laço pelas colunas da linha
20             for j ∈ {i,...,n} do
21                 A(k,j) ← A(k,j) − aki·A(i,j)
22             for j ∈ {1,...,n} do
23                 I(k,j) ← I(k,j) − aki·I(i,j)
24     Retro substituição
25     for i ∈ {n−1,n−2,...,1} do
26         Obtém a solução da linha i coluna j
27         for j ∈ {1,...,n} do
28             for k ∈ {i+1,...,n} do
29                 I(i,j) ← I(i,j) − A(i,k)·I(k,j)
30     Retorna a matriz I com a Inversa
31     return I
```

7. Decomposição LU

Na solução do sistema de equações $\mathbf{A}_{[n\times n]}\mathbf{x}_{[n]} = \mathbf{b}_{[n]}$, podemos re-escrever a matriz **A** na forma

$$\mathbf{A} = \mathbf{LU} \tag{7.1}$$

onde **L** é uma matriz triangular inferior e **U** uma matriz triangular superior com diagonal unitária (por definição). Os termos de **L** e de **U** são facilmente obtidos, uma vez que o número de incógnitas $l_{ij}\, i > j$ e $u_{ij}\, i < j$ é igual ao número de termos da matriz **A**. Uma vez realizado este procedimento, obtemos

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj} \quad \forall i > j$$

e

$$u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}}{l_{ii}} \quad \forall i < j$$

para os termos fora da diagonal e

$$l_{ii} = a_{ii} - \sum_{k=1}^{i-1} l_{ik}u_{ki}$$

e

$$u_{ii} = 1$$

para os termos das diagonais. Uma vez que tenhamos realizado este procedimento, teremos um sistema de equações na forma

$$\mathbf{LUx} = \mathbf{b}. \tag{7.2}$$

Definindo um problema auxiliar na forma

$$\mathbf{Ux} = \mathbf{y} \tag{7.3}$$

tal que

$$\mathbf{x} = \mathbf{U}^{-1}\mathbf{y} \tag{7.4}$$

e, inserindo a Eq. (7.4) na Eq. (7.2), obtemos

$$\mathbf{LUU}^{-1}\mathbf{y} = \mathbf{b}$$

tal que

$$\mathbf{Ly} = \mathbf{b}. \tag{7.5}$$

O problema definido na Eq. (7.5) pode ser solucionado por substituição direta, uma vez que **L** é triangular inferior, tal que

$$y_i = \frac{b_i - \sum_{k=1}^{i-1} l_{ik} y_k}{l_{ii}}, \quad i = 1..n$$

e, de posse de **y** e da Eq. (7.3), podemos finalmente obter **x** por meio de uma retro substituição, na forma

$$x_i = y_i - \sum_{k=i+1}^{n} u_{ik} x_k, \; i = n..1.$$

7.1 Calculo de Determinante por Decomposição LU

O determinante de uma matriz triangular é dado pelo produto de seus termos diagonais. Assim,

$$det(\mathbf{A}) = det\,(\mathbf{LU}) = \prod_{i=1}^{n} l_{ii}$$

pois os termos da diagonal de **U** são unitários. Para verificarmos, podemos utilizar o resultado do exemplo anterior, onde

$$\mathbf{L} = \begin{bmatrix} 2 & 0 & 0 \\ 3 & -5 & 0 \\ 1 & -1 & 1 \end{bmatrix}$$

e, portanto, $det(\mathbf{A}) = 2 \cdot -5 \cdot 1 = -10$.

O algoritmo para decomposição LU e cálculo do determinante é apresentado no Alg. 26. O Algoritmo para a solução de um sistema linear de equações utilizando a decomposição LU é apresentado no Alg. 27. Observe que a decomposição LU só precisa ser realizada uma vez, mesmo que tenhamos vários lados direitos no sistema de equações.

Algoritmo 26: Decomposição LU de uma matriz $\mathbf{A}$, $n \times n$. A triangular inferior (incluindo a diagonal principal) de $\mathbf{A}$ irá conter $\mathbf{L}$ e a triangular superior de $\mathbf{A}$ irá conter $\mathbf{U}$ (sem a diagonal, que é unitária). A rotina também calcula o determinante de $\mathbf{A}$.

```
1  LU(A,n)
       Entrada: A matriz n × n
2      Inicializa o determinante
3      dete ← 1
4      Laço pelas linhas da matriz
5      for i ∈ {1,...,n} do
6          Laço pelas colunas da matriz
7          for j ∈ {i,...,n} do
8              c ← i
9              Verifica se estamos abaixo da diagonal
10             if j < i then
11                 c ← j
12             Inicializa o somatório
13             soma ← A(i, j)
14             for k ∈ {1,...,c − 1} do
15                 soma ← soma − A(i,k) · A(k, j)
16             Armazena em A. Se i <= j armazena L
17             if j <= i then
18                 A(i, j) ← soma
19             else
20                 A(i, j) ← soma/A(i,i)
21         Produtório do determinante
22         dete ← dete · A(i,i)
23     Retorna a decomposição LU e o determinante
24     return A,dete
```

Algoritmo 27: Solução de um sistema linear de Equações $\mathbf{Ax} = \mathbf{b}$, com n equações. Espera-se que a matriz **ALU** tenha sido obtida com a rotina do Alg. 26.

```
1  SolveLU(ALU,b,n)
       Entrada: ALU matriz n × n com a decomposição LU
       Entrada: b vetor n × 1
2      Soluciona para y
3      y ← 0_n
4      for i ∈ {1,...,n} do
5          soma ← 0
6          for k ∈ {1,...,i−1} do
7              soma ← soma + ALU(i,k) · y(k)
8          y(i) ← (b(i) − soma)/ALU(i,i)
9      Soluciona para x, sobre-escrevendo b
10     for i ∈ {n,n−1,...,1} do
11         soma ← 0
12         for k ∈ {i+1,...,n} do
13             soma ← soma + ALU(i,k) · b(k)
14         b(i) ← y(i) − soma
15     return b
```

7.2 Inversa de uma Matriz por Decomposição LU

Da definição de matriz inversa e da decomposição LU, Eq. (7.1),

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}^{-1}(\mathbf{LU}) = \mathbf{I}$$

tal que se passarmos o produto **LU**para o lado direito da igualdade obteremos

$$\mathbf{A}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}$$

e, do exemplo anterior,

$$\mathbf{L}^{-1} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0.3 & -0.2 & 0 \\ -0.2 & -0.2 & 1 \end{bmatrix} \quad \mathbf{U}^{-1} = \begin{bmatrix} 1 & -2 & 0.5 \\ 0 & 1 & -0.5 \\ 0 & 0 & 1 \end{bmatrix}$$

tal que

$$\begin{bmatrix} 1 & -2 & 0.5 \\ 0 & 1 & -0.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0.3 & -0.2 & 0 \\ -0.2 & -0.2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 1 \\ 3 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

No entanto, não é vantajoso inverter as matrizes **L** e **U** e realizar o produto. Para isto, podemos solucionar uma série de sistemas de equações como fizemos com a eliminação de Gauss, porém usando a decomposição LU, que é mais eficiente.

Exercício 7.1 Tarefa : Propor um algoritmo para inverter uma matriz **A**, utilizando *LU*. ■

7.3 Análise de Complexidade

A solução de sistemas lineares de equações utilizando a decomposição LU é mais eficiente do que a eliminação de Gauss. Do ponto de vista de laços encadeados, ambas abordagens tem complexidade $\mathcal{O}(n^3)$. No entanto, o número efetivo de operações é menor. Não só isso, mas o número de divisões e de multiplicações, que são operações mais custosas, é menor. Comparando, para $n = 100$, o método de Gauss utiliza aproximadamente $0.334n^3$ subtrações, $0.33n^3$ multiplicações e aproximadamente $0.5n^2$ divisões. O método LU utiliza aproximadamente $0.332n^3$ subtrações e multiplicações, $0.5n^2$ divisões e n^2 somas. Essa pequena diferença faz com que o número total de operações em ponto flutuante seja menor no LU quando comparado com a decomposição de Gauss.

A decomposição LU é implementada na biblioteca LAPACK, que é uma extensão da BLAS https://www.netlib.org/utk/papers/factor/node7.html.Essas rotinas são incluidas em bibliotecas de algebra, tais como a MKL.

Em se tratando de matrizes esparsas, existem implementações extremamente eficientes da decomposição LU, como a UMFPACK que é parte da SuiteSparse

8. Decomposição de Cholesky

Quando uma matriz quadrada $\mathbf{A}_{[n\times n]}$ é simétrica ($\mathbf{A} = \mathbf{A}^T$) e positivo definida ($\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$, $\forall \mathbf{x}$), então é possível decompor a matriz na forma

$$\mathbf{A} = \mathbf{U}^T\mathbf{U} \tag{8.1}$$

onde $\mathbf{U}$ é triangular superior[1]. Assim, com o mesmo procedimento realizado na decomposição LU, podemos obter os termos de $\mathbf{U}$ na forma

$$u_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2}$$

para os elementos na diagonal e

$$u_{ij} = \frac{1}{u_{ii}}\left(a_{ij} - \sum_{k=1}^{i-1} u_{ki}u_{kj}\right)$$

para os elementos da triangular superior ($j > i$). É o cálculo da raiz quadrada dos termos da diagonal que necessita da hipótese de que a matriz seja positivo-definida, pois essa hipótese garante que o termo dentro da raiz será sempre positivo. Assim, se considerarmos a Eq. (8.1) no sistema de equações

$$\mathbf{U}^T\mathbf{U}\mathbf{x} = \mathbf{b} \tag{8.2}$$

e se definirmos um problema auxiliar com a forma

$$\mathbf{U}\mathbf{x} = \mathbf{y} \tag{8.3}$$

podemos substituir a Eq. (8.3) na Eq. (8.2), obtendo

$$\mathbf{U}^T\mathbf{y} = \mathbf{b}. \tag{8.4}$$

[1] Diversas referência desenvolvem este método na forma $\mathbf{A} = \mathbf{L}\mathbf{L}^T$, com as mesmas propriedades.

Assim, o procedimento é semelhante ao apresentado na decomposição LU, com a diferença de que podemos armazenar somente a triangular superior da matriz $\mathbf{A}$, o que proporciona uma grande economia de armazenamento e de processamento. As soluções dos problemas definidos nas Eqs. (8.4) e (8.3) são

$$y_i = \frac{1}{u_{ii}} \left(b_i - \sum_{k=1}^{i-1} u_{ki} y_k \right), \; i = 1..n$$

e

$$x_i = \frac{1}{u_{ii}} \left(y_i - \sum_{k=i+1}^{n} u_{ik} x_k \right), \; i = n..1.$$

A decomposição por Cholesky de uma matriz simétrica, sem preocupação com a eficiência em armazenamento, é ilustrada no Algoritmo (28). Utilizando este algoritmo para a matriz

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 10 \end{bmatrix}$$

obtemos

$$\mathbf{A} = \begin{bmatrix} 1.4142 & 0.7071 & 0.7171 \\ 1 & 2.7386 & 0.1825 \\ 1 & 1 & 3.0767 \end{bmatrix}$$

pois alteramos somente a triangular superior, sem modificar a triangular inferior.

8.1 Complexidade

A decomposição Cholesky é apresentada no Alg. 28. A solução de um sistema de equações por Cholesky é ilustrada no Algoritmo (29). O número de operações no método de Cholesky é menor do que usando LU, pois temos aproximadamente o mesmo número de operações de soma e de divisões mas um número menor de subtrações e multiplicações, embora tenhamos n cálculos adicionais de raiz. Salienta-se que, em termos de complexidade (por encadeamento de laços), também temos $\mathcal{O}(n^3)$.

Os algoritmos apresentados neste material são puramente didáticos, sendo que diversas melhorias podem ser realizadas para aumentar a eficiência. Um exemplo é identificarmos que a operação das linhas 10 e 11 em Algo. 28 nada mais é do que um produto interno.

A decomposição Cholesky é implementada de forma eficiente nas bibliotecas LAPACK e na SuiteSparse.

8.2 Decomposição LDL

De modo a evitar o cálculo da raiz na decomposição por Cholesky, é possível modificar a decomposição da matriz de coeficientes para a forma

$$\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^{\mathbf{T}}$$

onde $\mathbf{D}$ é uma matriz diagonal. Com isto, obtemos

$$\begin{aligned} d_{jj} &= a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 d_{kk} \\ l_{ij} &= \frac{1}{d_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} d_{kk} \right), \; i > j. \end{aligned}$$

Algoritmo 28: Decomposição Cholesky de uma matriz $\mathbf{A}$, $n \times n$. A triangular superior (incluindo a diagonal principal) de $\mathbf{A}$ irá conter $\mathbf{U}$. A rotina também calcula o determinante de $\mathbf{A}$.

```
1  Cholesky(A,n)
       Entrada: A matriz n × n
2      Inicializa o determinante
3      dete ← 1
4      Laço pelas linhas da matriz
5      for i ∈ {1,...,n} do
6          Laço pelas colunas da matriz
7          for j ∈ {i,...,n} do
8              Inicializa o somatório
9              soma ← A(i, j)
10             for k ∈ {1,...,i−1} do
11                 soma ← soma − A(k,i) ∗ A(k, j)
12             Verifica se estamos na diagonal principal
13             if j = i then
14                 A(i, j) ← √soma
15             else
16                 A(i, j) ← soma/A(i,i)
17         Produtório do determinante
18         dete ← dete · A(i,i)
19     O determinante será o quadrado do que foi computado no laço acima
20     dete ← dete · dete
21     Retorna a decomposição Cholesky e o determinante
22     return A,dete
```

Algoritmo 29: Solução de um sistema linear de Equações $\mathbf{Ax} = \mathbf{b}$, com n equações. Espera-se que a matriz **CholA** tenha sido obtida com a rotina do Alg. 28.

```
1  SolveCholesky(CholA,b,n)
       Entrada: CholA matriz n × n com a decomposição Cholesky
       Entrada: b vetor n × 1
2      Soluciona para y
3      y ← 0_n
4      for i ∈ {1,...,n} do
5          soma ← 0
6          for k ∈ {1,...,i−1} do
7              soma ← soma + CholA(k,i) · y(k)
8          y(i) ← (b(i) − soma)/CholA(i,i)
9      Soluciona para x, sobre-escrevendo b
10     for i ∈ {n,n−1,...,1} do
11         soma ← 0
12         for k ∈ {i+1,...,n} do
13             soma ← soma + CholA(i,k) · b(k)
14         b(i) ← y(i) − soma
15     return b
```

Exercício 8.1 Tarefa: Deduzir o algoritmo de decomposição LDL e o algoritmo de solução do sistema de equações. Após, implemente e apresente a validação. ■

9. Condicionamento de uma Matriz

Dos algoritmos apresentados nos capítulos anteriores fica claro que para solucionarmos um sistema de equações precisamos realizar uma grande quantidade de operações. Em especial, temos divisões pelos termos da diagonal principal, que não podem ser nulos e nem muito menores do que os demais termos da matriz, para que isto não ocasione erros de truncamento inaceitáveis. Quando uma matriz tem valores de tal forma que a solução numérica do sistema não propague erros inaceitáveis, é dita bem condicionada, do contrário, dizemos que a matriz é mal condicionada. Outra maneira de definir condicionamento é a sensibilidade da solução do sistema a perturbações no vetor $\mathbf{b}$

Seja um sistema linear de equações $\mathbf{Ax} = \mathbf{b}$. Supondo que a solução calculada seja $\bar{\mathbf{x}}$, teremos

$$\mathbf{A}^{-1}\mathbf{b} = \bar{\mathbf{x}}.$$

Agora, imaginando uma perturbação em $\mathbf{b}$ tal que o novo lado direito do sistema seja $\mathbf{b}_\delta$. Com isso, teremos

$$\mathbf{A}^{-1}\mathbf{b}_\delta = \bar{\mathbf{x}}_\delta.$$

Podemos definir o erro relativo causado pela perturbação em $\mathbf{b}_\delta$ como

$$E_\delta = \frac{\frac{\|\bar{\mathbf{x}}_\delta\|}{\|\bar{\mathbf{x}}\|}}{\frac{\|\mathbf{b}_\delta\|}{\|\mathbf{b}\|}} = \frac{\|\bar{\mathbf{x}}_\delta\|}{\|\mathbf{b}_\delta\|}\frac{\|\mathbf{b}\|}{\|\bar{\mathbf{x}}\|} = \frac{\|\mathbf{A}^{-1}\mathbf{b}_\delta\|}{\|\mathbf{b}_\delta\|}\frac{\|\mathbf{b}\|}{\|\mathbf{A}^{-1}\mathbf{b}\|},$$

e o maior erro relativo será obtido quando tivermos os extremos (maiores valores) de cada um dos dois termos. Para o primeiro termo (considerando que $\mathbf{b}_\delta \neq \mathbf{0}$),

$$\max \frac{\|\mathbf{A}^{-1}\mathbf{b}_\delta\|}{\|\mathbf{b}_\delta\|} \approx \max\|\mathbf{A}^{-1}\|$$

e, para o segundo (também assumindo $\mathbf{b} \neq \mathbf{0}$,

$$\max \frac{\|\mathbf{b}\|}{\|\mathbf{A}^{-1}\mathbf{b}\|} \approx \max\|\mathbf{A}\|.$$

Com isso, o maior erro relativo será limitado por cima pelo produto $\|\mathbf{A}^{-1}\|\|\mathbf{A}\|$.

Definição 9.0.1 Condicionamento de uma matriz
O número de condicionamento de uma matriz $\mathbf{A}$ pode ser definido como o limite superior do erro relativo E_δ

$$cond(\mathbf{A}) = \|\mathbf{A}^{-1}\|\|\mathbf{A}\|.$$

■ **Exemplo 9.1** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 2 & 3 & 4 \\ 3 & 9 & 5 \\ 4 & 5 & 3 \end{bmatrix}.$$

Podemos utilizar diferentes normas para calcular o condicionamento. Por exemplo, se utilizarmos a norma 2, teremos um condicionamento de 86.17 e se usarmos normas ∞ teremos um condicionamento de 31.5 A norma 2 é a comumente utilizada. ■

Quando a norma escolhida for a norma 2, podemos utilizar uma propriedade muito interessante, chamada de razão espectral. O raio espectral de uma matriz (ou operador se formos um pouco mais gerais) é definido como o maior dos autovalores em módulo, i.e.,

$$\rho(\mathbf{A}) = \max\{|\lambda_1|, ..., |\lambda_n|\}$$

com

$$\rho(\mathbf{A}) \leq \|\mathbf{A}\|.$$

Quando a norma é a 2, temos que

$$\rho(\mathbf{A}) \approx \|\mathbf{A}\|$$

tal que o condicionamento pode ser escrito por uma razão entre o maior autovalor da matriz, $\rho(\mathbf{A})$ pelo menor autovalor da matriz, $\rho(\mathbf{A}^{-1})$

$$C(\mathbf{A}) = \frac{|\lambda_{max}|}{|\lambda_{min}|}.$$

Com isso, podemos definir o condicionamento 'ideal' como o condicionamento da matriz identidade,

$$C(\mathbf{I}) = 1.$$

■ **Exemplo 9.2** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 2 & 3 & 4 \\ 3 & 9 & 5 \\ 4 & 5 & 3 \end{bmatrix}.$$

A norma 2 da matriz é 13.928 e a norma 2 da sua inversa é 0.7721. O maior autovalor da matriz é 13.67 e o menor autovalor é $|-1.6987|$. O maior autovalor da matriz inversa é $|-0.588|$ que é igual a $1/-1.6987$. Com isso, podemos ver que o número de condicionamento obtido com o produto das normas é $13.928 \cdot 0.7721 = 10.75$ e o obtido com a razão espectral é $13.67/1.6987 = 8.05$. Esses números não são exatamente iguais, mas estão próximos em ordem de grandeza.

■

9.1 Matrizes de Hilbert

Definição 9.1.1 Matriz de Hilbert

Uma matriz de Hilbert $n \times n$ tem a forma

$$H_{ij} = \frac{1}{i+j-1}$$

para $i, j = 1..n$. Essa matriz tem um condicionamento da ordem de

$$C(\mathbf{H}) = \mathcal{O}\left(\frac{\left(1+\sqrt{(2)}\right)^{4n}}{\sqrt{n}}\right).$$

Essas matrizes são extremamente mal condicionadas e servem como exemplo para estudarmos os erros na solução de sistemas de equações lineares.

■ **Exemplo 9.3** Para $n = 4$, temos

$$\mathbf{H} = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

tal que os autovalores extremos desta matriz são $\lambda_{max} = 1.5$ e $\lambda_{min} = 9.67 \times 10^{-5}$ e, portanto,

$$C(\mathbf{H}) = \frac{1.5}{9.67 \times 10^{-5}} = 15513.739 >>>> 1$$

indicando um mal condicionamento.

Para comparação, a matriz

$$\mathbf{A} = \begin{bmatrix} 2 & 4 & 1 \\ 3 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

tem $C(\mathbf{A}) = 6.91$. ■

10. Decomposição QR

Uma das decomposições de matrizes mais utilizadas na álgebra é a decomposição QR. Existem diversas maneiras de obter as matrizes que formam a decomposição, sendo que a mais usual é utilizando a ortogonalização de Gramm-Schmidt (conceito visto na primeira parte da matéria - Fundamentos de Álgebra).

10.1 Ortogonalização de uma matriz

Seja uma matriz quadrada $n \times n$

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_n \end{bmatrix}$$

onde cada coluna j de $\mathbf{A}$ é escrita como um vetor $\mathbf{a}_j$ de dimensão n. Se aplicarmos a ortogonalização de Gramm-Schmidt teremos, para a primeira coluna

$$\mathbf{u_1} = \mathbf{a}_1$$

que normalizado, se torna

$$\mathbf{e_1} = \mathbf{u}_1 / ||\mathbf{u}_1||.$$

Podemos observar a relação (que será importante ao longo desta dedução)

$$\mathbf{0} = \mathbf{a}_1 - \underbrace{\langle \mathbf{a}_1, \mathbf{e}_1 \rangle \mathbf{e}_1}_{\mathbf{a}_1}$$

pois

$$\langle \mathbf{a}_1, \mathbf{e}_1 \rangle \mathbf{e}_1 = \langle \mathbf{a}_1, \mathbf{a}_1 / ||\mathbf{a}_1|| \rangle \mathbf{e}_1 = \left(||\mathbf{a}_1||^2 / ||\mathbf{a}_1|| \right) \mathbf{e}_1 = \mathbf{a}_1,$$

ou seja, $\langle \mathbf{a}_1, \mathbf{e}_1 \rangle = ||\mathbf{a}_1||$.

Ortogonalizando a segunda coluna de $\mathbf{A}$ em relação a primeira

$$\mathbf{u}_2 = \mathbf{a}_2 - \langle \mathbf{a}_2, \mathbf{e}_1 \rangle \mathbf{e}_1 \tag{10.1}$$

que normalizado se torna

$$\mathbf{e_2} = \mathbf{u}_2/||\mathbf{u}_2||.$$

Ainda na segunda coluna, se continuarmos com o mesmo procedimento e subtrairmos o que seria o próximo termo da sequência (neste caso) $\langle \mathbf{a}_2, \mathbf{e}_2 \rangle \mathbf{e}_2$, observamos que

$$\mathbf{0} = \underbrace{\mathbf{a}_2 - \langle \mathbf{a}_2, \mathbf{e}_1 \rangle \mathbf{e}_1}_{\mathbf{u}_2} - \underbrace{\langle \mathbf{a}_2, \mathbf{e}_2 \rangle \mathbf{e}_2}_{\mathbf{u}_2}$$

pois, da Eq. (10.1),

$$\mathbf{a}_2 = \mathbf{u}_2 + \langle \mathbf{a}_2, \mathbf{e}_1 \rangle \mathbf{e}_1$$

tal que, substituindo e usando a distributividade

$$\langle \mathbf{a}_2, \mathbf{e}_2 \rangle \mathbf{e}_2 = \langle \mathbf{u}_2 + \langle \mathbf{a}_2, \mathbf{e}_1 \rangle \mathbf{e}_1, \mathbf{e}_2 \rangle \mathbf{e}_2 = \langle \mathbf{u}_2, \mathbf{e}_2 \rangle \mathbf{e}_2 + \langle \mathbf{a}_2, \mathbf{e}_1 \rangle \langle \mathbf{e}_1, \mathbf{e}_2 \rangle \mathbf{e}_2$$

onde

$$\langle \mathbf{e}_1, \mathbf{e}_2 \rangle = 0$$

devido a ortogonalidade. Com isso, o segundo termo se anula. O primeiro termo pode ser escrito como $\mathbf{u}_2$, pois

$$\langle \mathbf{u}_2, \mathbf{e}_2 \rangle \mathbf{e}_2 = ||\mathbf{u}_2|| \mathbf{e}_2 = \mathbf{u}_2.$$

A terceira coluna deve ser normalizada em relação às duas coluna anteriores, tal que

$$\mathbf{u}_3 = \mathbf{a}_3 - \langle \mathbf{a}_3, \mathbf{e}_1 \rangle \mathbf{e}_1 - \langle \mathbf{a}_3, \mathbf{e}_2 \rangle \mathbf{e}_2$$

que também é normalizada

$$\mathbf{e_3} = \mathbf{u}_3/||\mathbf{u}_3||.$$

Novamente, podemos acrescentar o último termo da sequência, tal que

$$\mathbf{0} = \mathbf{a}_3 - \langle \mathbf{a}_3, \mathbf{e}_1 \rangle \mathbf{e}_1 - \langle \mathbf{a}_3, \mathbf{e}_2 \rangle \mathbf{e}_2 - \langle \mathbf{a}_3, \mathbf{e}_3 \rangle \mathbf{e}_3$$

Seguindo esse procedimento, teremos a recorrência

$$\mathbf{0} = \mathbf{a}_n - \langle \mathbf{a}_n, \mathbf{e}_1 \rangle \mathbf{e}_1 - \langle \mathbf{a}_n, \mathbf{e}_2 \rangle \mathbf{e}_2 - ... - \langle \mathbf{a}_n, \mathbf{e}_n \rangle \mathbf{e}_n$$

com

$$\mathbf{e_n} = \mathbf{u}_n/||\mathbf{u}_n||.$$

Com isso, podemos notar que os vetores $\mathbf{e}_1$ multiplicam, de forma sistemática, $\langle \mathbf{a}_1, \mathbf{e}_1 \rangle$, $\langle \mathbf{a}_2, \mathbf{e}_1 \rangle$, $\langle \mathbf{a}_3, \mathbf{e}_1 \rangle ... \langle \mathbf{a}_n, \mathbf{e}_1 \rangle$. O vetor $\mathbf{e}_2$, por sua vez, multiplica, de forma sistemática 0 (ele não aparece na ortogonalização da primeira coluna), $\langle \mathbf{a}_2, \mathbf{e}_2 \rangle$, $\langle \mathbf{a}_3, \mathbf{e}_2 \rangle ... \langle \mathbf{a}_n, \mathbf{e}_2 \rangle$. Seguindo esse padrão, notamos que $\mathbf{e}_3$ multiplica 0, 0 (pois não participa da ortogonalização das primeiras 2 colunas) e $\langle \mathbf{a}_3, \mathbf{e}_3 \rangle ... \langle \mathbf{a}_n, \mathbf{e}_3 \rangle$. Isso pode ser organizado, então, como

$$\begin{bmatrix} \mathbf{0} & \mathbf{0} & ... & \mathbf{0} \end{bmatrix} = \mathbf{A} - \underbrace{\begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & ... & \mathbf{e}_n \end{bmatrix}}_{\mathbf{Q}} \underbrace{\begin{bmatrix} \langle \mathbf{a}_1, \mathbf{e}_1 \rangle & \langle \mathbf{a}_2, \mathbf{e}_1 \rangle & ... & \langle \mathbf{a}_n, \mathbf{e}_1 \rangle \\ 0 & \langle \mathbf{a}_2, \mathbf{e}_2 \rangle & ... & \langle \mathbf{a}_n, \mathbf{e}_2 \rangle \\ 0 & 0 & ... & \vdots \\ 0 & 0 & ... & \langle \mathbf{a}_n, \mathbf{e}_n \rangle \end{bmatrix}}_{\mathbf{R}}$$

onde a matriz $\mathbf{Q}$ é ortogonal:

Definição 10.1.1 Matriz ortogonal

Seja uma matriz quadrada $\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \dots & \mathbf{q}_n \end{bmatrix}$ onde cada coluna é um vetor de dimensão n. Essa matriz é dita ortogonal se $\langle \mathbf{q}_i, \mathbf{q}_j \rangle = \delta_{ij}, \forall i, j = 1..n$. Assim, se fizermos o produto

$$\mathbf{Q}^T\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1^T & \mathbf{q}_2^T & \dots & \mathbf{q}_n^T \end{bmatrix} \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \dots & \mathbf{q}_n \end{bmatrix} = \mathbf{I}.$$

Relembrando o conceito de matriz inversa,

$$\mathbf{Q}^{-1}\mathbf{Q} = \mathbf{I}$$

verificamos que, para uma matriz ortogonal

$$\mathbf{Q}^{-1} = \mathbf{Q}^T.$$

A outra matriz, $\mathbf{R}$, é triangular superior e consiste em todos os coeficientes que calculamos no processo de ortogonalização. Com isso, verificamos que a matriz $\mathbf{A}$ pode ser escrita como

$$\mathbf{A} = \mathbf{QR}$$

e essa decomposição é única se a matriz $\mathbf{A}$ for não singular.

É importante salientar que o processo de ortogonalização de Gramm-Schmidt é propenso a erros numéricos, piorando a medida que vamos iterando sobre as colunas da matriz. Isso pode ser melhorado com uma alteração relativamente simples do algoritmo, onde ao invés de calcularmos um vetor $\mathbf{u}$ de uma coluna k por meio de

$$\mathbf{u}_k = \mathbf{a}_k - \sum_{j=1}^{k-1} \langle \mathbf{a}_k, \mathbf{e}_j \rangle \mathbf{e}_j$$

que implica em "garantir" a ortogonalização de $\mathbf{u}_k$ em reação a todos os vetores anteriores. Como comentado, erros numéricos podem fazer com que esses vetores não sejam exatamente ortogonais. A alteração consiste em ortogonalizar de forma mais explícita, por meio de mais um loop. Seja a coluna k, iteramos

$$\mathbf{u}_k = \mathbf{u}_k - \langle \mathbf{u}_k, \mathbf{e}_j \rangle \mathbf{e}_j, \ j = 1..k-1$$

com $\mathbf{u}_k = \mathbf{a}_k$ quando $j = 1$. O segredo é que re-utilizamos o $\mathbf{u}_k$ para garantir a ortogonalização a cada j.

■ **Exemplo 10.1** Seja

$$\mathbf{A} = \begin{bmatrix} 2 & 4 & 1 \\ 3 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Começamos com a primeira coluna, $\mathbf{a}_1$ como referência

$$\mathbf{u}_1 = (2,3,1)$$

e, normalizando,

$$\mathbf{e}_1 = (2,3,1)/||(2,3,1)|| = (0.5345, 0.8018, 0.2673)$$

com $||(2,3,1)|| = \sqrt{2^2+3^2+1^2} = 3.742$. Aqui é importante notar que

$$\langle \mathbf{a}_1, \mathbf{e}_1 \rangle = \langle \mathbf{a}_1, \mathbf{a}_1/||\mathbf{a}_1|| \rangle = ||\mathbf{a}_1||^2/||\mathbf{a}_1|| = ||\mathbf{a}_1||.$$

Para segunda coluna

$$\mathbf{u}_2 = (4,1,1) - \langle (4,1,1), (0.5345, 0.8017, 0.2672) \rangle (0.5345, 0.8017, 0.2672)$$

ou

$$\mathbf{u}_2 = (4,1,1) - \underbrace{3.2071}_{\langle \mathbf{a}_2, \mathbf{e}_1 \rangle} (0.5345, 0.8018, 0.2673)$$

tal que

$$\mathbf{u}_2 = (2.2857, -1.5714, 0.1428)$$

e

$$\mathbf{e}_2 = \frac{(2.2857, -1.5714, 0.1428)}{\sqrt{2.2857^2 + (-1.5714)^2 + 0.1428^2}} = (0.823, -0.566, 0.0514).$$

Para zeramos o lado esquerdo referente a segunda coluna, procedemos com o terceiro termo do somatório, que tem como coeficiente

$$\langle \mathbf{a}_2, \mathbf{e}_2 \rangle = 2.777.$$

Por fim, a ortogonalização da terceira coluna de $\mathbf{A}$ é obtida com

$$\begin{aligned} \mathbf{u}_3 = (1,-1,1) &- \langle (1,-1,1), (0.5345, 0.8017, 0.2672) \rangle (0.5345, 0.8017, 0.2672) \\ &- \langle (1,-1,1), (0.823, -0.566, 0.051) \rangle (0.823, -0.566, 0.051) \end{aligned}$$

ou

$$\mathbf{u}_3 = (1,-1,1) - \underbrace{0}_{\langle \mathbf{a}_3, \mathbf{e}_1 \rangle} (0.5345, 0.8017, 0.2672) - \underbrace{1.44}_{\langle \mathbf{a}_3, \mathbf{e}_2 \rangle} (0.823, -0.566, 0.051)$$

tal que

$$\mathbf{u}_3 = (-0.1852, -0.1852, 0.926)$$

e, como $||\mathbf{u}_3|| = 0.962$

$$\mathbf{e}_3 = (-0.19227, -0.1921, 0.9623).$$

O terceiro termo do somatório, que irá zerar a terceira coluna, tem como coeficiente

$$\langle \mathbf{a}_3, \mathbf{e}_3 \rangle = 0.9622.$$

A matriz ortogonal $\mathbf{Q}$ é obtida organizando os $\mathbf{e}_j$ por colunas

$$\mathbf{Q} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \end{bmatrix} = \begin{bmatrix} 0.5345 & 0.823 & -0.19227 \\ 0.8017 & -0.566 & -0.1921 \\ 0.2672 & 0.051 & 0.9623 \end{bmatrix}$$

de onde observamos que $\langle \mathbf{e}_i, \mathbf{e}_j \rangle = 0$ para $i \neq j$ (ortogonal) e $\langle \mathbf{e}_i, \mathbf{e}_i \rangle = 1$ (ortonormal).

A matriz com os coeficiente que utilizamos para as ortogonalizações é

$$\mathbf{R} = \begin{bmatrix} \langle \mathbf{a}_1, \mathbf{e}_1 \rangle & \langle \mathbf{a}_2, \mathbf{e}_1 \rangle & \langle \mathbf{a}_3, \mathbf{e}_1 \rangle \\ 0 & \langle \mathbf{a}_2, \mathbf{e}_2 \rangle & \langle \mathbf{a}_3, \mathbf{e}_2 \rangle \\ 0 & 0 & \langle \mathbf{a}_3, \mathbf{e}_3 \rangle \end{bmatrix} = \begin{bmatrix} ||\mathbf{a}_1|| & \langle \mathbf{a}_2, \mathbf{e}_1 \rangle & \langle \mathbf{a}_3, \mathbf{e}_1 \rangle \\ 0 & \langle \mathbf{a}_2, \mathbf{e}_2 \rangle & \langle \mathbf{a}_3, \mathbf{e}_2 \rangle \\ 0 & 0 & \langle \mathbf{a}_3, \mathbf{e}_3 \rangle \end{bmatrix} = \begin{bmatrix} 3.742 & 3.2071 & 0 \\ 0 & 2.777 & 1.44 \\ 0 & 0 & 0.9622 \end{bmatrix}$$

Com isso, podemos verificar que (o correto é utilizar todas as casas decimais)

$$\mathbf{A} = \begin{bmatrix} 2 & 4 & 1 \\ 3 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0.5345 & 0.823 & -0.19227 \\ 0.8017 & -0.566 & -0.1921 \\ 0.2672 & 0.051 & 0.9623 \end{bmatrix} \begin{bmatrix} 3.742 & 3.2071 & 0 \\ 0 & 2.777 & 1.44 \\ 0 & 0 & 0.9622 \end{bmatrix}$$

■

O Algoritmo para realizar a decomposição QR é apresentado no Alg. 30

10.2 Solução de Sistemas de Equações usando a decomposição QR

Seja o sistema linear de equações

$$\mathbf{Ax} = \mathbf{b}$$

e aplicamos a decomposição QR na matriz de coeficientes, tal que $\mathbf{A} = \mathbf{QR}$. Assim, podemos escrever o sistema como

$$\mathbf{QRx} = \mathbf{b}$$

e, como $\mathbf{Q}$ é ortogonal, então podemos multiplicar ambos os lados por $\mathbf{Q}^T$, tal que

$$\mathbf{Rx} = \mathbf{Q}^T\mathbf{b}$$

e, finalmente, como $\mathbf{R}$ é triangular superior, podemos solucionar o sistema por meio de uma retro substituição. Essa abordagem é particularmente interessante quado a matriz $\mathbf{A}$ é mal condicionada. A complexidade desta operação é da ordem de n^3, não sendo indicada quando o sistema puder ser solucionado por LU ou por Cholesky.

O Algoritmo 31 apresenta os passos para a solução de um sistema de equações utilizando a decomposição QR.

■ **Exemplo 10.2** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 64. & -2016 & 20160 & -92400 & 221760 & -288288 & 192192 & -51480 \\ -2016 & 84672 & -952560 & 4656960 & -11642400 & 15567552 & -10594584 & 2882880 \\ 20160 & -952560 & 11430720 & -58212000 & 1.497 \times 10^8 & -2.043 \times 10^8 & 1.413 \times 10^8 & -38918880 \\ -92400 & 4656960 & -58212000 & 3.049 \times 10^8 & -8.004 \times 10^8 & 1.110 \times 10^9 & -7.769 \times 10^8 & 2.162 \times 10^8 \\ 221760 & -11642400 & 1.497 \times 10^8 & -8.004 \times 10^8 & 2.134 \times 10^9 & -2.997 \times 10^9 & 2.119 \times 10^9 & -5.946 \times 10^8 \\ -288288 & 15567552 & -2.043 \times 10^8 & 1.110 \times 10^9 & -2.997 \times 10^9 & 4.250 \times 10^9 & -3.030 \times 10^9 & 8.562 \times 10^8 \\ 192192 & -10594584 & 1.413 \times 10^8 & -7.769 \times 10^8 & 2.119 \times 10^9 & -3.030 \times 10^9 & 2.175 \times 10^9 & -6.184 \times 10^8 \\ -51480 & 2882880 & -38918880 & 2.162 \times 10^8 & -5.946 \times 10^8 & 8.562 \times 10^8 & -6.184 \times 10^8 & 1.767 \times 10^8 \end{bmatrix}$$

que é a inversa de uma matriz de Hilbert de dimensão 8. Essa matriz tem um número de condicionamento 1.526×10^{10}. Utilizando um vetor de termos independentes, $\mathbf{b}$, com $b_i = i$, obtemos, usando o Scilab

$$\mathbf{x}_1 = \begin{Bmatrix} 0 \\ 0 \\ -0.0031391 \\ -0.0035287 \\ -0.0028351 \\ -0.0018490 \\ -0.0008703 \\ 0 \end{Bmatrix}$$

Algoritmo 30: Decomposição QR de uma matriz $\mathbf{A}$ $n \times n$.

```
1  QR(A,n)
       Entrada: A matriz n × n
2      Alocações iniciais
3      U ← 0_{n×n}
4      Q ← 0_{n×n}
5      R ← 0_{n×n}
6      u ← 0_n
7      a ← 0_n
8      q ← 0_n
9      Primeira coluna
10     for j ∈ {1..n} do
11         u(i) ← A(i,1)
12     Norma da coluna
13     norma ← ||u||_2
14     Primeira posição de R
15     R(1,1) ← norma
16     Primeira coluna de Q
17     for j ∈ {1..n} do
18         Q(j,1) ← u(j)/norma
19     Laço pelas demais colunas
20     for k ∈ {2,...,n} do
21         Coluna atual
22         for j ∈ {1..n} do
23             u(j) ← A(j,k)
24             a(j) ← A(j,k)
25         Ortogonaliza em relação às linhas anteriores
26         Gramm-Schmidt
27         for i ∈ {1,..,k−1} do
28             for j ∈ {1..n} do
29                 q(j) ← Q(j,i)
30             Produto interno
31             proj ← ⟨a,q⟩
32             Ortogonaliza
33             u ← u − proj·q
34             Podemos guardar a projeção em R
35             R(j,k) ← proj
36         Calcula e_k
37         norma ← ||u||_2
38         for j ∈ {1..n} do
39             u(j) ← u(j)/norma
40             Q(j,k) ← u(j)
41         Projeção do ultimo termo
42         proj ← ⟨a,u⟩
43         R(k,k) ← proj
44     return Q,R
```

Algoritmo 31: Solução de um sistema linear de equações $\mathbf{Ax} = \mathbf{b}$, com n equações, utilizando a decomposição QR.

```
1  SolQR(A,b,n)
       Entrada: A matriz n × n
       Entrada: b vetor n × 1
2      Decomposição QR
3      Q,R ← QR(A,n)
4      Calcula Q^T b
5      for i ∈ {1..n} do
6          soma ← 0
7          for j ∈ {1,...,n} do
8              soma ← soma + Q(j,i) · b(j)
9          b(i) ← soma
10     Retro-substituição usando R
11     for i ∈ {n,n−1,...,1} do
12         soma ← 0
13         for k ∈ {i+1,...,n} do
14             soma ← soma + R(i,k) · b(k)
15         b(i) ← (b(i) − soma)/R(i,i)
16     Retorna o resultado no vetor b
17     return b
```

obviamente após recebermos o aviso de que o sistema é mal condicionado. Se testarmos esta solução com

$$\mathbf{Ax}_1 - \mathbf{b} = \begin{Bmatrix} -1.1727528 \\ -0.9367498 \\ -3.4111172 \\ -4.9773307 \\ -5.8545016 \\ -6.3134753 \\ -6.5224791 \\ -6.5805729 \end{Bmatrix}$$

que é diferente de $\mathbf{0}$. Se utilizarmos a decomposição QR e o procedimento apresentado aqui, iremos obter, após a retro substituição

$$\mathbf{x}_2 = \begin{Bmatrix} 8.0000012 \\ 6.1710323 \\ 5.1420638 \\ 4.4403682 \\ 3.9204908 \\ 3.515998 \\ 3.1906261 \\ 2.9223971 \end{Bmatrix}$$

que resulta em

$$\mathbf{A}\mathbf{x}_2 - \mathbf{b} = \left\{ \begin{array}{c} 0.0000014 \\ 5.588 \times 10^{-08} \\ 1.490 \times 10^{-08} \\ 0.0000004 \\ -0.0000002 \\ 0.0000005 \\ 0.0000017 \\ 0.0000004 \end{array} \right\}$$

que é uma solução muito mais próxima do que se espera (lembrando que o sistema é muito mal condicionado).

■

11. Métodos Iterativos

Métodos iterativos para a solução de sistemas de equações lineares partem de uma estimativa para $\mathbf{x}$ e operam por uma redução sistemática do erro de solução, até um determinado critério de convergência. A grande vantagem na utilização de tais métodos está em sua eficiência computacional. No que segue, serão mostrados dois métodos muito utilizados, sendo que existe uma infinidade de métodos iterativos disponíveis na literatura.

11.1 Método de Gauss-Jacobi

Seja um sistema de equações na forma $\mathbf{A}_{[n\times n]}\mathbf{x}_{[n\times 1]} = \mathbf{b}_{[n\times 1]}$. Podemos isolar os termos das diagonais em função dos demais termos na forma

$$x_i^{k+1} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{n} a_{ij}x_j^k\right]_{j\neq i}, \quad k \in \mathbb{N}^*$$

onde iniciamos com um vetor $\mathbf{x}^0$ de modo a obter um vetor $\mathbf{x}^1$ e assim sucessivamente, até que $\left\|\mathbf{x}^{k+1} - \mathbf{x}^k\right\| \leq tol$.

Existem duas condições suficientes para a convergência:

- O sistema deve ser irredutível, isto é, se $\mathbf{x}^0 = \mathbf{0}$, então $x_i^1 = b_i/a_{ii}$, $i = 1..n$;
- A matriz deve ser diagonal-dominante, isto é,

$$|a_{ii}| \geq \sum_{j=1}^{n} \left|a_{ij}\right|_{j\neq i}, \; i = 1..n$$

$$|a_{kk}| > \sum_{j=1}^{n} \left|a_{kj}\right|_{j\neq k}, \; k = 1..n$$

que na prática implica em uma condição onde o módulo dos elementos da diagonal deve ser maior ou igual ao somatório módulos dos elementos restantes da linha.

O método de Gauss-Jacobi está ilustrado no Alg.(32).

Algoritmo 32: Método iterativo de Gauss-Jacobi.

```
1  GaussJacobi(A,b,n,tol,nmax)
       Entrada: A matriz n × n
       Entrada: b vetor n × 1
       Entrada: tol tolerância de parada
       Entrada: nmax número máximo de iterações
2      Vetor auxiliar
3      xp ← 0_n
4      Laço pelas iterações
5      for k ∈ {1,...,nmax} do
6          norma ← 0
7          Laço pelas equações do sistema
8          for i ∈ {1,..,n} do
9              Calcula o somatório
10             soma ← 0
11             for j ∈ {1,..,i−1} do
12                 soma ← soma + A(i, j) · x(j)
13             for j ∈ {i+1,..,n} do
14                 soma ← soma + A(i, j) · x(j)
15             Calcula a estimativa para x_i
16             xp(i) ← (b(i) − soma)/A(i,i)
17             Incrementa a norma
18             norma ← norma + (x(i) − xp(i))^2
19         Verifica a tolerância
20         if √norma ≤ tol then
21             return xp
22         Copia xp para x
23         x ← xp
24     Se chegamos aqui, não atingimos a tolerância especificada
25     escreve "A tolerância não foi atingida"
26     return x
```

11.1.1 Análise do método

Uma outra maneira de definir o método de Gauss-Jacobi é reescrever a matriz $\mathbf{A}$ como

$$\mathbf{A} = \mathbf{D} + (\mathbf{A} - \mathbf{D}),$$

onde $\mathbf{D}$ contém a diagonal de $\mathbf{A}$, tal que

$$\mathbf{D}\mathbf{x}^{k+1} = -(\mathbf{A} - \mathbf{D})\mathbf{x}^k + \mathbf{b}$$

ou

$$\mathbf{x}^{k+1} = -\mathbf{D}^{-1}(\mathbf{A} - \mathbf{D})\mathbf{x}^k + \mathbf{D}^{-1}\mathbf{b},$$

onde fica claro que

$$\mathbf{G} = -\mathbf{D}^{-1}(\mathbf{A} - \mathbf{D})$$

é um operador que representa o método de Gauss-Jacobi. Para que o método não tenha uma amplificação ao longo das iterações (o que iria causar a instabilidade do método), observamos que o maior autovalor (em módulo) de $\mathbf{G}$ deve ser menor ou igual a 1, fato este que é satisfeito se a condições de diagonal dominante for observada.

■ **Exemplo 11.1** Seja

$$\mathbf{A} = \begin{bmatrix} 10 & 1 & 1 \\ 2 & 7 & 0 \\ 1 & 1 & 8 \end{bmatrix} = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 8 \end{bmatrix} + \left(\begin{bmatrix} 10 & 1 & 1 \\ 2 & 7 & 0 \\ 1 & 1 & 8 \end{bmatrix} - \begin{bmatrix} 10 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 8 \end{bmatrix} \right)$$

então o operador que descreve o método de Gauss-Jacobi é

$$\mathbf{G} = -\mathbf{D}^{-1}(\mathbf{A} - \mathbf{D}) = \begin{bmatrix} 0 & -0.1 & -0.1 \\ -0.2857143 & 0 & 0 \\ -0.125 & -0.125 & 0 \end{bmatrix}$$

que tem todos os autovalores menores do que 1. Se $\mathbf{b} = \mathbf{1}$ e iniciarmos o método com $\mathbf{x}_0 = \mathbf{1}$, então

$$\mathbf{x}^1 = -\mathbf{D}^{-1}(\mathbf{A} - \mathbf{D})\mathbf{x}^0 + \mathbf{D}^{-1}\mathbf{b} = \begin{pmatrix} -0.1 & -0.1428571 & -0.125 \end{pmatrix},$$

$$\mathbf{x}^2 = -\mathbf{D}^{-1}(\mathbf{A} - \mathbf{D})\mathbf{x}^1 + \mathbf{D}^{-1}\mathbf{b} = \begin{pmatrix} 0.1267857 & 0.1714286 & 0.1553571 \end{pmatrix},$$

$$\mathbf{x}^3 = -\mathbf{D}^{-1}(\mathbf{A} - \mathbf{D})\mathbf{x}^2 + \mathbf{D}^{-1}\mathbf{b} = \begin{pmatrix} 0.0673214 & 0.1066327 & 0.0877232 \end{pmatrix},$$

$$\mathbf{x}^4 = -\mathbf{D}^{-1}(\mathbf{A} - \mathbf{D})\mathbf{x}^3 + \mathbf{D}^{-1}\mathbf{b} = \begin{pmatrix} 0.0805644 & 0.1236224 & 0.1032557 \end{pmatrix},$$

$$\mathbf{x}^5 = -\mathbf{D}^{-1}(\mathbf{A} - \mathbf{D})\mathbf{x}^4 + \mathbf{D}^{-1}\mathbf{b} = \begin{pmatrix} 0.0773122 & 0.1198387 & 0.0994766 \end{pmatrix}$$

e

$$\mathbf{x}^6 = -\mathbf{D}^{-1}(\mathbf{A} - \mathbf{D})\mathbf{x}^5 + \mathbf{D}^{-1}\mathbf{b} = \begin{pmatrix} 0.0780685 & 0.1207679 & 0.1003561 \end{pmatrix}$$

sendo que mais iterações vão levar a uma melhor acurácia do resultado.

■

11.2 Método de Gauss-Seidel

O método de Gauss Seidel difere do método de Gauss Jacobi apenas pela fórmula de recorrência, onde os valores de $\mathbf{x}^{k+1}$ que já foram calculados são utilizados para computar os $\mathbf{x}^{k+1}$ restantes. Assim

$$x_i^{k+1} = \frac{1}{a_{i,i}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right], \quad k \in \mathbb{N}^*$$

e isto torna a convergência mais rápida. O método é ilustrado no Alg. (33).

Algoritmo 33: Método iterativo de Gauss-Seidel.

```
1  GaussSeidel(A,b,n,tol,nmax)
       Entrada: A matriz n × n
       Entrada: b vetor n × 1
       Entrada: tol tolerância de parada
       Entrada: nmax número máximo de iterações
2      Vetor auxiliar
3      xp ← 0_n
4      Laço pelas iterações
5      for k ∈ {1,...,nmax} do
6          norma ← 0
7          Laço pelas equações do sistema
8          for i ∈ {1,..,n} do
9              Calcula o somatório com xp
10             soma ← 0
11             for j ∈ {1,..,i−1} do
12                 soma ← soma + A(i, j) · x(j)
13             Calcula a segunda parte do somatório
14             for j ∈ {i+1,..,n} do
15                 soma ← soma + A(i, j) · xp(j)
16             Calcula a estimativa para x_i
17             xp(i) ← (b(i) − soma)/A(i,i)
18             Incrementa a norma
19             norma ← norma + (x(i) − xp(i))^2
20         Verifica a tolerância
21         if √norma ≤ tol then
22             return xp
23         Copia xp para x
24         x ← xp
25     Se chegamos aqui, não atingimos a tolerância especificada
26     escreve "A tolerância não foi atingida"
27     return x
```

11.2.1 Análise do método

Uma outra maneira de demonstrar a dedução e implementação deste método consiste em reescrever a matriz $\mathbf{A}$ na forma

$$\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$$

onde $\mathbf{D}$ contém a diagonal de $\mathbf{A}$, $\mathbf{L}$ contém a triangular abaixo da diagonal e $\mathbf{U}$ contém a triangular acima da diagonal (não confundir com as matrizes obtidas por LU). Desta forma, o sistema de equações pode ser escrito como

$$(\mathbf{D} + \mathbf{L})\mathbf{x}^{k+1} = -\mathbf{U}\mathbf{x}^k + \mathbf{b},$$

tal que

$$\mathbf{x}^{k+1} = \mathbf{G}\mathbf{x}^k + \bar{\mathbf{b}}$$

onde

$$\mathbf{G} = -(\mathbf{D} + \mathbf{L})^{-1}\mathbf{U}$$

e

$$\bar{\mathbf{b}} = -(\mathbf{D} + \mathbf{L})^{-1}\mathbf{b}$$

e, se o operador $\mathbf{G}$ tiver seu maior autovalor, em módulo, menor ou igual a um, então não haverá amplificação durante o processo iterativo.

■ **Exemplo 11.2** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 10 & 1 & 1 \\ 2 & 7 & 0 \\ 1 & 1 & 8 \end{bmatrix} = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 8 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Assim, o operador $\mathbf{G}$ é

$$\mathbf{G} = \begin{bmatrix} 0 & -0.1 & -0.1 \\ 0 & 0.0285714 & 0.0285714 \\ 0 & 0.0089286 & 0.0089286 \end{bmatrix}$$

com autovalores

$$\lambda_1 = \lambda_2 = 0 \quad e \quad \lambda_3 = 0.0375,$$

tal que o método irá convergir. De fato, se $\mathbf{b} = \mathbf{1}$, então, iniciando o algoritmo com $\mathbf{x}^0 = \mathbf{1}$, teremos

$$\mathbf{x}^1 = -(\mathbf{D} + \mathbf{L})^{-1}\mathbf{x}^0 + (\mathbf{D} + \mathbf{L})^{-1}\mathbf{b} = \begin{pmatrix} -0.1 & 0.1714286 & 0.1160714 \end{pmatrix},$$

$$\mathbf{x}^2 = -(\mathbf{D} + \mathbf{L})^{-1}\mathbf{x}^1 + (\mathbf{D} + \mathbf{L})^{-1}\mathbf{b} = \begin{pmatrix} 0.07125 & 0.1225 & 0.1007813 \end{pmatrix},$$

$$\mathbf{x}^3 = -(\mathbf{D} + \mathbf{L})^{-1}\mathbf{x}^2 + (\mathbf{D} + \mathbf{L})^{-1}\mathbf{b} = \begin{pmatrix} 0.0776719 & 0.1206652 & 0.1002079 \end{pmatrix}$$

e

$$\mathbf{x}^4 = -(\mathbf{D} + \mathbf{L})^{-1}\mathbf{x}^3 + (\mathbf{D} + \mathbf{L})^{-1}\mathbf{b} = \begin{pmatrix} 0.0776719 & 0.1206652 & 0.1002079 \end{pmatrix},$$

que já apresenta convergência até a quarta casa decimal. É importante salientar esta análise por operadores é utilizada para entendermos as propriedades do método, sendo que o algoritmo iterativo apresentado no Alg. 33 deve ser utilizado na prática.

■

11.3 Métodos baseados em otimização

Métodos iterativos baseados em métodos de otimização costumam ser muito eficientes. A ideia básica é partir de uma função potencial

$$\phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$$

de onde podemos constatar que, se a matriz $\mathbf{A}$ for simétrica e positivo-definida, então existe somente um ponto de mínimo para a função e este ponto é igual a solução do sistema de equações $\mathbf{A}\mathbf{x} = \mathbf{b}$. Isto pode ser verificado utilizando a condição necessária para o mínimo de uma função de várias variáveis, ou seja,

$$\nabla_{\mathbf{x}}\phi(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}$$

e, da condição suficiente, sabemos que a matriz Hessiana (segundas derivadas em relação a $\mathbf{x}$)

$$\mathbf{H} = \mathbf{A}$$

deve simétrica (pois a derivada é um operador linear) e ser positivo-definida (para que a função tenha "concavidade" para baixo, i.e., tenha um mínimo).

■ **Exemplo 11.3** Seja o sistema definido por

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

ou

$$\begin{cases} x_1 + x_2 &= 1 \\ x_1 + 2x_2 &= 1 \end{cases}.$$

Neste caso, observamos que a função ϕ é

$$\phi(\mathbf{x}) = 0.5\,(x_2\,(2x_2 + x_1) + x_1\,(x_2 + x_1)) - x_2 - x_1$$

e, do cálculo, sabemos que o mínimo desta função é obtido quando o gradiente é nulo e a matriz Hessiana é positivo definida. De fato, calculando o gradiente, obtemos

$$\nabla_{\mathbf{x}}\phi(\mathbf{x}) = \begin{Bmatrix} x_1 + x_2 - 1 \\ x_1 + 2x_2 - 1 \end{Bmatrix},$$

ou seja, $\mathbf{A}\mathbf{x} - \mathbf{b}$. A matriz de segundas derivadas é obtida com

$$\mathbf{H} = \begin{bmatrix} \frac{d^2\phi}{dx_1^2} & \frac{d^2\phi}{dx_1 dx_2} \\ \frac{d^2\phi}{dx_2 dx_1} & \frac{d^2\phi}{dx_2^2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} = \mathbf{A}.$$

■

Definição 11.3.1 Direção de descida

Seja uma função $\phi(\mathbf{x})$, diferenciável em relação a $\mathbf{x}$. Seja, também, uma regra de atualização de $\mathbf{x}^k$ com a forma

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta\mathbf{x}^k. \tag{11.1}$$

Para que essa sequência tenha sempre valores decrescentes de ϕ, impomos a condição

$$\phi(\mathbf{x}^{k+1}) \leq \phi(\mathbf{x}^k). \tag{11.2}$$

Partindo dessa última equação, podemos expandir o lado da esquerda por uma Série de Taylor de primeira ordem, de tal forma que

$$\phi(\mathbf{x}^{k+1}) = \phi(\mathbf{x}^k + \Delta\mathbf{x}^k) \approx \phi(\mathbf{x}^k) + \nabla_{\mathbf{x}}\phi(\mathbf{x}^k)^T \Delta\mathbf{x}^k$$

e, inserindo na Eq. (11.2), obtemos

$$\nabla_{\mathbf{x}}\phi(\mathbf{x}^k)^T \Delta\mathbf{x}^k \leq 0. \tag{11.3}$$

Se escrevermos o incremento $\Delta\mathbf{x}^k$ como um passo $\alpha^k \in \mathbb{R}_>$ vezes uma direção $\mathbf{d}^k$

$$\Delta\mathbf{x}^k = \alpha^k \mathbf{d}^k$$

e, usando a Eq. (11.1), observamos que

$$\nabla_{\mathbf{x}}\phi(\mathbf{x}^k)^T \alpha^k \mathbf{d}^k \leq 0 \tag{11.4}$$

indicando que a direção $\mathbf{d}^k$ deve formar um ângulo entre 90 e 270 graus em relação ao vetor gradiente. Qualquer direção $\mathbf{d}$ que satisfizer essa desigualdade levará a um decréscimo **local** da função ϕ.

11.3.1 Método do Gradiente - *Steepest Descent*

O método do gradiente, ou *steepest descent*, consiste em utilizar

$$\mathbf{d}^k = -\nabla_{\mathbf{x}}\phi(\mathbf{x}^k) \tag{11.5}$$

que corresponde a um ângulo de 180 graus entre a direção de descida e o gradiente de ϕ.

Assim, se estamos em uma iteração k, temos que a direção de busca será dada por

$$\mathbf{d}^k = -\nabla_{\mathbf{x}}\phi(\mathbf{x}^k) = -\left(\mathbf{A}\mathbf{x}^k - \mathbf{b}\right),$$

tal que o próximo ponto será

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{d}^k$$

e

$$\phi(\mathbf{x}^{k+1}) = \frac{1}{2}\left(\mathbf{x}^k + \alpha^k\mathbf{d}^k\right)^T \mathbf{A}\left(\mathbf{x}^k + \alpha^k\mathbf{d}^k\right) - \mathbf{b}^T\left(\mathbf{x}^k + \alpha^k\mathbf{d}^k\right).$$

Com isso, igualando a derivada de $\phi(\mathbf{x}^{k+1})$ em relação a α^k a zero, podemos obter o valor do passo α^k que leva ao menor valor de ϕ ao longo da direção $\mathbf{d}^k$

$$\frac{d\phi(\mathbf{x}^{k+1})}{d\alpha^k} = \alpha^k\left(\mathbf{d}^k\right)^T\mathbf{A}\mathbf{d}^k - \left(\mathbf{d}^k\right)^T\left(\mathbf{A}\mathbf{x}^k - \mathbf{b}\right) = \alpha^k\left(\mathbf{d}^k\right)^T\mathbf{A}\mathbf{d}^k - \left(\mathbf{d}^k\right)^T\mathbf{d}^k = 0$$

que leva a

$$\alpha^k = \frac{\left(\mathbf{d}^k\right)^T\mathbf{d}^k}{\left(\mathbf{d}^k\right)^T\mathbf{A}\mathbf{d}^k}.$$

Se o ponto $\mathbf{x}_{k+1}$ for a solução do sistema, então o próximo vetor gradiente será nulo, do contrário, devemos repetir o procedimento até que uma determinada tolerância seja obtida. O procedimento está ilustrado no algoritmo 34.

■ **Exemplo 11.4** Método dos gradientes
Seja o sistema de equações

$$\mathbf{A} = \begin{bmatrix} 10 & 2 & 1 \\ 2 & 5 & 1 \\ 1 & 1 & 7 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 17 \\ 15 \\ 24 \end{Bmatrix}.$$

Se utilizarmos $\mathbf{x}^0 = \mathbf{1}$, obtemos

$$\mathbf{d}^0 = \begin{pmatrix} 4 & 7 & 15 \end{pmatrix}, \quad \left\|\mathbf{d}^0\right\| = 17.03, \quad \mathbf{x}^1 = \begin{pmatrix} 1.4789 & 1.8381, 2.79603 \end{pmatrix}$$
$$\mathbf{d}^1 = \begin{pmatrix} -4.26176 & 0.05532 & 1.1106 \end{pmatrix}, \quad \left\|\mathbf{d}^1\right\| = 4.404, \quad \mathbf{x}_2 = \begin{pmatrix} 1.0196 & 1.84411 & 2.91574 \end{pmatrix}$$
$$\mathbf{d}^2 = \begin{pmatrix} 0.199925, 0.82447 & 0.7260 \end{pmatrix}, \quad \left\|\mathbf{d}^2\right\| = 1.11, \quad \mathbf{x}^3 = \begin{pmatrix} 1.04548 & 1.9508 & 3.0097 \end{pmatrix}$$
$$\mathbf{d}^3 = \begin{pmatrix} -0.3662 & 0.14531 & -0.06419 \end{pmatrix}, \quad \left\|\mathbf{d}^3\right\| = 0.4, \quad \mathbf{x}^4 = \begin{pmatrix} 1.00029 & 1.9687 & 3.0017 \end{pmatrix}$$
$$\mathbf{d}^4 = \begin{pmatrix} 0.0578 & 0.1539 & 0.0185 \end{pmatrix}, \quad \left\|\mathbf{d}^4\right\| = 0.16, \quad \mathbf{x}^5 = \begin{pmatrix} 1.008 & 1.990 & 3.00434 \end{pmatrix}$$

com uma acurácia na segunda casa decimal.

■

Algoritmo 34: Método dos Gradientes.

```
1  SteepestDescent(A,b,n,tol,nmax)
       Entrada: A matriz n × n
       Entrada: b vetor n × 1
       Entrada: tol tolerância de parada
       Entrada: nmax número máximo de iterações
2      Laço pelas iterações
3      for i ∈ {1,...,nmax} do
4          Vetor direção
5          d ← −(Ax − b)
6          Norma do gradiente
7          norma ← ‖d‖_2
8          Critério de parada
9          if norma ≤ tol then
10             return x
11         Passo na direção
12         α ← ⟨d,d⟩ / ⟨d,Ad⟩
13         Incrementa x
14         x ← x + αd
15     Se chegamos aqui, não atingimos a tolerância especificada
16     escreve "A tolerância não foi atingida"
17     return x
```

11.4 Método dos Gradientes Conjugados

O método dos gradientes conjugados é um refinamento dos conceitos envolvidos no método do gradiente. A grande diferença está no fato de utilizarmos uma outra direção diferente do negativo do gradiente, melhorando assim as características de convergência do método.

Um conceito importante é o de direções conjugadas: Seja uma matriz positivo-definida $\mathbf{A}$ e dois vetores $\mathbf{u}$ e $\mathbf{v}$. Esses vetores são ditos conjugados em relação a $\mathbf{A}$ se

$$\mathbf{u}^T\mathbf{A}^T\mathbf{v} = \mathbf{v}^T\mathbf{A}\mathbf{u} = \mathbf{0}.$$

Assim, a cada iteração do método dos Gradientes Conjugados, utilizamos uma direção que seja conjugada com a direção anterior de minimização.

Como na primeira iteração não temos duas direções para conjugar, observamos que o método se torna o método dos gradientes. Após a primeira iteração, impomos a condição de que as direções de busca sejam conjugadas. Isso faz com que escrevamos o método com uma lógica $k-1$ para k, ao contrário do método anterior.

Assim, após a primeira iteração, impomos a condição

$$\left(\mathbf{d}^k\right)^T \mathbf{A}\mathbf{d}^{k-1} = 0$$

sendo que a direção de busca na iteração k é obtida por uma combinação linear

$$\mathbf{d}^k = -\nabla\phi^k + \beta^k\mathbf{d}^{k-1}$$

onde β^{k-1} é um escalar positivo. Este termo é obtido diretamente da condição de conjugação, pois

$$\left(\mathbf{d}^k\right)^T \mathbf{A}\mathbf{d}^{k-1} = 0 \implies \left(-\nabla\phi^k + \beta^k\mathbf{d}^{k-1}\right)^T \mathbf{A}\mathbf{d}^{k-1} = 0$$

resultando em

$$\beta^k = \frac{\left(\nabla\phi^k\right)^T \mathbf{A}\mathbf{d}^{k-1}}{\left(\mathbf{d}^{k-1}\right)^T \mathbf{A}\mathbf{d}^{k-1}}.$$

Assim, o próximo ponto será

$$\mathbf{x}^k = \mathbf{x}^{k-1} + \alpha^k\mathbf{d}^k$$

e, utilizando o mesmo procedimento do método do gradiente, obtemos

$$\alpha^k = \frac{\left(\mathbf{d}^k\right)^T \mathbf{d}^k}{\left(\mathbf{d}^k\right)^T \mathbf{A}\mathbf{d}^k}.$$

O procedimento está ilustrado no algoritmo 35. É importante salientar que este método é muito utilizado em computação de alto desempenho, sendo que existem diversas melhorias e modificações propostas na literatura. Um exemplo é o artigo *Qi Wang, Huaxiang Wang, A modified conjugate gradient method based on the Tikhonov system for computerized tomography (CT), ISA Transactions, Volume 50, Issue 2, April 2011, Pages 256-261.*

■ **Exemplo 11.5** Método dos gradientes conjugados

Seja o sistema de equações

$$\mathbf{A} = \begin{bmatrix} 10 & 2 & 1 \\ 2 & 5 & 1 \\ 1 & 1 & 7 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 17 \\ 15 \\ 24 \end{Bmatrix}.$$

Se utilizarmos $\mathbf{x}^0 = \mathbf{1}$, obtemos

Algoritmo 35: Método dos Gradientes Conjugados.

```
GradConj(A,b,n,tol,nmax)
    Entrada: A matriz n × n
    Entrada: b vetor n × 1
    Entrada: tol tolerância de parada
    Entrada: nmax número máximo de iterações
    Define o vetor com a direção anterior
    a ← 0_n
    Laço pelas iterações
    for i ∈ {1,...,nmax} do
        Vetor gradiente
        g ← (Ax − b)
        Norma do gradiente
        norma ← ||g||_2
        Critério de parada
        if norma ≤ tol then
            return x
        Calcula β
        β ← 0
        if i > 1 then
            β ← ⟨g,Aa⟩ / ⟨a,Aa⟩
        Direção de busca
        d ← −g + βa
        Passo na direção
        α ← ⟨d,d⟩ / ⟨d,Ad⟩
        Incrementa x
        x ← x + αd
        Copia d para a
        a ← d
    Se chegamos aqui, não atingimos a tolerância especificada
    escreve "A tolerância não foi atingida"
    return x
```

$$\mathbf{d}^0 = \begin{pmatrix} 4 & 7 & 15 \end{pmatrix}, \quad \left\|\mathbf{d}^0\right\| = 17.03, \quad \mathbf{x}^1 = \begin{pmatrix} 1.4789 & 1.8381, 2.79603 \end{pmatrix}$$
$$\mathbf{d}^1 = \begin{pmatrix} -3.9942 & 0.52358 & 2.11406 \end{pmatrix}, \quad \left\|\mathbf{d}^1\right\| = 4.404, \quad \mathbf{x}_2 = \begin{pmatrix} 0.99022 & 1.9022 & 3.0547 \end{pmatrix}$$
$$\mathbf{d}^2 = \begin{pmatrix} -0.077177 & 0.495167 & -0.108245 \end{pmatrix}, \quad \left\|\mathbf{d}^2\right\| = 0.58, \quad \mathbf{x}^3 = \begin{pmatrix} 0.97218 & 2.018 & 3.02939 \end{pmatrix}$$
$$\mathbf{d}^3 = \begin{pmatrix} 0.2093 & -0.04126 & -0.2008 \end{pmatrix}, \quad \left\|\mathbf{d}^3\right\| = 0.29, \quad \mathbf{x}^4 = \begin{pmatrix} 1.0008 & 2.0123 & 3.0019 \end{pmatrix}$$
$$\mathbf{d}^4 = \begin{pmatrix} -0.0245 & -0.0673 & -0.03648 \end{pmatrix}, \quad \left\|\mathbf{d}^4\right\| = 0.07, \quad \mathbf{x}^5 = \begin{pmatrix} 0.9978 & 2.003 & 2.998 \end{pmatrix}$$

com uma acurácia na segunda casa decimal. Pode-se notar que a convergência da norma é mais rápida do que no método do gradiente.

■

11.5 Pré-Condicionamento

Os métodos iterativos são muito sensíveis ao condicionamento da matriz de coeficientes. Uma maneira de melhorar o condicionamento consiste em modificar o sistema linear por meio de um escalonamento.

Considere a operação

$$\mathbf{x} = \mathbf{D}\tilde{\mathbf{x}} \tag{11.6}$$

em que $\mathbf{D}$ é uma matriz $n \times n$. Utilizando essa mudança de variável no sistema linear, obtemos

$$\mathbf{AD}\tilde{\mathbf{x}} = \mathbf{b}$$

e, se multiplicarmos novamente por $\mathbf{D}$, obtemos

$$\mathbf{DAD}\tilde{\mathbf{x}} = \mathbf{Db}.$$

Existem diversas possibilidades para a matriz $\mathbf{D}$, sendo que a mais simples é utilizar o escalonamento diagonal, onde a matriz $\mathbf{D}$ assume a forma $D_{ii} = \sqrt{A_{ii}}$.

■ **Exemplo 11.6** Considere o sistema $\mathbf{Ax} = \mathbf{b}$

$$\begin{bmatrix} 1 & 10 & 2000 \\ 10 & 25 & 40 \\ 2000 & 40 & 9 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 6021 \\ 180 \\ 2107 \end{Bmatrix}. \tag{11.7}$$

A matriz de escalonamento diagonal será

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 3 \end{bmatrix}, \tag{11.8}$$

tal que a matriz de coeficientes modificada será

$$\tilde{\mathbf{A}} = \mathbf{DAD} = \begin{bmatrix} 1 & 50 & 6000 \\ 50 & 625 & 600 \\ 6000 & 600 & 81 \end{bmatrix} \tag{11.9}$$

e o vetor de coeficientes modificado será

$$\tilde{\mathbf{b}} = \mathbf{Db} = \begin{Bmatrix} 6021 \\ 900 \\ 6321 \end{Bmatrix}.$$

Se calcularmos o condicionamento de ambas as matrizes pela razão espectral, observamos que

$$C(\mathbf{A}) = \frac{|\lambda_{max}|}{|\lambda_{min}|} = 81.56$$

e

$$C(\tilde{\mathbf{A}}) = \frac{|\lambda_{max}|}{|\lambda_{min}|} = 9.98.$$

Sendo assim, vamos considerar a solução do sistema de equações modificado por $\mathbf{D}$

$$\begin{bmatrix} 1 & 50 & 6000 \\ 50 & 625 & 600 \\ 6000 & 600 & 81 \end{bmatrix} \begin{Bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \end{Bmatrix} = \begin{Bmatrix} 6021 \\ 900 \\ 6321 \end{Bmatrix}$$

com solução $\tilde{\mathbf{x}} = \begin{pmatrix} 1 & 0.4 & 1 \end{pmatrix}$. Com isso, podemos obter $\mathbf{x}$ utilizando a Eq. (11.6), tal que

$$\mathbf{x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 3 \end{bmatrix} \begin{Bmatrix} 1 \\ 0.4 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 2 \\ 3 \end{Bmatrix}$$

o resultado correto. ■

Exercício 11.1 Solucione um sistema de dimensão $n = 10$ cuja matriz de coeficientes é uma matriz de Hilbert e o vetor $\mathbf{b}$ é um vetor de sua escolha. Utilize todos os métodos de solução do sistema de equações e verifique a sensibilidade do sistema ao número de condicionamento da matriz. ■

12. Sistemas de Equações Lineares Complexas

Os números complexos, $\mathbb{C}$, são uma extensão dos números reais, tal que $\mathbb{R} \subset \mathbb{C}$. Se considerarmos que um polinômio com coeficientes reais pode ter raízes complexas, podemos entender que o conjunto de números complexos *completa* o conjunto dos números reais.

Um número complexo é representado na forma $z = z_r + iz_c$ com z_r e z_c **pertencentes aos reais** e $i^2 = -1$. Chamamos z_r de parte real e z_c de parte imaginária, ou complexa de z.

Se as grandezas que estão envolvidas na definição do problema forem complexas, então o sistema de equações lineares terá a forma

$$(\mathbf{A}_r + \mathbf{A}_c i)(\mathbf{x}_r + \mathbf{x}_c i) = (\mathbf{b}_r + \mathbf{b}_c i)$$

de tal forma que, usando a propriedade de distributividade, podemos re-escrever o sistema na forma

$$(\mathbf{A}_r\mathbf{x}_r - \mathbf{A}_c\mathbf{x}_c) + (\mathbf{A}_c\mathbf{x}_r + \mathbf{A}_r\mathbf{x}_c)\, i = (\mathbf{b}_r + \mathbf{b}_c i)$$

pois $i^2 = -1$. Assim, podemos reagrupar o sistema na forma

$$\begin{bmatrix} \mathbf{A}_r & -\mathbf{A}_c \\ \mathbf{A}_c & \mathbf{A}_r \end{bmatrix} \begin{Bmatrix} \mathbf{x}_r \\ \mathbf{x}_c \end{Bmatrix} = \begin{Bmatrix} \mathbf{b}_r \\ \mathbf{b}_c \end{Bmatrix}$$

que terá o dobro da dimensão do sistema original, mas que será composto somente por números reais. Esse sistema pode ser solucionado por qualquer método capaz de lidar com números reais e solucionar um sistema com matriz de coeficientes não simétrica e não positivo definida.

Métodos como a decomposição LU e a triangularização de Gauss também podem ser utilizados sem que tenhamos que aumentar o sistema, desde que tenhamos o cuidado de definir as variáveis como complexas. Por exemplo, no Julia, podemos utilizar a flexibilidade que a linguagem tem de lidar com tipos parametrizados (Listagem 12.1)

Listing 12.1: Exemplo de solução de um sistema de equações não lineares utilizando LU e parametrização de tipo na linguagem Julia

```
# Decomp da matriz A de dimen n x n
# utilizando LU. A matriz é modificada e o
# determinante é retornado.
#
function LU!(A::Matrix{T},n) where T
    # Inicializa o determinante com o mesmo tipo de A
    dete = one(T)
    # Loop pelas linhas da matriz
    for i=1:n
        # Loop pelas colunas da matriz
        for j=1:n
           # Verifica se estamos acima ou abaixo da
           # diagonal principal
           c = ifelse(j<i,j,i)
           # Inicializa a soma
           soma = A[i,j]
           # Loop do somatório
           for k=1:c-1
               soma = soma - A[i,k]*A[k,j]
           end
           # Posiciona o resultado na matriz A
           if j<=i
              A[i,j] = soma
            else
              A[i,j] = soma/A[i,i]
            end
        end
        # Atualiza o produtório do determinante
        dete = dete*A[i,i]
    end
    # Retorna o determinante
    return dete
end

#
# Soluciona o sistema Ax=b utilizando a decomp
# LU. A matriz A é modificada e o resultado é armazenado
# no vetor b
#
function SolLU!(A::Matrix{T},b::Vector{T},n) where T

    # Decomp LU
    dete = LU!(A,n)
    # Inicializa o vetor y com o mesmo tipo da matriz A
    y = zeros(T,n)
    # Subst direta
    for i=1:n
        soma = zero(T)
        for k=1:i-1
            soma =  soma + A[i,k]*y[k]
        end
        y[i] = ( b[i] - soma ) / A[i,i]
    end
    # Retro subst
    for i=n:-1:1
        soma = zero(T)
        for k=i+1:n
            soma = soma + A[i,k]*b[k]
        end
        b[i] = y[i] - soma
    end
    # Retorna o determinante de A
    return dete
end
```

■ **Exemplo 12.1** Sejam o sistema de duas equações lineares

$$\begin{array}{l}(3+2i)a+(1-4i)b=7+5i\\(-1+2i)a+(10+8i)b=25\end{array}.$$

Podemos identificar os seguintes elementos:

$$\mathbf{A}_r=\begin{bmatrix}3 & 1\\ -1 & 10\end{bmatrix},$$

$$\mathbf{A}_c=\begin{bmatrix}2 & -4\\ 2 & 8\end{bmatrix},$$

$$\mathbf{b}_r=\begin{Bmatrix}7\\ 25\end{Bmatrix}$$

e

$$\mathbf{b}_c=\begin{Bmatrix}5\\ 0\end{Bmatrix}.$$

As incógnitas a e b também devem ser complexas, com expressões $a=a_r+a_ci$ e $b=b_r+b_ci$, de tal forma que

$$\mathbf{x}_r=\begin{Bmatrix}a_r\\ b_r\end{Bmatrix}$$

e

$$\mathbf{x}_c=\begin{Bmatrix}a_c\\ b_c\end{Bmatrix}.$$

Assim, o sistema complexo assume a forma

$$\begin{bmatrix}3 & 1 & -2 & 4\\ -1 & 10 & -2 & -8\\ 2 & -4 & 3 & 1\\ 2 & 8 & -1 & 10\end{bmatrix}\begin{Bmatrix}a_r\\ b_r\\ a_c\\ b_c\end{Bmatrix}=\begin{Bmatrix}7\\ 25\\ 5\\ 0\end{Bmatrix},$$

com solução $a_r=5.266$, $b_r=1.475$, $a_c=0.838$ e $b_c=-2.149$. Verificando a primeira equação:

$$(3+2i)(5.266+0.838i)+(1-4i)(1.475-2.149i)=7+5i$$

pois $(3+2i)(5.266+0.838i)=14.122+13,048i$ e $(1-4i)(1,475-2,149i)=-7,122-8,048i$, resultando em $7+5i$. Da mesma forma, a segunda equação resulta em:

$$(-1+2i)(5.266+0.838i)+(10+8i)(1.475-2.149i)=25$$

pois $(-1+2i)(5.266+0.838i)=-6.944+9.695i$ e $(10+8i)(1.475-2.149i)=31.944-9.695i$.

■

13. Solução de Sistemas Não-Lineares

Os procedimentos apresentados até agora dizem respeito a solução de problemas lineares, ou seja, é possível obter uma matriz de coeficientes e um vetor que são independentes dos valores das variáveis **x**. No entanto, diversos problemas de engenharia são descritos por equações não-lineares, de tal forma que é importante entendermos os procedimentos de solução para esta classe de problemas.

A forma geral de um sistema de equações não-lineares é

$$\begin{cases} f_1(x_1, x_2, ..., x_n) = 0 \\ \quad\quad\quad ... \\ \quad\quad\quad ... \\ f_n(x_1, x_2, ..., x_n) = 0 \end{cases}$$

onde cada uma das equações pode ser aproximada no entorno de um ponto $\mathbf{x}^k$ por

$$f_i(x_1, x_2, ..., x_n) \simeq f_i(x_1^k, x_2^k, ..., x_n^k) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j^k} \Delta x_j, \;\; j = 1..n$$

onde $\Delta x_i = x_i - x_i^k$. Assim, podemos considerar que no entorno do ponto

$$\sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j^k} \Delta x_j = -f_i(x_1^k, x_2^k, ..., x_n^k), \;\; j = 1..n$$

ou, matricialmente

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1^k} & \cdots & \frac{\partial f_1}{\partial x_n^k} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1^k} & \cdots & \frac{\partial f_n}{\partial x_n^k} \end{bmatrix} \begin{Bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_n \end{Bmatrix} = - \begin{Bmatrix} f_1(\mathbf{x}^k) \\ \vdots \\ f_n(\mathbf{x}^k) \end{Bmatrix}. \tag{13.1}$$

Na forma compacta podemos escrever a Eq. (13.1) como

$$\mathbf{JY} = \mathbf{B}$$

onde a matriz é conhecida como matriz Jacobiana do sistema. Assim, de posse de uma estimativa $\mathbf{x}^k$, podemos obter a nova estimativa como

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{Y}^k$$

até que $\|\mathbf{Y}\| < tol$. Esta é a extensão do Método de Newton-Raphson para múltiplas variáveis, tendo portanto as mesmas propriedades.

■ **Exemplo 13.1** Dado o sistema de equações

$$\begin{cases} x_1 + 2x_2 + x_3 + 4x_4 - 20.7 = 0 \\ x_1^2 + 2x_1x_2 + x_4^3 - 15.88 = 0 \\ x_1^3 + x_3^3 + x_4 - 21.218 = 0 \\ 3x_2 + x_3x_4 - 21.11 = 0 \end{cases}$$

temos que

$$\mathbf{J} = \begin{bmatrix} 1 & 2 & 1 & 4 \\ 2x_2 + 2_{x1} & 2x_1 & 0 & 3x_4^2 \\ 3x_1^2 & 0 & 3x_3^2 & 1 \\ 0 & 3 & x_4 & x_3 \end{bmatrix}$$

e, partindo do ponto $\mathbf{x}^0 = \mathbf{1}$, temos

$$\begin{bmatrix} 1 & 2 & 1 & 4 \\ 4 & 2 & 0 & 3 \\ 3 & 0 & 3 & 1 \\ 0 & 3 & 1 & 1 \end{bmatrix} \mathbf{Y}^0 = \begin{Bmatrix} 12.7 \\ 11.88 \\ 18.218 \\ 17.11 \end{Bmatrix}$$

tal que

$$\mathbf{x}^1 = \begin{Bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{Bmatrix} + \begin{Bmatrix} 1.23285 \\ 4.19175 \\ 5.00045 \\ -0.4819 \end{Bmatrix} = \begin{Bmatrix} 2.23285 \\ 5.19175 \\ 6.00045 \\ 0.52001 \end{Bmatrix}$$

e, na segunda iteração, teremos

$$\begin{bmatrix} 1 & 2 & 1 & 4 \\ 14.8533 & 4.4663 & 0 & 0.81151 \\ 14.9613 & 0 & 107.9802 & 1 \\ 0 & 3 & 0.520099 & 5.99945 \end{bmatrix} \mathbf{Y}^0 = \begin{Bmatrix} 2.6645 \times 10^{-15} \\ -12.44379 \\ -206.3798 \\ 2.39923 \end{Bmatrix}.$$

■

Exercício 13.1 Termine de solucionar o problema, até uma tolerância de 1×10^{-6}.

■

14. Problema de Mínimos Quadrados

Seja uma função $f(\mathbf{x})$ de n variáveis, a ser determinada a partir de uma série de pontos experimentais, avaliados em um conjunto discreto de n_e pontos $\bar{\mathbf{x}}$. Se os valores experimentais nos pontos $\bar{\mathbf{x}}$ forem organizados em um vetor $\mathbf{V}_{[n_e \times 1]}$, então podemos definir o erro médio quadrático entre os valores da função e os valores experimentais na forma

$$\Phi = \sqrt{\sum_{i=1}^{n_{exp}} \left(f(\bar{\mathbf{x}}_i) - V_i\right)^2}.$$

Supondo agora que a função seja descrita por um polinômio de ordem m, na forma

$$f(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})\mathbf{a}$$

onde

$$\mathbf{p}(\mathbf{x}) = \begin{pmatrix} 1 & x & x^2 & .... & x^m \end{pmatrix}$$

é um vetor de dimensão $m+1$ e $\mathbf{a}$ é um vetor de coeficientes a determinar pelo procedimento de ajuste.

Assim,

$$\Phi = \sqrt{\sum_{i=1}^{n_e} \left(\mathbf{p}^T(\bar{\mathbf{x}}_i)\mathbf{a} - V_i\right)^2}$$

e a condição necessária o para mínimo desta função é

$$\nabla_{\mathbf{a}}\Phi = \mathbf{0}.$$

Estudando a derivada em relação a um coeficiente a_j, onde $0 \leqslant j \leqslant m$

$$\frac{d\Phi}{da_j} = \frac{1}{2}\left(\sum_{i=1}^{n_e}\left(\mathbf{p}^T(\bar{\mathbf{x}}_i)\mathbf{a} - V_i\right)^2\right)^{-1/2} \sum_{i=1}^{n_e} 2\left(\mathbf{p}^T(\bar{\mathbf{x}}_i)\mathbf{a} - V_i\right) p_j(\bar{\mathbf{x}}_i) = 0$$

e, como estamos igualando a zero, podemos descartar o termo elevado a $-1/2$, tal que

$$\frac{d\Phi}{da_j} = \sum_{i=1}^{n_e} \left(\mathbf{p}^T(\bar{\mathbf{x}}_i)\mathbf{a} - V_i\right) p_j(\bar{\mathbf{x}}_i) = 0$$

ou

$$\frac{d\Phi}{da_j} = \sum_{i=1}^{n_e} \left(\mathbf{p}^T(\bar{\mathbf{x}}_i)\mathbf{a}\right) p_j(\bar{\mathbf{x}}_i) = \sum_{i=1}^{n_e} V_i p_j(\bar{\mathbf{x}}_i).$$

Desenvolvendo esta expressão obteremos um sistema de equações lineares na forma

$$\begin{bmatrix} \sum_{i=1}^{n_e} p_0(\bar{\mathbf{x}}_i)p_0(\bar{\mathbf{x}}_i) & \sum_{i=1}^{n_e} p_1(\bar{\mathbf{x}}_i)p_0(\bar{\mathbf{x}}_i) & \dots & \sum_{i=1}^{n_e} p_m(\bar{\mathbf{x}}_i)p_0(\bar{\mathbf{x}}_i) \\ \sum_{i=1}^{n_e} p_0(\bar{\mathbf{x}}_i)p_1(\bar{\mathbf{x}}_i) & \sum_{i=1}^{n_e} p_1(\bar{\mathbf{x}}_i)p_1(\bar{\mathbf{x}}_i) & \dots & \sum_{i=1}^{n_e} p_m(\bar{\mathbf{x}}_i)p_1(\bar{\mathbf{x}}_i) \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{n_e} p_0(\bar{\mathbf{x}}_i)p_m(\bar{\mathbf{x}}_i) & \sum_{i=1}^{n_e} p_1(\bar{\mathbf{x}}_i)p_m(\bar{\mathbf{x}}_i) & \dots & \sum_{i=1}^{n_e} p_m(\bar{\mathbf{x}}_i)p_m(\bar{\mathbf{x}}_i) \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{n_e} p_0(\bar{\mathbf{x}})V_i \\ \sum_{i=1}^{n_e} p_1(\bar{\mathbf{x}})V_i \\ \vdots \\ \sum_{i=1}^{n_e} p_m(\bar{\mathbf{x}})V_i \end{Bmatrix}$$

ou

$$\mathbf{A}\mathbf{a} = \mathbf{b}$$

onde

$$\mathbf{A} = \sum_{i=1}^{n_e} \mathbf{p}(\bar{\mathbf{x}}_i)\mathbf{p}^T(\bar{\mathbf{x}}_i)$$

e

$$\mathbf{b} = \sum_{i=1}^{n_e} \mathbf{p}(\bar{\mathbf{x}}_i)V_i.$$

■ **Exemplo 14.1** Seja a função a aproximar $f(x_1,x_2) = a_0 + a_1x_1 + a_2x_2 + a_3x_1^2 + a_4x_2^2$ e os dados experimentais

$\bar{x}_1$	0	1	1	0	2
$\bar{x}_2$	0	1	0	1	2
V	3	-5	11	-13	-25

.

Obtenha os coeficientes que melhor aproximam a função aos dados experimentais.
O polinômio f pode ser representado por

$$f(x_1,x_2) = \mathbf{p}^T(\mathbf{x})\mathbf{a} = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_2^2 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{Bmatrix}$$

tal que

$$\begin{aligned} \Phi &= \Big\{ \left(a_0 + a_1 0 + a_2 0 + a_3 0^2 + a_4 0^2 - 3\right)^2 + \left(a_0 + a_1 1 + a_2 1 + a_3 1^2 + a_4 1^2 - (-5)\right)^2 + \dots \\ &\quad + \left(a_0 + a_1 2 + a_2 2 + a_3 2^2 + a_4 2^2 - (-25)\right)^2 \Big\}^{\frac{1}{2}} \end{aligned}$$

e, se calcularmos o gradiente em relação aos coeficientes do polinômio obteremos

$$\begin{aligned}\frac{d\Phi}{da_0} &= \big\{\big(a_0+a_1 0+a_2 0+a_3 0^2+a_4 0^2-3\big)+\big(a_0+a_1 1+a_2 1+a_3 1^2+a_4 1^2-(-5)\big)+\ldots\\ &\quad+\big(a_0+a_1 2+a_2 2+a_3 2^2+a_4 2^2-(-25)\big)\big\}=0\end{aligned}$$

$$\begin{aligned}\frac{d\Phi}{da_1} &= \big\{\big(a_0+a_1 0+a_2 0+a_3 0^2+a_4 0^2-3\big)*0+\big(a_0+a_1 1+a_2 1+a_3 1^2+a_4 1^2-(-5)\big)*1+\ldots\\ &\quad+\big(a_0+a_1 2+a_2 2+a_3 2^2+a_4 2^2-(-25)\big)*2\big\}=0\end{aligned}$$

.....

$$\begin{aligned}\frac{d\Phi}{da_4} &= \big\{\big(a_0+a_1 0+a_2 0+a_3 0^2+a_4 0^2-3\big)*0^2+\big(a_0+a_1 1+a_2 1+a_3 1^2+a_4 1^2-(-5)\big)*1^2+\ldots\\ &\quad+\big(a_0+a_1 2+a_2 2+a_3 2^2+a_4 2^2-(-25)\big)*2^2\big\}=0\end{aligned}$$

e, colocando os coeficientes em evidência, obtemos

$$\begin{aligned}\frac{d\Phi}{da_0} &= a_0\sum_{i=1}^{5}1+a_1\sum_{i=1}^{5}\bar{x}_1(i)+a_2\sum_{i=1}^{5}\bar{x}_2(i)+a_3\sum_{i=1}^{5}\bar{x}_1^2(i)+a_4\sum_{i=1}^{5}\bar{x}_2^2(i)=\sum_{i=1}^{5}V(i)\\ \frac{d\Phi}{da_1} &= a_0\sum_{i=1}^{5}\bar{x}_1(i)+a_1\sum_{i=1}^{5}\bar{x}_1^2(i)+a_2\sum_{i=1}^{5}\bar{x}_2(i)\bar{x}_1(i)+a_3\sum_{i=1}^{5}\bar{x}_1^2(i)\bar{x}_1(i)+a_4\sum_{i=1}^{5}\bar{x}_2^2(i)\bar{x}_1(i)=\sum_{i=1}^{5}V(i)\bar{x}_1(i)\\ &\ldots\\ \frac{d\Phi}{da_4} &= a_0\sum_{i=1}^{5}\bar{x}_2^2(i)+a_1\sum_{i=1}^{5}\bar{x}_1(i)\bar{x}_2^2(i)+a_2\sum_{i=1}^{5}\bar{x}_2(i)\bar{x}_2^2(i)+a_3\sum_{i=1}^{5}\bar{x}_1^2(i)\bar{x}_2^2(i)+a_4\sum_{i=1}^{5}\bar{x}_2^2(i)\bar{x}_2^2(i)=\sum_{i=1}^{5}V(i)\bar{x}_2^2(i)\end{aligned}$$

tal que

$$\begin{bmatrix}\sum_{i=1}^{5}1 & \sum_{i=1}^{5}\bar{x}_1(i) & \sum_{i=1}^{5}\bar{x}_2(i) & \sum_{i=1}^{5}\bar{x}_1^2(i) & \sum_{i=1}^{5}\bar{x}_2^2(i)\\ & \sum_{i=1}^{5}\bar{x}_1^2(i) & \sum_{i=1}^{5}\bar{x}_1(i)\bar{x}_2(i) & \sum_{i=1}^{5}\bar{x}_1^3(i) & \sum_{i=1}^{5}\bar{x}_2(i)\bar{x}_1(i)\\ & & \sum_{i=1}^{5}\bar{x}_2^2(i) & \sum_{i=1}^{5}\bar{x}_1^2(i)\bar{x}_2(i) & \sum_{i=1}^{5}\bar{x}_2^3(i)\\ & \text{sim.} & & \sum_{i=1}^{5}\bar{x}_1^4(i) & \sum_{i=1}^{5}\bar{x}_2^2(i)\bar{x}_1^2(i)\\ & & & & \sum_{i=1}^{5}\bar{x}_2^4(i)\end{bmatrix}\begin{Bmatrix}a_0\\a_1\\a_2\\a_3\\a_4\end{Bmatrix}=\begin{Bmatrix}\sum_{i=1}^{5}V(i)\\ \sum_{i=1}^{5}V(i)\bar{x}_1(i)\\ \sum_{i=1}^{5}V(i)\bar{x}_2(i)\\ \sum_{i=1}^{5}V(i)\bar{x}_1^2(i)\\ \sum_{i=1}^{5}V(i)\bar{x}_2^2(i)\end{Bmatrix}$$

onde somente a triangular superior da matriz é mostrada, pois a mesma é simétrica. Assim, com os dados deste problema, obteremos o sistema

$$\begin{bmatrix}5 & 4 & 4 & 6 & 6\\ & 6 & 5 & 10 & 9\\ & & 6 & 9 & 10\\ & \text{sim.} & & 18 & 17\\ & & & & 18\end{bmatrix}\begin{Bmatrix}a_0\\a_1\\a_2\\a_3\\a_4\end{Bmatrix}=\begin{Bmatrix}-29\\-44\\-68\\-94\\-118\end{Bmatrix}$$

tal que

$$\begin{Bmatrix}a_0\\a_1\\a_2\\a_3\\a_4\end{Bmatrix}=\begin{Bmatrix}3\\6\\-8\\2\\-8\end{Bmatrix}$$

ou seja, o polinômio que melhor se ajusta aos dados é

$$p(x_1,x_2)=3+6x_1-8x_2+2x_1^2-8x_2^2.$$

■

14.0.1 Escalonamento das Variáveis

É comum lidarmos com uma matriz de coeficientes mal condicionada, principalmente quando as variáveis envolvidas no problema de mínimos quadrados apresentam ordem de grandeza muito diferentes. Neste caso, a melhor estratégia consiste em trabalhar com um novo conjunto de variáveis normalizadas.

Se uma variável original x é definida no intervalo $[x_{min}, x_{max}]$ e queremos que a variável passe a ter valores no intervalo $[r_{min}, r_{max}]$, podemos definir o mapeamento

$$r = r_{min} + (x - x_{min}) \left(\frac{r_{max} - r_{min}}{x_{max} - x_{min}} \right).$$

Desta forma, podemos fazer com que todas as variáveis apresentem a mesma ordem de grandeza, reduzindo muito a chance de observarmos o mal condicionamento. É importante salientar que as equações obtidas serão em função das novas variáveis r, mas que o mapeamento inverso é obtido por

$$x = x_{min} + (r - r_{min}) \left(\frac{x_{max} - x_{min}}{r_{max} - r_{min}} \right).$$

14.1 Problema de Mínimos Quadrados Não-Linear

Se o modelo a ser ajustado não puder ser escrito na forma $f(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})\mathbf{a}$, então não é possível solucionar o problema através da solução de um sistema de equações lineares. Neste caso, a condição de mínimo

$$\nabla_{\mathbf{a}} \Phi = \mathbf{0}$$

da origem a um sistema de equações não lineares

$$\begin{cases} f_1(a_0, a_1, ..., a_n) = 0 \\ \quad\quad\quad ... \\ \quad\quad\quad ... \\ f_n(a_0, a_1, ..., a_n) = 0 \end{cases}$$

que deve ser solucionado pelo método de Newton-Raphson multidimensional.

■ **Exemplo 14.2** Seja o modelo descrito pela equação $f(x) = a_0 + \cos(a_1 x)$ e os dados experimentais $f(1) = 0$, $f(2) = 2$, $f(3) = 0$ e $f(4) = 2$. Neste caso, a equação de mínimos quadrados se torna

$$\Phi = \sqrt{(a_0 + \cos(a_1 1) - 0)^2 + (a_0 + \cos(a_1 2) - 2)^2 + (a_0 + \cos(a_1 3) - 0)^2 + (a_0 + \cos(a_1 4) - 2)^2}$$

com condição de mínimo

$$\nabla_{\mathbf{a}} \Phi = \begin{Bmatrix} \frac{d\Phi}{da_0} \\ \frac{d\Phi}{da_1} \end{Bmatrix} = \mathbf{0} \implies \begin{cases} f_1(a_0, a_1) = 0 \\ f_2(a_0, a_1) = 0 \end{cases}$$

tal que

$$f_1(\mathbf{a}) = \frac{d\Phi}{da_0} = 2\left(\cos\left(4a1\right) + a_0 - 2\right) + 2\left(\cos\left(3a1\right) + a_0\right)$$
$$+2\left(\cos\left(2a_1\right) + a_0 - 2\right) + 2\left(\cos\left(a_1\right) + a_0\right) = 0$$

e

$$f_2(\mathbf{a}) = \frac{d\Phi}{da_1} = -8\left(\cos\left(4a_1\right) + a0 - 2\right)\sin\left(4a_1\right) - 6\left(\cos\left(3a_1\right) + a_0\right)\sin\left(3a_1\right)$$
$$-4\left(\cos\left(2a_1\right) + a_0 - 2\right)\sin\left(2a_1\right) - 2\left(\cos\left(a_1\right) + a_0\right)\sin\left(a_1\right) = 0$$

Assim, a matriz Jacobiana é descrita por

$$\begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} = \begin{bmatrix} \frac{df_1}{da_0} & \frac{df_1}{da_1} \\ \frac{df_2}{da_0} & \frac{df_2}{da_1} \end{bmatrix} = \begin{bmatrix} \frac{d^2\Phi}{da_0 da_0} & \frac{d^2\Phi}{da_0 da_1} \\ \frac{d^2\Phi}{da_1 da_0} & \frac{d^2\Phi}{da_1 da_1} \end{bmatrix}.$$

onde

$$J_{11} = 8,$$

$$J_{12}(\mathbf{a}) = J_{21}(\mathbf{a}) = -(8\sin(4a_1)) - 6\sin(3a_1) - 4\sin(2a_1) - 2\sin(a_1),$$

e

$$J_{22}(\mathbf{a}) = 2\sin(4a_1)^2 - 32\cos(4a_1)(\cos(4a_1) + a_0 - 2) + 18\sin(3a_1)^2$$
$$-18\cos(3a_1)(\cos(3a_1) + a_0) + 8\sin(2a_1)^2 - 8\cos(2a_1)(\cos(2a_1) + a_0 - 2)$$
$$+2\sin(a_1)^2 - 2\cos(a_1)(cos(a_1) + a_0)$$

Com isso, seguimos com o procedimento apresentado no capítulo de solução de sistemas de equações não lineares. ∎

15. Autovalores e Autovetores

Um problema de autovalores e autovetores tem a forma

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{15.1}$$

onde λ é um autovalor e $\mathbf{x}$ é um autovetor. As propriedades deste tipo de problema foram estudadas na primeira parte deste material (Fundamentos de Álgebra).

Para obtermos os autovalores, podemos rearranjar a Eq. (15.1) na forma

$$[\mathbf{A} - \lambda\mathbf{I}]\,\mathbf{x} = \mathbf{0} \tag{15.2}$$

e a solução não trivial é obtida quando

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0$$

dando origem a um polinômio (característico) com a forma

$$a_n\lambda^n + a_{n-1}\lambda^{n-1} \ldots + a_1$$

que terá n raízes (autovalores). De posse dos autovalores, podemos obter o autovalor $\mathbf{x}_i$ associado a um autovalor λ_i por meio da solução do sistema de equações homogêneas e Eq. (15.2).

O objetivo deste capítulo é apresentar diferentes métodos para a solução de problemas de autovalores e de autovetores.

15.1 Método da Potência

Seja uma matriz $\mathbf{A}_{[n\times n]}$ com um autovalor dominante, isto é, existe um autovalor que em módulo é maior do que os demais.

Começamos mostrando que os autovalores de $\mathbf{A}^k$ são iguais aos autovalores da matriz original elevados a k. Para isso, lembramos que a matriz pode ser escrita como

$$\mathbf{A} = \mathbf{\Phi}^{-1}\mathbf{\Lambda}\mathbf{\Phi}$$

onde $\mathbf{\Lambda}$ é uma matriz diagonal com os mesmos autovalores de $\mathbf{A}$. Com isso, dizemos que $\mathbf{A}$ e $\mathbf{\Lambda}$ são **similares**. Se elevarmos a matriz $\mathbf{A}$ ao expoente k

$$\mathbf{A}^k = \mathbf{\Phi}\mathbf{\Lambda}^k\mathbf{\Phi}^{-1} = \begin{bmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_n^k \end{bmatrix}.$$

Com esse resultado, podemos então definir uma sequência

$$\mathbf{x}_{k+1} = \mathbf{A}^k\mathbf{x}_0$$

onde $\mathbf{x}_0$ é um vetor não nulo. Como $\mathbf{x}_0$ pertence ao espaço da matriz $\mathbf{A}$, podemos utilizar a base dos autovetores de $\mathbf{A}$ tal que

$$\mathbf{x}_0 = \sum_{i=1}^{n} \alpha_i \boldsymbol{\phi}_i$$

tal que, inserindo a combinação linear na sequência

$$\mathbf{x}_{k+1} = \mathbf{A}^k \sum_{i=1}^{n} \alpha_i \boldsymbol{\phi}_i = \sum_{i=1}^{n} \alpha_i \mathbf{A}^k \boldsymbol{\phi}_i.$$

Agora, da definição do problema de autovalores e autovetores

$$\mathbf{A}\boldsymbol{\phi}_j = \lambda_j \boldsymbol{\phi}_j$$

onde $\boldsymbol{\phi}_j$ é o j-ésimo autovetor associado ao autovalor λ_j, podemos utilizar o resultado da potência k da matriz para obter

$$\mathbf{A}^k\boldsymbol{\phi}_j = \lambda_j^k \boldsymbol{\phi}_j.$$

Assim, podemos escrever a nossa sequência como

$$\mathbf{x}_{k+1} = \sum_{i=1}^{n} \alpha_i \mathbf{A}^k \boldsymbol{\phi}_i = \sum_{i=1}^{n} \alpha_i \lambda_i^k \boldsymbol{\phi}_i.$$

Estudando a sequência quando $k > K$ e considerando que temos um autovalor dominante λ_p, podemos verificar que o λ_p^k irá dominar todo o somatório, tal que

$$\mathbf{x}_{k+1} \to \lambda_p^k \boldsymbol{\phi}_p$$

ou seja, a sequência tende para o produto da potência k do autovalor dominante pelo autovetor associado. O autovalor dominante pode ser obtido de diferentes maneiras, pois ele é um fator multiplicativo dos elementos da sequência. Por exemplo, a norma Euclidiana de um elemento da sequência será (no limite)

$$\|\mathbf{x}_k\| \to \|\lambda_p^k \boldsymbol{\phi}_p\| = |\lambda_p^k| \|\boldsymbol{\phi}_p\|$$

tal que a razão de dois elementos da sequência

$$\frac{|\lambda_p^{k+1}| \|\boldsymbol{\phi}_p\|}{|\lambda_p^k| \|\boldsymbol{\phi}_p\|} = |\lambda_p|,$$

mas, neste caso, perdemos o sinal do autovalor. No entanto, essa não é a única relação que pode ser utilizada, pois a razão da componente 1 (por exemplo) de dois elementos da sequência também tenderá ao autovalor dominante.

■ **Exemplo 15.1** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

com autovalores $\lambda_1 = 1$ e $\lambda_2 = 2$ e autovetores $\boldsymbol{\phi}_1 = \begin{pmatrix} 1 & 0 \end{pmatrix}$ e $\boldsymbol{\phi}_2 = \begin{pmatrix} 0 & 1 \end{pmatrix}$. O autovalor dominante desta matriz é o λ_2. Assim, se assumirmos um vetor $\mathbf{x}_0 = \begin{pmatrix} 1 & 1 \end{pmatrix}$, podemos escrever a combinação linear na base dos autovetores como

$$\mathbf{x}_0 = \alpha_1 \boldsymbol{\phi}_1 + \alpha_2 \boldsymbol{\phi}_2 = 1 \begin{pmatrix} 1 & 0 \end{pmatrix} + 1 \begin{pmatrix} 0 & 1 \end{pmatrix}.$$

Com isso, podemos definir a sequência

$$\mathbf{x}_{k+1} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}^k \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 2^k \end{Bmatrix} == 1^k \begin{Bmatrix} 1 \\ 0 \end{Bmatrix} + 2^k \begin{Bmatrix} 0 \\ 1 \end{Bmatrix}$$

mostrando que as deduções que foram feitas estão corretas. Mais do que isso, podemos verificar que se k aumenta, $2^k >> 1^k$ e, no limite,

$$\mathbf{x}_{k+1} \to 2^k \begin{Bmatrix} 0 \\ 1 \end{Bmatrix} = \lambda_2^k \boldsymbol{\phi}_2.$$

Estudando a relação entre dois elementos da sequência

$$\frac{|2^{k+1}| \|\boldsymbol{\phi}_2\|}{|2^k| \|\boldsymbol{\phi}_2\|} = |2^{k+1-k}| = |2|.$$

■

■ **Exemplo 15.2** Considerando a matriz

$$\mathbf{A} = \begin{bmatrix} 10 & 1 & 2 & 3 \\ -1 & 20 & 2 & 1 \\ 2 & 3 & 30 & 1 \\ 1 & 2 & 3 & 40 \end{bmatrix}$$

e iniciado com $\mathbf{x}_0 = \mathbf{1}$, teremos

$$\mathbf{x}_1 = \begin{pmatrix} 16 & 22 & 36 & 46 \end{pmatrix}$$
$$\mathbf{x}_2 = \begin{pmatrix} 392 & 542 & 1224 & 2008 \end{pmatrix}$$
$$....$$
$$\mathbf{x}_{19} = \begin{pmatrix} 5.7101 & 2.9656 & 7.0122 & 52.5745 \end{pmatrix} \times 10^{29}$$
$$\mathbf{x}_{20} = \begin{pmatrix} 9.4118 & 4.8743 & 11.4576 & 86.7484 \end{pmatrix} \times 10^{32}$$

e, dividindo o último resultado pela norma Euclidiana do vetor, obtemos

$$\boldsymbol{\phi} = \begin{pmatrix} 0.1067813 & 0.0553020 & 0.1299917 & 0.9841959 \end{pmatrix}$$

que é muito próximo ao autovetor associado ao maior autovalor desta matriz. O autovalor pode ser obtido, por exemplo, com

$$\frac{x(1)_{20}}{x(1)_{19}} = 40.6171$$

que tem acurácia de 2 dígitos decimais. ■

■ **Exemplo 15.3** ■

É interessante notarmos também que a operação

$$\boldsymbol{\phi}^T \mathbf{A} \boldsymbol{\phi} = \lambda$$

o que é esperado, uma vez que o autovetor diagonaliza a matriz (neste caso, somente na posição correspondente ao autovalor dominante). Assim, da propriedade de que os autovalores de uma matriz são transladados por uma magnitude c por meio de uma operação (*shift*) na forma

$$\mathbf{A} - c\mathbf{I}$$

podemos realizar esta operação com a matriz original e com $c = \lambda$,obtendo

$$\mathbf{A}_1 = \begin{bmatrix} -30.6116 & 1 & 2 & 3 \\ -1 & -20.6116 & 2 & 1 \\ 2 & 3 & -10.6116 & 1 \\ 1 & 2 & 3 & -0.6116 \end{bmatrix}$$

e, realizando novamente o mesmo procedimento (até a vigésima iteração), obtemos

$$\mathbf{x}_{20} = \begin{pmatrix} 0.9854277 & 0.1224214 & -0.1143751 & -0.0293873 \end{pmatrix}$$

e

$$\lambda = -30.8078 + 40.6171 = 9.8082$$

que é o menor autovalor da matriz $\mathbf{A}$. Assim, podemos restringir o algoritmo de forma sucessiva ao espaço nulo (núcleo) dos autovetores já obtidos, de modo a obter os demais autovalores.

15.1.1 Método da Potência Inversa

Esta é uma variação do Método da Potência, onde trabalhamos com o inverso da matriz de coeficientes. Com isto, iremos obter o autovetor associado ao **menor** autovalor pois, utilizando novamente a propriedade de matrizes similares

$$\mathbf{A}^{-1} = \boldsymbol{\Phi} \boldsymbol{\Lambda}^{-1} \boldsymbol{\Phi}^{-1} = \begin{bmatrix} \lambda_1^{-1} & & \\ & \ddots & \\ & & \lambda_n^{-1} \end{bmatrix}$$

tal que o menor autovalor inverso será o maior e irá dominar a sequência.

Assim

$$\mathbf{x}_k = \mathbf{A}^{-k} \mathbf{x}_0$$

e, com uma dedução muito parecida com a realizada no método da potência, verificamos que

$$\frac{\|\mathbf{x}_{k+1}\|}{\|\mathbf{x}_k\|} \rightarrow \frac{1}{\lambda}.$$

Deve-se salientar que não invertemos a matriz, mas solucionamos um sistema de equações na forma

$$\mathbf{A}\mathbf{x}_{k+1} = \mathbf{x}_k.$$

■ **Exemplo 15.4** Novamente iremos considerar a matriz

$$\mathbf{A} = \begin{bmatrix} 10 & 1 & 2 & 3 \\ -1 & 20 & 2 & 1 \\ 2 & 3 & 30 & 1 \\ 1 & 2 & 3 & 40 \end{bmatrix}$$

Assim, iniciando com $\mathbf{x}_0 = \mathbf{1}$ teremos

$$\begin{aligned}
&\mathbf{A}\mathbf{x}_1 = \mathbf{1} \rightarrow \mathbf{x}_1 = \begin{pmatrix} 0.0849 & 0.0511 & 0.0212 & 0.0187 \end{pmatrix} \\
&\mathbf{A}\mathbf{x}_2 = \mathbf{x}_1 \rightarrow \mathbf{x}_2 = \begin{pmatrix} 0.008179 & 0.0029702 & -0.0001151 & 0.0001226 \end{pmatrix} \\
&\ldots\ldots \\
&\mathbf{A}\mathbf{x}_{19} = \mathbf{x}_{18} \rightarrow \mathbf{x}_{19} = \begin{pmatrix} 1.11866 & 0.1384 & -0.1296799 & -0.0333282 \end{pmatrix} \times 10^{-19} \\
&\mathbf{A}\mathbf{x}_{20} = \mathbf{x}_{19} \rightarrow \mathbf{x}_{20} = \begin{pmatrix} 1.412 & 1.1411 & -0.132292 & -0.0339997 \end{pmatrix} \times 10^{-20}
\end{aligned}$$

e, normalizando o ultimo vetor

$$\boldsymbol{\phi} = \begin{pmatrix} 0.9855 & 0.1219 & -0.1142436 & -0.0293611 \end{pmatrix}.$$

A menor autovalor é obtido por

$$\lambda_{min} = \left(\frac{1.412 \times 10^{-20}}{1.11866 \times 10^{-19}} \right)^{-1} = 9.8024.$$

■

15.2 Método de Jacobi

Sabendo que uma matriz $\mathbf{A}$ é simétrica e real, então existe uma matriz de mudança de base $\mathbf{R}$, real, que diagonaliza a matriz, na forma

$$\boldsymbol{\Lambda} = \mathbf{R}^T \mathbf{A} \mathbf{R},$$

onde $\boldsymbol{\Lambda}$ é diagonal e contém os autovalores de $\mathbf{A}$ e as colunas $\mathbf{R}$ contém os autovetores. A mudança de base pode ser construída por uma série de operações de rotação sucessivas em torno de diferentes eixos, com o objetivo de zerar blocos de dimensão 2×2 fora da diagonal. Desta forma, a cada iteração do método temos uma matriz de rotação na forma

$$\mathbf{R}_k = \begin{bmatrix} 1 & 0 & & & & & \\ 0 & 1 & & & & & \\ & & \ddots & & & & \\ & & & \cos(\theta) & & -\sin(\theta) & \\ & & & & \ddots & & \\ & & & \sin(\theta) & & \cos(\theta) & \\ & & & & & & \ddots \\ & & & & & & & 1 \end{bmatrix}.$$

Se realizarmos a operação

$$A_{k+1} = \mathbf{R}_k^T \mathbf{A}_k \mathbf{R}_k,$$

então as posições i e j da matriz $\mathbf{A}_{k+1}$ serão

$$a_{ii} = a_{ii}\cos(\theta)^2 + 2a_{ij}\sin(\theta)\cos(\theta) + a_{ij}\sin(\theta)^2,$$

e

$$a_{ij} = a_{ji} = (a_{jj} - a_{ii})\sin(\theta)\cos(\theta) + a_{ij}(\cos(\theta)^2 - \sin(\theta)^2),$$

sendo que os termos fora da diagonal são iguais a zero quando

$$\theta = \frac{1}{2}atan\left(\frac{2a_{ij}}{a_{ii} - a_{jj}}\right).$$

Uma escolha comum é selecionar o termo de maior magnitude fora da diagonal a cada iteração para calcularmos o θ.

Assim, podemos ir zerando os termos fora da diagonal sucessivamente, tal que ao final do processo, termos uma matriz diagonal contendo os autovalores e uma matriz de autovetores, tal que

$$\mathbf{\Phi} = \mathbf{R}_K\mathbf{R}_{K-1}...\mathbf{R}_1$$

onde K é o número de iterações.

■ **Exemplo 15.5** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 4 & 2 & 1 \\ 2 & 6 & 1 \\ 1 & 1 & 15 \end{bmatrix}$$

verificamos que o maior número fora da diagonal é 2, na posição $i = 1,\ j = 2$. Assim, a matriz de rotação $\mathbf{R}_1$ será

$$\mathbf{R}_1 = \begin{bmatrix} \cos(\theta_1) & -sin(\theta_1) & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

com

$$\theta_1 = \frac{1}{2}atan\left(\frac{2\cdot 2}{4-6}\right)$$

tal que $\theta_1 = -0.5535744$.

Assim, temos que, ao final da primeira iteração ($k = 1$),

$$\mathbf{A}_2 = \mathbf{R}_1^T\mathbf{A}\mathbf{R}_1 = \begin{bmatrix} 2.763932 & 0 & 0.3249197 \\ 0 & 7.236068 & 1.3763819 \\ 0.3249197 & 1.3763819 & 15 \end{bmatrix}.$$

Agora, o maior termo fora da diagonal é 1.3763819, com $i = 2$ e $j = 3$. Assim, a matriz de rotação será

$$\mathbf{R}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_2) & -\sin(\theta_2) \\ 0 & \sin(\theta_2) & \cos(\theta_2) \end{bmatrix}$$

com

$$\theta_2 = \frac{1}{2} atan\left(\frac{2 \cdot 1.3763819}{7.236068 - 15}\right)$$

tal que $\theta_2 = -0.1703648$ e

$$\mathbf{A}_3 = \mathbf{R}_2^T \mathbf{A}_2 \mathbf{R}_2 = \begin{bmatrix} 2.763932 & -0.0550875 & 0.3202158 \\ -0.0550875 & 6.9992857 & 0 \\ 0.3202158 & 0 & 15.236782 \end{bmatrix}.$$

Agora, identificamos que o termo de maior magnitude é 0.3202158 na posição $i = 1$, $j = 3$. A matriz de rotação será

$$\mathbf{R}_3 = \begin{bmatrix} \cos(\theta_3) & 0 & -sin(\theta_3) \\ 0 & 1 & 0 \\ \sin(\theta_3) & 0 & \cos(\theta_3) \end{bmatrix}$$

com $\theta_3 = -0.0256505$, tal que

$$\mathbf{A}_4 = \mathbf{R}_3^T \mathbf{A}_3 \mathbf{R}_3 = \begin{bmatrix} 2.7557165 & -0.0550694 & 0 \\ -0.0550694 & 6.9992857 & -0.0014129 \\ 0 & -0.0014129 & 15.244998 \end{bmatrix}.$$

Finalmente, como o maior termo (em magnitude) agora é -0.0550694 na posição $i = 1$, $j = 2$, temos que

$$\mathbf{R}_4 = \begin{bmatrix} \cos(\theta_4) & -sin(\theta_4) & 0 \\ \sin(\theta_4) & \cos(\theta_4) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

e

$$\mathbf{A}_5 = \mathbf{R}_4^T \mathbf{A}_4 \mathbf{R}_4 = \begin{bmatrix} 2.755002 & 0 & -0.0000183 \\ 0 & 7.0000002 & -0.0014127 \\ -0.0000183 & -0.0014127 & 15.244998 \end{bmatrix}.$$

Neste momento, verificamos que os termos fora da diagonal já se encontram em uma ordem de grandeza de aproximadamente 1×10^{-3} dos demais valores da diagonal. De fato, os autovalores da matriz $\mathbf{A}$ são $\lambda_1 = 2.755002$, $\lambda_2 = 7$ e $\lambda_3 = 15.244998$. Os autovetores são obtidos pelo produto

$$\mathbf{\Phi} = \mathbf{R}_1 \mathbf{R}_2 \mathbf{R}_3 \mathbf{R}_4 = \begin{bmatrix} 0.8547356 & 0.5070736 & 0.1109213 \\ -0.5183361 & 0.8451319 & 0.1306897 \\ -0.0274739 & -0.1691997 & 0.9851988 \end{bmatrix},$$

que também mostram uma grande acurácia. Deve-se comentar que este método utiliza um grande número de multiplicações de matrizes e o cálculo do arco tangente, que é propenso a erros de arredondamento.

■

O algoritmo do método de Jacobi é apresentado no Alg. 37 e o algoritmo utilizado para a determinação da posição do maior valor em módulo da triangular superior de uma matriz é apresentado no Alg. 36.

Algoritmo 36: Retorna a posição do termo com maior valor em módulo em uma matriz simétrica, sem considerar a diagonal

```
1  PosMaiorValor(A,n)
       Entrada: A matriz simétrica n × n
2      Inicializa as posições
3      pos_i ← 0
4      pos_j ← 0
5      Maior valor até o momento
6      maior ← 0
7      Laços pela triangular superior
8      for i ∈ {1,...,n−1} do
9          for j ∈ {i+1,...,n} do
10             Valor absoluto na posição
11             v ← |A(i, j)|
12             Verifica se é o maior valor
13             if v > maior then
14                 pos_i ← i
15                 pos_j ← j
16                 maior ← v
17     Retorna linha e coluna
18     return pos_i, pos_j
```

15.3 Solucionando problemas de autovalores e autovetores utilizando a decomposição QR

Lembrando, dizemos que duas matrizes quadradas $\mathbf{A}$ e $\mathbf{B}$ são **similares** se contém os mesmos autovalores e autovetores. Se afirmativo, ambas satisfazem a relação

$$\mathbf{A} = \mathbf{C}^{-1}\mathbf{B}\mathbf{C},$$

onde $\mathbf{C}$ é inversível.

Sejam as matrizes $\mathbf{Q}_k$ e $\mathbf{R}_k$ obtidas pela decomposição QR de uma matriz $\mathbf{A}_k$

$$\mathbf{A}_k = \mathbf{Q}_k\mathbf{R}_k$$

Vamos gerar uma matriz

$$\mathbf{A}_{k+1} = \mathbf{R}_k\mathbf{Q}_k$$

onde deve ser observado que a ordem das matrizes $\mathbf{R}_k$ e $\mathbf{Q}_k$ estão trocadas (importante!). O interessante é que se realizarmos a operação

$$\mathbf{Q}_k^{-1}\mathbf{A}_k\mathbf{Q}_k$$

sobre a matriz $\mathbf{A}_k$ obtemos

$$\mathbf{Q}_k^{-1}\left(\mathbf{Q}_k\mathbf{R}_k\right)\mathbf{Q}_k = \mathbf{I}\mathbf{R}_k\mathbf{Q}_k = \mathbf{A}_{k+1}$$

o que mostra que $\mathbf{A}_k$ e $\mathbf{A}_{k+1}$ são similares. Isso leva a um procedimento iterativo em que

$$\mathbf{A}_{k+1} = \mathbf{Q}_k^{-1}\mathbf{A}_k\mathbf{Q}_k \tag{15.3}$$

Algoritmo 37: Método de Jacobi para calcular os autovalores e autovetores de uma matriz simétrica.

```
1  Jacobi(A,n,niter)
       Entrada: A matriz simétrica n × n
       Entrada: niter número de iterações
2      Inicializa a matriz dos autovetores
3      X ← I_{n×n}
4      Iterações do método
5      for i ∈ {1,...,niter} do
6          Posições com maior valor
7          mi, mj ← PosMaiorValor(A, n)
8          Determina o ângulo
9          θ ← 0.5 atan(2A(mi,mj)/(A(mi,mi) − A(mj,mj)))
10         Monta a matriz de rotação
11         R ← I_{n×n}
12         R(mi,mi) ← cos(teta)
13         R(mi,mj) ← −sin(teta)
14         R(mj,mi) ← sin(teta)
15         R(mj,mj) ← cos(teta)
16         Rotaciona a matriz
17         A ← R^T AR
18         Acumula a matriz dos autovetores
19         X ← XR
20     Retorna a matriz A e a matriz com os autovetores X
21     return A, X
```

sendo que após um número suficiente de ortogonalizações iremos obter uma matriz **A** triangular superior cujos elementos da diagonal serão os autovalores. Caso a matriz **A** seja simétrica, então a matriz **A** será diagonal, com os autovalores na diagonal.

Conforme vimos no método de Jacobi, a Eq. (15.3) pode ser interpretada como uma mudança de base (não necessariamente provocada por rotações neste caso), tal que podemos obter a matriz de autovetores com uma abordagem similar ao método de Jacobi

$$\mathbf{X}_{k+1} = \mathbf{X}_k \mathbf{Q}_k.$$

Caso **A** não seja simétrica, então devemos obter os autovetores de interesse por meio da solução do sistema de equações da Eq. (15.2), para cada λ de interesse. Por esse motivo dizemos que o método QR é muito prático para obtermos os autovalores para uma matriz geral, mas nem tão prático para a obtenção dos autovetores. No caso simétrico ele é um método bem competitivo.

O procedimento para o caso de uma matriz simétrica está ilustrado no Alg. 38.

Algoritmo 38: Utilizando o método QR para calcular os autovalores e autovetores de uma matriz.

```
1  AutovalQR(A,n,niter)
       Entrada: A n × n
       Entrada: niter número de iterações
2      Inicializa a matriz dos autovetores
3      X ← I_{n×n}
4      for i ∈ {1...niter} do
5          Decomposição QR da matriz
6          Q,R ← QR(A,n)
7          Monta estimativa de A_k
8          A ← RQ
9          Acumula os autovetores
10         X ← XQ
11     Retorna a matriz A e a matriz X
12     return A, X
```

■ **Exemplo 15.6** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 4 & 2 & 1 \\ 2 & 6 & 1 \\ 1 & 1 & 15 \end{bmatrix}$$

Na primeira iteração, obtemos a decomposição QR da matriz $\mathbf{A} = \mathbf{QR}$ tal que

$$\mathbf{A}_1 = \mathbf{RQ} = \begin{bmatrix} 7.0 & 2.04939 & -3.1305 \\ 2.04939 & 4.0 & 0.0 \\ -3.1305 & 0.0 & 14.0 \end{bmatrix}$$

e

$$\mathbf{X}_1 = \mathbf{Q}_1 = \begin{bmatrix} -0.87287 & 0.44721 & -0.19518 \\ -0.43643 & -0.894427 & -0.09759 \\ -0.218218 & 0 & 0.9759 \end{bmatrix}.$$

Na segunda iteração realizamos uma nova decomposição QR, $\mathbf{A}_1 = \mathbf{Q}_1\mathbf{R}_1$, tal que

$$\mathbf{A}_2 = \mathbf{R}_1\mathbf{Q}_1 = \begin{bmatrix} 11.0 & -1.03775 & 4.19401 \\ -1.03775 & 4.84615 & -3.41969 \\ 4.19401 & -3.41969 & 9.15385 \end{bmatrix}$$

com

$$\mathbf{X}_2 = \mathbf{X}_1\mathbf{Q}_2 = \begin{bmatrix} 0.57735 & -0.47075 & -0.66712 \\ 0.57735 & 0.813125 & -0.074125 \\ 0.57735 & -0.342369 & 0.74125 \end{bmatrix}.$$

Continuando, verificamos que na iteração 20

$$\mathbf{A}_{20} = \mathbf{R}_{20}\mathbf{Q}_{20} = \begin{bmatrix} 15.244998 & -6\times10^{-6} & 4\times10^{-13} \\ -6\times10^{-6} & 7 & -2\times10^{-7} \\ 4\times10^{-13} & -2\times10^{-7} & 2.755 \end{bmatrix}$$

com os autovalores na diagonal e

$$\mathbf{X}_{20} = \begin{bmatrix} 0.1108336 & 0.5070924 & -0.8547357 \\ 0.1305463 & 0.8451542 & 0.5183359 \\ 0.985227 & -0.1690316 & 0.02747243 \end{bmatrix}$$

com os autovetores correspondentes aos autovalores de cada coluna em $\mathbf{A}$. Pode-se avaliar que a operação

$$\mathbf{X}_{20}^{-1}\mathbf{A}\mathbf{X}_{20} = \mathbf{\Lambda} \approx \mathbf{A}_{20}$$

como esperado, já que $\mathbf{X}$ contém os autovetores.

■

■ **Exemplo 15.7** Seja a matriz não simétrica

$$\mathbf{A} = \begin{bmatrix} 4 & 2 & 1 \\ 1 & 6 & 1 \\ 2 & 1 & 15 \end{bmatrix}.$$

A utilização do procedimento do exemplo anterior resulta em

$$\mathbf{A}_{20} = \begin{bmatrix} 15.3371 & 0.601199 & -0.647008 \\ -1\times10^{-7} & 6.42632 & 1.1045 \\ 2\times10^{-12} & -5\times10^{-6} & 3.23657 \end{bmatrix}.$$

e

$$\mathbf{X}_{20} = \begin{bmatrix} 0.107768 & 0.571758 & -0.813313 \\ 0.117274 & 0.805055 & 0.581492 \\ 0.987235 & -0.158047 & 0.019707 \end{bmatrix}.$$

Embora a matriz $\mathbf{A}$ seja triangular superior, ainda contém os autovalores na sua diagonal principal. No entanto,

$$\mathbf{X}_{20}^{-1}\mathbf{A}\mathbf{X}_{20} = \begin{bmatrix} 15.2434 & 0.103792 & 0.0605429 \\ 0.103792 & 6.97698 & 0.321207 \\ 0.0605429 & 0.321207 & 2.77966 \end{bmatrix} \neq \mathbf{\Lambda}$$

que não é diagonal, tal que $\mathbf{X}_{20}$ não contém os autovetores.

■

15.3.1 Calculando autovetores quando a matriz A não é simétrica

Como vimos no exemplo anterior, o procedimento de obtenção dos autovalores por QR funciona muito bem para autovalores, mas não retorna os autovetores. Neste caso, poderíamos obter os autovetor correspondente a um autovalor λ_i solucionando o sistema linear

$$\mathbf{A} - \lambda_1 \mathbf{I} = \mathbf{0}$$

mas esse sistema de equações irá retornar $\mathbf{0}$. Uma alternativa é aumentar o sistema pela consideração de uma restrição de que a soma de todos os elementos do autovetor seja 1 (outras alternativas podem ser utilizadas), gerando o sistema

$$\begin{bmatrix} \mathbf{A} - \lambda_i \mathbf{I} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{Bmatrix} \boldsymbol{\phi}_i \\ z \end{Bmatrix} = \begin{Bmatrix} \mathbf{0} \\ 1 \end{Bmatrix}$$

onde $\mathbf{1}$ é um vetor $n \times 1$ com valores unitários e z é uma variável auxiliar (que não será utilizada).

■ **Exemplo 15.8** Vamos considerar o exemplo anterior,

$$\mathbf{A} = \begin{bmatrix} 4 & 2 & 1 \\ 1 & 6 & 1 \\ 2 & 1 & 15 \end{bmatrix}$$

com autovalores (da diagonal principal de $\mathbf{A}_{20}$): $\lambda_1 = 15.3371$, $\lambda_2 = 6.42632$ e $\lambda_3 = 3.23657$.

Para λ_1

$$\left[\begin{bmatrix} 4 & 2 & 1 \\ 1 & 6 & 1 \\ 2 & 1 & 15 \end{bmatrix} - 15.3371 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} \right] \begin{Bmatrix} \boldsymbol{\phi}_1 \\ z \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{Bmatrix}$$

$$\begin{Bmatrix} 1 & 1 & 1 \end{Bmatrix} \qquad \begin{matrix} \\ \\ 0 \end{matrix}$$

ou

$$\begin{bmatrix} -11.3371 & 2 & 1 & 1 \\ 1 & -9.3371 & 1 & 1 \\ 2 & 1 & -0.3371 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{Bmatrix} \phi_1(1) \\ \phi_1(2) \\ \phi_1(3) \\ z \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{Bmatrix}$$

com $\boldsymbol{\phi}_1 = \begin{pmatrix} 0.08889681248475953 & 0.09673804282450776 & 0.8143651446907328 \end{pmatrix}$.

Para λ_2

$$\left[\begin{bmatrix} 4 & 2 & 1 \\ 1 & 6 & 1 \\ 2 & 1 & 15 \end{bmatrix} - 6.42632 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} \right] \begin{Bmatrix} \boldsymbol{\phi}_2 \\ z \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{Bmatrix}$$

$$\begin{Bmatrix} 1 & 1 & 1 \end{Bmatrix} \qquad \begin{matrix} \\ \\ 0 \end{matrix}$$

com $\boldsymbol{\phi}_2 = \begin{pmatrix} 0.4964832146523916 & 0.7011071779599485 & -0.1975903926123397 \end{pmatrix}$.

Finalmente, para λ_2

$$\left[\begin{bmatrix} 4 & 2 & 1 \\ 1 & 6 & 1 \\ 2 & 1 & 15 \end{bmatrix} - 3.23657 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} \right] \begin{Bmatrix} \boldsymbol{\phi}_3 \\ z \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{Bmatrix}$$

$$\begin{Bmatrix} 1 & 1 & 1 \end{Bmatrix} \qquad \begin{matrix} \\ \\ 0 \end{matrix}$$

com $\boldsymbol{\phi}_3 = \begin{pmatrix} 1.8300066615054524 & -0.5670784170288626 & -0.2629282444765897 \end{pmatrix}$.

Assim, a matriz de autovetores será

$$\mathbf{X} = \begin{bmatrix} \boldsymbol{\phi}_1 & \boldsymbol{\phi}_2 & \boldsymbol{\phi}_3 \end{bmatrix}$$

tal que

$$\mathbf{X}^{-1}\mathbf{A}\mathbf{X} = \boldsymbol{\Lambda}$$

como esperado.

■

15.4 Problema generalizado de autovalores e autovetores

É possível utilizar a decomposição Cholesky para a solução de um problema generalizado de autovalores e autovetores na forma $\mathbf{AX} = \lambda\mathbf{BX}$, onde a matriz $\mathbf{B}$ é simétrica, positivo-definida e real. Neste caso, já vimos que a decomposição Cholesky de $\mathbf{B}$ é $\mathbf{U}^T\mathbf{U}$, tal que

$$\mathbf{AX} = \lambda\mathbf{U}^T\mathbf{U}\mathbf{X}.$$

Definido uma nova variável $\mathbf{Y}$, tal que

$$\mathbf{Y} = \mathbf{U}\mathbf{X}$$

e

$$\mathbf{X} = \mathbf{U}^{-1}\mathbf{Y},$$

podemos escrever

$$\mathbf{A}\mathbf{U}^{-1}\mathbf{Y} = \lambda\mathbf{U}^T\mathbf{Y},$$

e, se multiplicarmos esta equação por $\mathbf{U}^{-T}$, obtemos

$$\mathbf{U}^{-T}\mathbf{A}\mathbf{U}^{-1}\mathbf{Y} = \lambda\mathbf{U}^{-T}\mathbf{U}^T\mathbf{Y},$$

onde fica claro que $\mathbf{U}^{-T}\mathbf{U}^T = \mathbf{I}$, tal que

$$\mathbf{U}^{-T}\mathbf{A}\mathbf{U}^{-1}\mathbf{Y} = \lambda\mathbf{Y}.$$

Este problema de autovalores e autovetores pode ser solucionado pelos demais métodos vistos neste capítulo, resultando nos mesmos autovalores do problema original. Ainda, de posse dos vetores $\mathbf{Y}$, é possível obter os autovetores do problema generalizado, pois $\mathbf{X} = \mathbf{U}^{-1}\mathbf{Y}$.

Como já vimos, a decomposição Cholesky é bem eficiente. Da mesma forma, é possível deduzir uma expressão para obtermos $\mathbf{U}^{-1}$ que compartilha da eficiência do método Cholesky. Para entendermos a relação, vamos considerar um problema de dimensão 3

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} \begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ 0 & \alpha_{22} & \alpha_{23} \\ 0 & 0 & \alpha_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

onde $\boldsymbol{\alpha}$ contém a inversa de $\mathbf{U}$. Solucionando para incógnitas α_{ij}, obtemos

$$\alpha_{11} = \frac{1}{u_{11}}, \quad \alpha_{22} = \frac{1}{u_{22}}, \quad \alpha_{33} = \frac{1}{u_{33}}$$

e

$$\alpha_{12} = -\frac{u_{12}\alpha_{22}}{u_{22}}, \quad \alpha_{13} = -\frac{u_{12}\alpha_{23} + u_{13}\alpha_{33}}{u_{11}}, \quad \alpha_{23} = -\frac{u_{23}\alpha_{33}}{u_{22}}.$$

Esse procedimento pode ser generalizado para

$$\alpha_{ii} = 1/u_{ii}$$

$$\alpha_{ij} = -\frac{1}{u_{ii}}\left(\sum_{k=i+1}^{j} u_{ik}\alpha_{kj}\right) \quad i < j$$

$$\alpha_{ij} = 0 \quad i > j$$

com i e j variando de n até 1. O Algoritmo 39 apresenta as etapas do cálculo.

Algoritmo 39: Inversa de uma matriz triangular superior **U**

```
1  InverteU(U,n)
       Entrada: U matriz triangular superior n × n
2      Inicializa a matriz inversa
3      S ← 0_{n×n}
4      Iteração pelas linhas da matriz (back)
5      for i ∈ {n, n−1, ...1} do
6          Termo diagonal
7          S(i,i) ← 1/U(i,i)
8          for j ∈ {n, n−1, ..., 1} do
9              if i < j then
10                 Inicializa o somatório
11                 soma ← 0
12                 for k ∈ {i+1, .., j} do
13                     soma ← soma + U(i,k) · U(k,j)
14                 Armazena o valor
15                 S(i,j) ← soma/U(i,i)
16     Retorna a inversa
17     return S
```

■ **Exemplo 15.9** Seja o problema generalizado de autovalores e autovetores definidos por

$$\mathbf{A} = \begin{bmatrix} 6 & 1 & 1 \\ 1 & 9 & 1 \\ 1 & 4 & 1 \end{bmatrix}$$

e

$$\mathbf{B} = \begin{bmatrix} 4 & 2 & 1 \\ 2 & 6 & 1 \\ 1 & 1 & 15 \end{bmatrix}.$$

A decomposição Cholesky de **B** é

$$\mathbf{U} = \begin{bmatrix} 2 & 1 & 0.5 \\ 0 & 2.236068 & 0.2236068 \\ 0 & 0 & 3.8340579 \end{bmatrix}$$

com inversa

$$\mathbf{U}^{-1} = \begin{bmatrix} 0.5 & -0.2236068 & -0.0521641 \\ 0 & 0.4472136 & -0.0260820 \\ 0 & 0 & 0.2608203 \end{bmatrix}.$$

Definindo um operador $\mathbf{D}$

$$\mathbf{D} = \mathbf{U}^{-T}\mathbf{A}\mathbf{U}^{-1} = \begin{bmatrix} 1.5 & -0.4472136 & -0.0391230 \\ -0.4472136 & 1.9 & 0.0058321 \\ -0.0391230 & 0.3557592 & 0.0319728 \end{bmatrix}$$

temos que a solução do problema de autovalores e autovetores definido por

$$\mathbf{D}\mathbf{Y} = \lambda\mathbf{Y}$$

resulta em $\lambda_1 = 2.1937615$, $\lambda_2 = 1.2060725$, $\lambda_3 = 0.0321388$ com autovetores

$$\mathbf{Y}_1 = \begin{Bmatrix} 0.5418033 \\ -0.8277240 \\ -0.1460214 \end{Bmatrix},$$

$$\mathbf{Y}_2 = \begin{Bmatrix} -0.8334240 \\ -0.5359830 \\ -0.1346349 \end{Bmatrix}$$

e

$$\mathbf{Y}_3 = \begin{Bmatrix} 0.0277133 \\ 0.0035141 \\ 0.9996097 \end{Bmatrix}.$$

Assim, utilizando o mapeamento $\mathbf{X} = \mathbf{U}^{-1}\mathbf{Y}$, obtemos

$$\mathbf{X}_1 = \begin{Bmatrix} 0.4636034 \\ -0.3663609 \\ -0.0380853 \end{Bmatrix},$$

$$\mathbf{X}_2 = \begin{Bmatrix} -0.2898395 \\ -0.2361873 \\ -0.0351155 \end{Bmatrix}$$

e

$$\mathbf{X}_3 = \begin{Bmatrix} -0.0390728 \\ -0.0245003 \\ 0.2607185 \end{Bmatrix},$$

que são os autovetores do problema original.

■

15.5 OPCIONAL - Método de Leverrier-Faddev

O de Leverrier-Faddev é um método direto, onde obtemos os coeficientes do polinômio característico

$$P(\lambda) = (-1)^n \left[\lambda^n - P_1\lambda^{n-1} .... - P_n\right]$$

onde

$$P_1 = tr(\mathbf{A})$$

$$P_k = \frac{1}{k} tr(\mathbf{A}_k), \; k = 2..n$$

com

$$\begin{aligned} \mathbf{A}_k &= \mathbf{B}_k\mathbf{A} \\ \mathbf{B}_k &= \mathbf{A}_{k-1} - P_{k-1}\mathbf{I} \end{aligned}$$

■ **Exemplo 15.10** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 10 & 2 \\ 3 & 7 \end{bmatrix}$$

Iniciamos com

$$P_1 = tr(\mathbf{A}) = 10 + 7 = 17$$

e, para o segundo coeficiente, temos que $\mathbf{A}_1 = \mathbf{A}$ e $P_1 = 17$, tal que

$$\mathbf{B}_1 = \mathbf{A} - 17\mathbf{I} = \begin{bmatrix} -7 & 2 \\ 3 & -10 \end{bmatrix}$$

$$\mathbf{A}_2 = \begin{bmatrix} -7 & 2 \\ 3 & -10 \end{bmatrix} \begin{bmatrix} 10 & 2 \\ 3 & 7 \end{bmatrix} = \begin{bmatrix} -64 & 0 \\ 0 & -64 \end{bmatrix}$$

com

$$P_2 = \frac{1}{2} tr(\mathbf{A}_2) = -64.$$

Portanto, o polinômio característico será

$$P(\lambda) = (-1)^2 \left[\lambda^2 - 17\lambda + 64\right]$$

com raizes

$$\lambda_1 = 11.3723 \text{ e } \lambda_2 = 5.6277.$$

■

É interessante notar que uma vez de posse dos autovalores e das matrizes $\mathbf{B}_k$ obtidas ao longo do processo, podemos obter facilmente os autovetores associados a cada um dos autovalores. Para isto fazemos,

$$\left(\mathbf{X}^k\right)^T = \lambda \left(\mathbf{X}^{k-1}\right)^T + \left(\mathbf{B}^{k-1}\right)^T$$

sendo que $\mathbf{X}^0 = \mathbf{I}$.

■ **Exemplo 15.11** Considerando o exemplo anterior,

$$\left(\mathbf{X}^1\right)^T = \begin{bmatrix} 11.3723 & 0 \\ 0 & 5.6277 \end{bmatrix}\mathbf{I} + \begin{bmatrix} -7 & 2 \\ 3 & -10 \end{bmatrix}^T = \begin{bmatrix} 4.3723 & 3 \\ 2 & -4.3722 \end{bmatrix}^T$$

onde cada linha conterá um dos autovetores.

■

■ **Exemplo 15.12** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 10 & 1 & 2 & 3 \\ -1 & 20 & 2 & 1 \\ 2 & 3 & 30 & 1 \\ 1 & 2 & 3 & 40 \end{bmatrix}$$

Os autovalores são obtidos com:

$$P_1 = tr(\mathbf{A}) = 100$$

$$\mathbf{B}_2 = \begin{bmatrix} 10 & 1 & 2 & 3 \\ -1 & 20 & 2 & 1 \\ 2 & 3 & 30 & 1 \\ 1 & 2 & 3 & 40 \end{bmatrix} - 100\mathbf{I} = \begin{bmatrix} -90 & 1 & 2 & 3 \\ -1 & -80 & 2 & 1 \\ 2 & 3 & -70 & 1 \\ 1 & 2 & 3 & -60 \end{bmatrix}$$

$$\mathbf{A}_2 = \mathbf{B_2}\mathbf{A}$$

$$P_2 = \frac{1}{2}tr(\mathbf{A}_2) = -3483$$

$$\mathbf{B}_3 = \mathbf{A_2} + 3483\mathbf{I} = \begin{bmatrix} 2589 & -58 & -109 & -147 \\ 75 & 1890 & -99 & -41 \\ -122 & -146 & 1396 & -21 \\ -46 & -70 & -84 & 1091 \end{bmatrix}$$

$$\mathbf{A}_3 = \mathbf{B}_3\mathbf{A}$$

$$P_3 = \frac{1}{3}tr(\mathbf{A}_3) = 49236$$

$$\mathbf{B}_4 = \mathbf{A}_3 - 49236\mathbf{I} = \begin{bmatrix} 23653 & 808 & 1351 & 1720 \\ -1379 & -11740 & 837 & 376 \\ 1697 & 1104 & -7955 & 44 \\ 533 & 484 & 521 & -5888 \end{bmatrix}$$

$$\mathbf{A}_4 = \mathbf{B}_4\mathbf{A}$$

$$P_4 = \frac{1}{4}tr(\mathbf{A}_4) = -232916$$

dando origem ao polinômio

$$P(\lambda) = (-1)^4\left[\lambda^4 - 100\lambda^3 + 3483\lambda^2 - 49236\lambda + 232916\right]$$

com raízes

$$\lambda_1 = 40.612, \lambda_2 = 30.2235, \lambda_3 = 19.351, \text{ e } \lambda_4 = 9.803.$$

Os autovetores são obtidos com

$$\left(\mathbf{x}^1\right)^T = \begin{bmatrix} 40.612 & 0 & 0 & 0 \\ 0 & 30.2235 & 0 & 0 \\ 0 & 0 & 19.351 & 0 \\ 0 & 0 & 0 & 9.803 \end{bmatrix} I + \begin{bmatrix} -90 & 1 & 2 & 3 \\ -1 & -80 & 2 & 1 \\ 2 & 3 & -70 & 1 \\ 1 & 2 & 3 & -60 \end{bmatrix}^T = \begin{bmatrix} -49.388 & -1 & 2 & 1 \\ 1 & -49.765 & 3 & 2 \\ 2 & 2 & -50.648 & 3 \\ 3 & 1 & 1 & -50.197 \end{bmatrix}$$

$$\left(\mathbf{X}^2\right)^T = \begin{bmatrix} 40.612 & 0 & 0 & 0 \\ 0 & 30.2235 & 0 & 0 \\ 0 & 0 & 19.351 & 0 \\ 0 & 0 & 0 & 9.803 \end{bmatrix} \left(\mathbf{X}^1\right)^T + \begin{bmatrix} 2589 & -58 & -109 & -147 \\ 75 & 1890 & -99 & -41 \\ -122 & -146 & 1396 & -21 \\ -46 & -70 & -84 & 1091 \end{bmatrix}^T = \begin{bmatrix} 583.258 & 34.388 & -40.777 & -5.388 \\ -27.765 & 385.358 & -55.295 & -9.530 \\ -70.297 & -60.297 & 415.889 & -25.946 \\ -117.592 & -31.197 & -11.197 & 598.939 \end{bmatrix}$$

$$\left(\mathbf{X}^3\right)^T = \begin{bmatrix} 40.612 & 0 & 0 & 0 \\ 0 & 30.2235 & 0 & 0 \\ 0 & 0 & 19.351 & 0 \\ 0 & 0 & 0 & 9.803 \end{bmatrix} \left(\mathbf{X}^2\right)^T + \begin{bmatrix} 23653 & 808 & 1351 & 1720 \\ -1379 & -11740 & 837 & 376 \\ 1697 & 1104 & -7955 & 44 \\ 533 & 484 & 521 & -5888 \end{bmatrix}^T = \begin{bmatrix} 34.041 & 17.568 & 40.998 & 314.168 \\ -31.475 & -88.753 & -567.849 & 195.852 \\ -9.338 & -329.828 & 92.897 & 18.901 \\ 567.300 & 70.186 & -65.763 & -16.901 \end{bmatrix}$$

■

16. Cálculo de Derivada

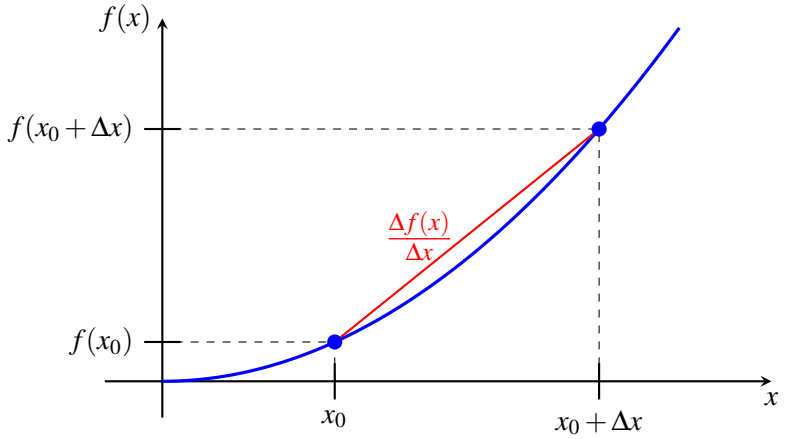
A derivada indica o quanto uma grandeza varia se um dos seus argumentos for perturbado. Seu cálculo é muito importante em diversas situações, como, por exemplo, otimização e plasticidade. Existem diversas abordagens para calcularmos derivadas utilizando calculo numérico e algoritmos. O obejtivo deste capítulo é mostrar os conceitos básicos de alguns dos métodos mais utilizados.

16.1 Diferenças finitas

A derivada de uma função de uma variável, $f(x)$, é definida como

$$\frac{df(x)}{dx} = \lim_{x \to a} \frac{f(x) - f(a)}{x - a}$$

onde $a = x + \Delta x$ e, geometricamente, pode ser interpretada como a reta tangente a função em um dado ponto x_0



Com base nesta interpretação geométrica, podemos observar que uma *aproximação* para o cálculo da derivada em torno de um dado ponto pode ser obtido por uma simples

interpretação geométrica, onde a hipotenusa do triângulo formado por um cateto adjacente de dimensão Δx e cateto oposto de dimensão $\Delta f = f(x_0 + \Delta x) - f(x_0)$ tende a reta tangente definida pela derivada no ponto x_0, quando $\Delta x \to 0$, tal que

$$\frac{df(x)}{dx} \simeq \left.\frac{\Delta f}{\Delta x}\right|_{x_0}.$$

Obviamente, esta definição matemática não pode ser aplicada diretamente para obtermos uma derivada numérica, pois se Δx for da ordem do epsilon da máquina, observamos que Δf não será corretamente calculado, levando a erros consideráveis de cálculo. Este erro é conhecido como erro de **arredondamento**, causado simplesmente pela precisão utilizada para descrever os números reais no computador.

Outro erro que pode aparecer está associado ao uso de uma perturbação Δx muito grande. Neste caso, a interpretação geométrica permite ver que a hipotenusa não coincide com a reta tangente no ponto x_0, levando a uma estimativa errada para o cálculo da derivada.

Se considerarmos uma expansão em série de Taylor de primeira ordem da função $f(x)$ em torno do ponto x_0, observamos que

$$f(x_0 + \Delta x) = f(x_0) + \frac{df}{dx}\Delta x + \mathscr{O}$$

onde $\mathscr{O}$ representa todos os termos de alta ordem. Estes termos tendem a zero muito mais rapidamente do que o termo de primeira ordem a medida que $\Delta x \to 0$, justificando a interpretação geométrica e a validade para pequenos valores de perturbação. Com esta expansão, podemos então avaliar o chamado erro de **truncamento**, associado ao erro ocasionado por perturbações muito grandes, pois, isolando a derivada

$$\frac{df}{dx} \simeq \frac{f(x_0 + \Delta x) - f(x_0) - \mathscr{O}}{\Delta x}.$$

Esta forma de calcularmos a derivada é chamada de Diferenças Finitas a Frente, pois utilizamos o cálculo da função em um ponto a frente do ponto atual, $f(x_0 + \Delta x)$. De forma análoga, é possível definirmos um cálculo para trás, conhecido como Diferenças Finitas para Trás, calculado como

$$\frac{df}{dx} \simeq \frac{f(x_0) - f(x_0 - \Delta x)}{\Delta x},$$

e as Diferenças Finitas Centrais, calculadas com

$$\frac{df}{dx} \simeq \frac{f(x_0 + \Delta x) - f(x_0 - \Delta x)}{2\Delta x}.$$

Cada uma destas formas tem vantagens e desvantagens. As diferenças finitas centrais, por exemplo, por utilizarem um intervalo mais amplo, são menos suscetíveis ao erro de arredondamento do que as diferenças finitas para trás e para frente. No entanto, necessitam do cálculo da função nestes dois pontos, o que acarreta um maior custo computacional.

16.1.1 Diferenças Finitas Complexas

Se expandirmos uma função em série de Taylor, com perturbação somente na parte complexa de x, tal que $\Delta x = 0 + \delta i$ então

$$f(x_0 + \delta i) = f(x_0) + \frac{df(x_0)}{dx}\delta i + \mathscr{O}.$$

Descartando os temos de alta ordem e considerando somente a parte complexa de cada uma das operações acima, temos que

$$\frac{df}{dx} \simeq \frac{\Im f(x_0 + \Delta x)}{\delta} = \frac{\Im f(x_0 + \delta i)}{\delta}$$

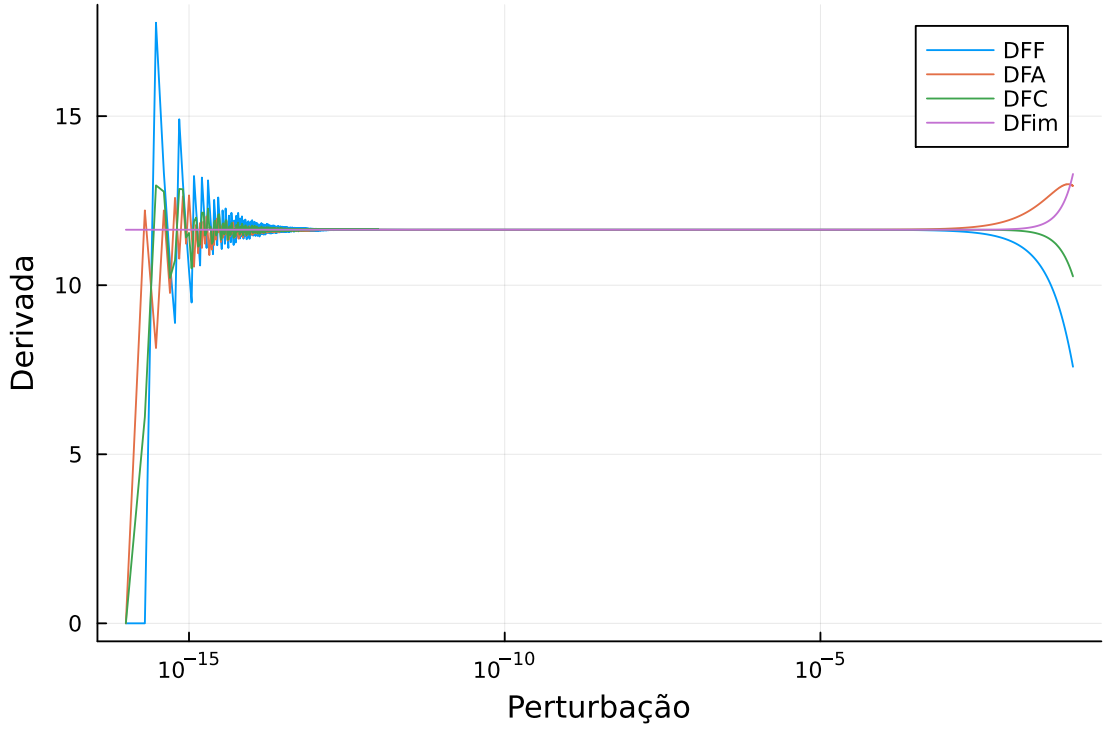
pois $\Im f(x_0) = 0$, uma vez que tanto o ponto atual quanto a função são originalmente reais. O interessante é que este procedimento faz com que não seja necessário o cálculo da subtração presente nos outros tipos de diferenças finitas, eliminando assim o erro de arredondamento. De fato, tem-se observado o correto cálculo da derivada mesmo com perturbação de ordem menor do que o epsilon da máquina. No entanto, observe que ainda utilizamos a hipótese de que os termos de alta ordem são desprezados, o que indica que essa abordagem é sucetível a erros de truncamento.

■ **Exemplo 16.1** Para ilustrar os conceitos vistos nas seções anteriores, vamos considerar a função $f(x) = 3\cos(x^2)^3$, com derivada analítica $\frac{df(x)}{dx} = -18x\cos\left(x^2\right)^2 \sin\left(x^2\right)$. No ponto $x = 2$, a derivada assume o valor analítico 11.640379 (considerando 6 casas decimais para fins de ilustração), que será utilizado como referência para este exemplo.

Utilizando uma perturbação $\delta = 1 \times 10^{-10}$, obtemos

- Diferenças finitas para frente: $\frac{f(2+\delta)-f(2)}{\delta} = 11.640378$;
- Diferenças finitas para trás: $\frac{f(2)-f(2-\delta)}{\delta} = 11.640382$;
- Diferenças finitas centrais: $\frac{f(2+\delta)-f(2-\delta)}{2\delta} = 11.64038$;
- Diferenças finitas complexas: $\frac{Imag(f(2+\delta i))}{\delta} = 11.640379$.

E o gráfico abaixo ilustra o comportamento de cada um dos métodos em relação ao valor da perturbação (DFF significa Diferenças Finitas para Frente, DFA para trás, DFC centrais e DFim complexas). Observe que a resposta calculada por diferenças finitas complexas não apresenta os erros de arredondamento (na faixa de perturbações estudada), mas ainda apresenta o erro de truncamento.



■

16.2 Cálculo do Vetor Gradiente

O gradiente de uma função de n variáveis $f(\mathbf{x})$ é definido como

$$\nabla f(\mathbf{x}) = \begin{Bmatrix} \frac{df}{dx_1} \\ \frac{df}{dx_2} \\ \vdots \\ \frac{df}{dx_n} \end{Bmatrix}$$

sendo que cada uma das posições pode ser calculada utilizando um dos procedimentos unidimensionais descritos acima. O algoritmo 40 ilustra o cálculo do gradiente de uma função de n variáveis utilizando diferenças finitas centrais.

Algoritmo 40: Cálculo do vetor gradiente utilizando Diferenças Finitas Centrais

```
1  GradDF(f,x,n,δ)
       Entrada: f função ℝ^n → ℝ
       Entrada: x vetor n × 1
       Entrada: δ perturbação
2      Define o vetor gradiente
3      D ← 0_n
4      Laço pelas componentes de x
5      for i ∈ {1,...,n} do
6          Guarda o valor atual de x_i
7          x_b ← x(i)
8          Perturba a posição x_i para frente
9          x(i) ← x_b + δ
10         Calcula valor para frente
11         f_2 ← f(x)
12         Perturba a posição x_i para trás
13         x(i) ← x_b − δ
14         Calcula valor para trás
15         f_1 ← f(x)
16         Calcula a derivada
17         D(i) ← (f_2 − f_1)/(2δ)
18         Restaura a posição x_i
19         x(i) ← x_b
20     Retorna o vetor gradiente
21     return D
```

16.3 Derivada de expressões envolvendo matrizes

De especial interesse para a área de mecânica computacional é a derivada de expressões que relacionam matrizes e vetores. Vamos explorar algumas expressões.

16.3.1 Derivada de um mapeamento $\mathbb{R}^n \to \mathbb{R}^m$ em relação a um elemento do domínio.

Seja o mapeamento $\mathbf{A} : \mathbb{R}^n \to \mathbb{R}^m$

$$\mathbf{v}_{m\times 1} = \mathbf{A}_{m\times n}\mathbf{x}_{n\times 1}$$

ou

$$\mathbf{v} = \begin{Bmatrix} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \end{Bmatrix}.$$

Assim,

$$\frac{d\mathbf{v}}{d\mathbf{x}} = \begin{bmatrix} \frac{dv_1}{dx_1} & \frac{dv_1}{dx_2} & \cdots & \frac{dv_1}{dx_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dv_m}{dx_1} & \frac{dv_m}{dx_2} & \ldots & \frac{dv_m}{dx_n} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} = \mathbf{A}.$$

Em notação indicial, a linha i de $\mathbf{v}$ é escrita como

$$v_i = \sum_{j=1}^{n} a_{ij}x_j$$

tal que

$$\frac{dv_i}{dx_p} = \sum_{j=1}^{n} a_{ij}\frac{dx_j}{dx_p} = \sum_{j=1}^{n} a_{ij}\delta_{jp} = a_{ip},\ i = 1..m, p = 1..n.$$

16.3.2 Composição de operações lineares

Sejam duas operações lineares

$$\mathbb{R}^n \to \mathbb{R}^m : \quad \mathbf{v} = \mathbf{A}\mathbf{x}$$

e

$$\mathbb{R}^m \to \mathbb{R}^o : \quad \mathbf{w} = \mathbf{B}\mathbf{v},$$

tal que a composição das operações é dada por

$$\mathbb{R}^n \to \mathbb{R}^o : \quad \mathbf{w} = \mathbf{B}\mathbf{A}\mathbf{x}.$$

Indicialmente, podemos escrever a composição como

$$w_i = \sum_{j=1}^{o} b_{ij} \sum_{k=1}^{n} a_{jk}x_k$$

e a derivada da linha i de $\mathbf{w}$ em relação a x_p é dada por

$$\frac{dw_i}{dx_p} = \sum_{j=1}^{o} b_{ij} \sum_{k=1}^{n} a_{jk}\frac{dx_k}{dx_p} = \sum_{j=1}^{o} b_{ij} \sum_{k=1}^{n} a_{jk}\delta_{kp} = \sum_{j=1}^{o} b_{ij}a_{jp}$$

tal que

$$\frac{d\mathbf{w}}{d\mathbf{x}} = \begin{bmatrix} \frac{dw_1}{dx_1} & \frac{dw_1}{dx_2} & \cdots & \frac{dw_1}{dx_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dw_m}{dx_1} & \frac{dw_m}{dx_2} & \ldots & \frac{dw_m}{dx_n} \end{bmatrix} = \begin{bmatrix} b_{11}a_{11} + \ldots + b_{1o}a_{o1} & \ldots & b_{11}a_{1n} + \ldots + b_{1o}a_{on} \\ \vdots & \ddots & \vdots \\ b_{m1}a_{11} + \ldots + b_{mo}a_{o1} & \ldots & b_{m1}a_{1n} + \ldots + b_{mo}a_{on} \end{bmatrix} = \mathbf{B}\mathbf{A}.$$

16.3.3 Derivada de uma forma bilinear

Seja a forma bilinear

$$\mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}: \quad b = \mathbf{v}^T \mathbf{B} \mathbf{x}$$

que, indicialmente, pode ser escrita com

$$b = \sum_{i=1}^{m} \sum_{j=1}^{n} v_i b_{ij} x_j.$$

Assim, avaliando a derivada de b em relação a x_p

$$\frac{db}{dx_p} = \sum_{i=1}^{m} \sum_{j=1}^{n} v_i b_{ij} \frac{dx_j}{dx_p} = \sum_{i=1}^{m} \sum_{j=1}^{n} v_i b_{ij} \delta_{jp} = \sum_{i=1}^{m} v_i b_{ip}$$

tal que

$$\frac{db}{d\mathbf{x}} = \begin{Bmatrix} v_1 b_{11} + \ldots + v_m b_{m1} \\ v_1 b_{12} + \ldots + v_m b_{m2} \\ \vdots \\ v_1 b_{1n} + \ldots + v_m b_{mn} \end{Bmatrix} = \mathbf{B}^T \mathbf{v}.$$

que é um vetor de dimensão $n \times 1$. Observe que a derivada da mesma forma bilinear em relação a $\mathbf{v}$ pode ser calculada com (lembrando que b é um escalar)

$$b = \mathbf{v}^T \mathbf{B} \mathbf{x} = \left(\mathbf{v}^T \mathbf{B} \mathbf{x}\right)^T = \mathbf{x}^T \mathbf{B}^T \mathbf{v}$$

tal que podemos utilizar o resultado original, resultando em

$$\frac{db}{d\mathbf{v}} = \mathbf{B}\mathbf{x}.$$

É importante notarmos que este resultado é diretamente aplicável para o produto interno no $\mathbb{R}^n$ (lembrando que os axiomas do produto interno requerem que $\mathbf{B}$ seja positivo-definida). Em especial, no produto interno canônico, $\mathbf{B} = \mathbf{I}$, tal que

$$\frac{d\langle \mathbf{v}, \mathbf{x} \rangle}{d\mathbf{x}} = \mathbf{v}$$

e

$$\frac{d\langle \mathbf{v}, \mathbf{x} \rangle}{d\mathbf{v}} = \mathbf{x}.$$

16.3.4 Derivada de uma forma quadrática

Seja a forma quadrática

$$\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}: \quad q = \mathbf{x}^T \mathbf{Q} \mathbf{x}$$

com $\mathbf{Q} = \mathbf{Q}^T$ (simétrica). A simetria da matriz de coeficientes é consequência do fato de que

$$q = \mathbf{x}^T \mathbf{Q} \mathbf{x} = \left(\mathbf{x}^T \mathbf{Q} \mathbf{x}\right)^T = \mathbf{x}^T \mathbf{Q}^T \mathbf{x} \implies \mathbf{Q} = \mathbf{Q}^T.$$

Indicialmente

$$q = \sum_{i=1}^{n} x_i \sum_{j=1}^{n} q_{ij} x_j.$$

Assim,

$$\frac{dq}{dx_p} = \sum_{i=1}^{n} \delta_{ip} \sum_{j=1}^{n} q_{ij} x_j + \sum_{i=1}^{n} x_i \sum_{j=1}^{n} q_{ij} \delta_{jp}$$

ou

$$\frac{dq}{dx_p} = \sum_{j=1}^{n} q_{pj} x_j + \sum_{i=1}^{n} x_i q_{ip}.$$

Como a matriz da forma quadrática é simétrica, podemos rearranjar os somatórios para a mesma variável, tal que

$$\frac{dq}{dx_p} = \sum_{j=1}^{n} (q_{pj} x_j + q_{pj} x_j) = 2 \sum_{j=1}^{n} q_{pj} x_j,$$

ou

$$\frac{dq}{d\mathbf{x}} = 2\mathbf{Q}\mathbf{x}.$$

16.4 Derivada Automática

Considere uma função $f : \mathbb{R}^n \to \mathbb{R}^m$

$$\mathbf{y} = f(\mathbf{x})$$

com

$$f(\mathbf{x}) = f_1 \circ f_2 \circ f_3$$

ou

$$\begin{aligned} \mathbf{a} &= f_3(\mathbf{x}) \\ \mathbf{b} &= f_2(\mathbf{a}) \\ \mathbf{y} &= f_1(\mathbf{b}) \end{aligned}$$

Com isso, podemos escrever a derivada de $\mathbf{y}$ em relação a $\mathbf{x}$ como

$$\left.\frac{d\mathbf{y}}{d\mathbf{x}}\right|_{m\times n} = \left.\frac{\partial f_1(\mathbf{b})}{\partial \mathbf{b}}\right|_{m\times|b|} \left.\frac{\partial f_2(\mathbf{a})}{\partial \mathbf{a}}\right|_{|b|\times|a|} \left.\frac{df_3(\mathbf{x})}{d\mathbf{x}}\right|_{|a|\times n}.$$

Podemos pensar em realizar essas operações de duas formas, que chamamos de para frente, *forward*, ou para trás, *backward* ou *back propagation*.

No modo *backwards* realizamos as operações como

$$\left.\frac{d\mathbf{y}}{d\mathbf{x}}\right|_{m\times n} = \left.\frac{\partial f_1(\mathbf{b})}{\partial \mathbf{b}}\right|_{m\times|b|} \left.\frac{\partial f_2(\mathbf{a})}{\partial \mathbf{a}}\right|_{|b|\times|a|} \left.\frac{df_3(\mathbf{x})}{d\mathbf{x}}\right|_{|a|\times n},$$

na sequência $m \times |b|$, $m \times |b| \cdot |b| \times |a|$ e $m \times |b| \cdot |b| \times |a| \cdot |a| \times n$. No modo *forward* agrupamos as operações como

$$\left.\frac{d\mathbf{y}}{d\mathbf{x}}\right|_{m\times n} = \left(\left.\frac{df_3(\mathbf{x})}{d\mathbf{x}}\right|^T_{|a|\times n} \left.\frac{\partial f_2(\mathbf{a})}{\partial \mathbf{a}}\right|^T_{|b|\times|a|} \left.\frac{\partial f_1(\mathbf{b})}{\partial \mathbf{b}}\right|^T_{m\times|b|} \right)^T,$$

e, por termos revertido a sequência de operações, devemos transpor cada um dos termos e, depois, transpor toda a operação. Assim, a sequência de operações é $(|a| \times |n|)^T$, $(|a| \times |n|)^T \cdot (|b| \times |a|)^T$, $(|a| \times |n|)^T \cdot (|b| \times |a|)^T \cdot (m \times |b|)^T$ e, por fim, transpor o resultado.

Embora pareçam ser equivalentes (e são, pois tem que dar o mesmo resultado), devemos prestar atenção nas dimensões dos produtos entre as matrizes. No caso *forward* primeiro realizamos a multiplicação $n \times |a|$ e no *backwards* a $m \times |b|$. Assim, cada uma das abordagens será mais vantajosa a depender das dimensões de m, n e das dimensões das operações intermediárias. Uma regra que se observa é que se a dimensão da entrada, n, for maior do que a de saída, m, então a abordagem para trás é mais eficiente, do contrário, a abordagem para frente se torna mais vantajosa.

■ **Exemplo 16.2** Vamos começar com um exemplo unidimensional, i.e, $m = n = 1$. Seja a função

$$f(x) = y = \sin(x^2)^3 + 2x,$$

com derivada

$$\frac{dy}{dx} = 6x\cos(x^2)\sin(x^2)^2 + 2.$$

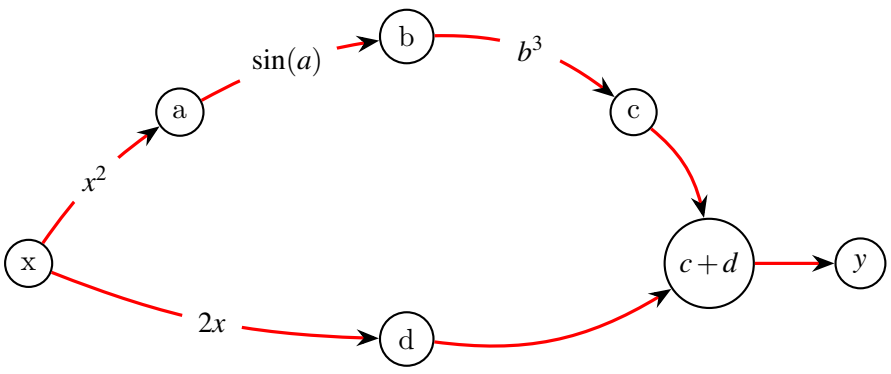
Podemos identificar as operações:

$$x \to x^2 \to \sin(x^2) \to \sin(x^2)^3$$
$$x \to 2x$$

ou

$$x \to a = x^2 \to b = \sin(a) \to c = b^3$$
$$x \to d = 2x$$
$$c + d \to y$$

com gráfico



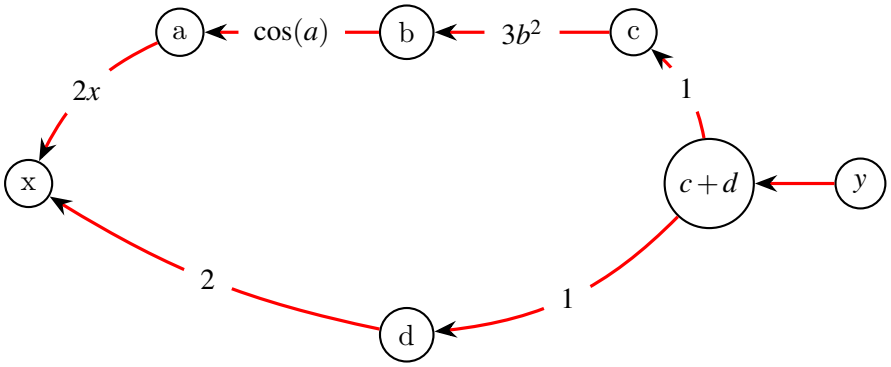
Da sequência de operações, podemos verificar que

$$\frac{\partial a}{\partial x} = 2x \quad \frac{\partial b}{\partial a} = \cos(a)$$
$$\frac{\partial c}{\partial b} = 3b^2 \quad \frac{\partial d}{\partial x} = 2$$
$$\frac{\partial y}{\partial c} = 1 \quad \frac{\partial y}{\partial d} = 1$$

A derivada para trás é equivalente a regra da cadeia

$$\frac{dy}{dx} = \left(\frac{\partial y}{\partial c}\frac{\partial c}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial x}\right) + \left(\frac{\partial y}{\partial d}\frac{\partial d}{\partial x}\right)$$

com gráfico



ou

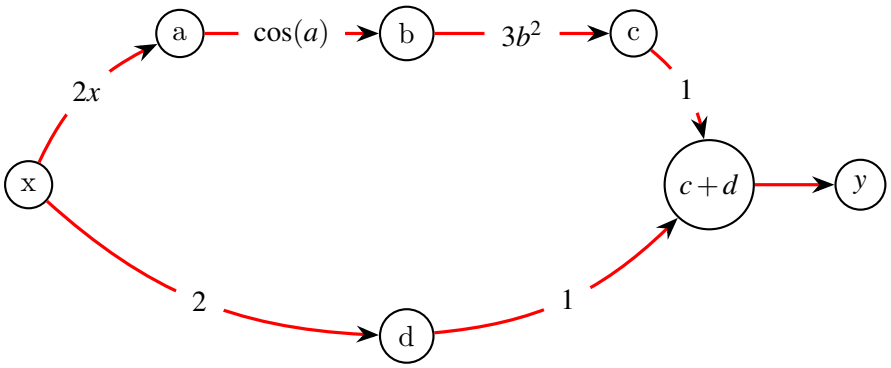
$$\frac{dy}{dx} = \underbrace{(1)\left(3b^2\right)(\cos(a))(2x)}_{\text{ramo de cima}} + \underbrace{(1)(2)}_{\text{ramo de baixo}}$$

e, substituindo as expressões de a e b

$$\frac{dy}{dx} = \left(3(\sin(x^2)^2\right)\left(\cos(x^2)\right)(2x) + (1)(2) = 6x\cos(x^2)\sin(x^2)^2 + 2.$$

A outra alternativa é fazer o procedimento para frente. Neste caso,

$$\frac{dy}{dx} = \left(\frac{\partial a}{\partial x}\frac{\partial b}{\partial a}\frac{\partial c}{\partial b}\frac{\partial y}{\partial c}\right) + \left(\frac{\partial d}{\partial x}\frac{\partial y}{\partial d}\right)$$



ou

$$\frac{dy}{dx} = \underbrace{(2x)(\cos(a))\left(3b^2\right)(1)}_{\text{ramo de cima}} + \underbrace{(2)(1)}_{\text{ramo de baixo}}$$

que dá (ufa!) o mesmo resultado. ∎

■ **Exemplo 16.3** Seja a operação $f : \mathbb{R}^n \to \mathbb{R}^m$

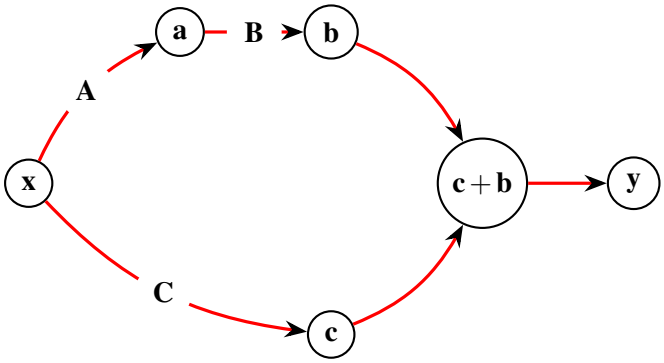
$$\mathbf{y}_{m\times 1} = \mathbf{B}(\mathbf{A}\mathbf{x}) + \mathbf{C}\mathbf{x}$$

onde $\mathbf{A}_{n\times n}, \mathbf{B}_{m\times n}$ e $\mathbf{C}_{m\times n}$ são matrizes.

Podemos identificar as operações:

$$\mathbf{a} = \mathbf{A}\mathbf{x}$$
$$\mathbf{b} = \mathbf{B}\mathbf{a}$$
$$\mathbf{c} = \mathbf{C}\mathbf{x}$$
$$\mathbf{d} = \mathbf{b} + \mathbf{c}$$
$$\mathbf{y} = \mathbf{d}$$

com gráfico



Tal que a derivada *backwards* pode ser escrita como

$$\frac{d\mathbf{y}}{d\mathbf{x}} = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{b}}\frac{\partial \mathbf{b}}{\partial \mathbf{a}}\right)\frac{\partial \mathbf{a}}{\partial \mathbf{x}} + \left(\frac{\partial \mathbf{y}}{\partial \mathbf{c}}\right)\frac{\partial \mathbf{c}}{\partial \mathbf{x}}$$

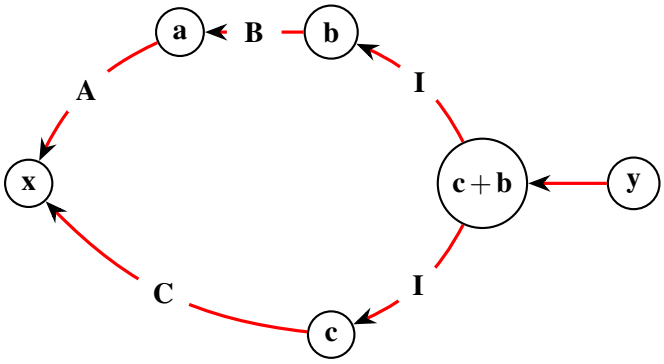
onde

$$\frac{\partial \mathbf{y}}{\partial \mathbf{b}} = \mathbf{I} \quad \frac{\partial \mathbf{b}}{\partial \mathbf{a}} = \mathbf{B} \quad \frac{\partial \mathbf{a}}{\partial \mathbf{x}} = \mathbf{A}$$
$$\frac{\partial \mathbf{y}}{\partial \mathbf{c}} = \mathbf{I} \quad \frac{\partial \mathbf{c}}{\partial \mathbf{x}} = \mathbf{C}$$

tal que a derivada será

$$\frac{d\mathbf{y}}{d\mathbf{x}} = \mathbf{I}\mathbf{B}\mathbf{A} + \mathbf{I}\mathbf{C} = \mathbf{B}\mathbf{A} + \mathbf{C}$$

ou, graficamente,



A outra alternativa é fazer o procedimento *forward*. Neste caso,

$$\frac{d\mathbf{y}}{d\mathbf{x}} = \left(\frac{\partial \mathbf{a}}{\partial \mathbf{x}}^T \frac{\partial \mathbf{b}}{\partial \mathbf{a}}^T \frac{\partial \mathbf{y}}{\partial \mathbf{b}}^T\right)^T + \left(\frac{\partial \mathbf{c}}{\partial \mathbf{x}}^T \frac{\partial \mathbf{y}}{\partial \mathbf{c}}^T\right)^T$$

tal que

$$\left(\mathbf{A}^T \mathbf{B}^T \mathbf{I}^T\right)^T = \mathbf{IBA}$$

seguido de

$$\left(\mathbf{C}^T \mathbf{I}^T\right)^T = \mathbf{IC}.$$

■

■ **Exemplo 16.4** Seja a operação $\mathbb{R}^n \to \mathbb{R}$ indicada por

$$y = \alpha \mathbf{x}^T \mathbf{A}\mathbf{x}$$

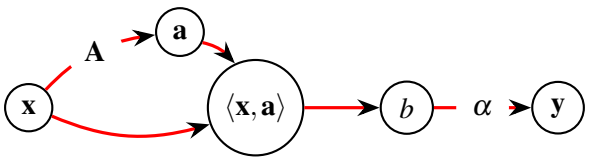
com $\alpha \in \mathbb{R}$, $\mathbf{A}_{n\times n}$ simétrica e $\mathbf{x}_n$. Com os resultados da seção anterior sabemos que

$$\frac{dy}{d\mathbf{x}} = 2\alpha \mathbf{A}\mathbf{x}.$$

Podemos identificar as operações

$$\mathbf{a} = \mathbf{A}\mathbf{x}$$
$$b = \langle \mathbf{x}, \mathbf{a} \rangle$$
$$c = \alpha b$$
$$y = c$$

com representação



Com isso, a derivada *backward* pode ser obtida com

$$\frac{dy}{d\mathbf{x}} = \frac{dy}{dc}\frac{dc}{db}\left(\frac{db}{d\mathbf{x}} + \frac{db}{d\mathbf{a}}\frac{d\mathbf{a}}{d\mathbf{x}}\right)$$

em que

$$\frac{db}{d\mathbf{x}} = \mathbf{a} = \mathbf{A}\mathbf{x}$$

$$\frac{db}{d\mathbf{a}} = \mathbf{x}^T$$

e

$$\frac{d\mathbf{a}}{d\mathbf{x}} = \mathbf{A}$$

As derivadas $\frac{dc}{db} = \alpha$ e $\frac{dy}{dc} = 1$ são triviais. Com isso, obtemos

$$\frac{dy}{d\mathbf{x}} = \alpha\left(\mathbf{A}\mathbf{x} + \mathbf{x}^T\mathbf{A}\right)$$

e, como a a matriz $\mathbf{A}$ é simétrica

$$\frac{dy}{d\mathbf{x}} = 2\alpha\mathbf{A}\mathbf{x}.$$

Utilizando a derivada para frente

$$\frac{dy}{d\mathbf{x}} = \left(\frac{d\mathbf{a}}{d\mathbf{x}}^T \frac{db}{d\mathbf{a}}^T \frac{dy}{db}\right)^T + \left(\frac{db}{d\mathbf{x}}^T \frac{dy}{db}\right)^T$$

e, usando os cálculos que já fizemos

$$\frac{dy}{d\mathbf{x}} = \left(\mathbf{A}^T\mathbf{x}\alpha\right)^T + \left((\mathbf{A}\mathbf{x})^T\alpha\right)^T = \alpha\mathbf{x}^T\mathbf{A} + \alpha\mathbf{A}\mathbf{x} = 2\alpha\mathbf{A}\mathbf{x}.$$

■

16.5 Números Duais

A derivada automática para frente pode ser calculada de forma utilizando o conceito de número dual.

Definição 16.5.1 Número dual
Definimos o número dual

$$d = d_r + \varepsilon d_d$$

em que $d_r, d_d \in \mathbb{R}$ e $\varepsilon^2 = 0, \varepsilon \neq 0$.

Podemos equipar os números duais com axiomas básicos. Sejam dois números duais a e b e um escalar α:

- Soma:

$$a + b = (a_r + \varepsilon a_d) + (b_r + \varepsilon b_d) = (a_r + b_r) + \varepsilon\,(a_d + b_d)$$

- Multiplicação por um escalar:

$$\alpha a = \alpha\,(a_r + \varepsilon a_d) = (\alpha a_r) + \varepsilon\,(\alpha a_d)$$

- Multiplicação de dois números duais:

$$ab = (a_r + \varepsilon a_d)\,(b_r + \varepsilon b_d) = a_r b_r + \varepsilon a_r b_d + \varepsilon a_d b_r + \varepsilon^2 a_d b_d = (a_r b_r) + \varepsilon\,(a_r b_d + a_d b_r)$$

- Divisão de dois números duais:

$$\frac{a}{b} = \frac{a_r + \varepsilon a_d}{b_r + \varepsilon b_d} = \frac{a_r}{b_r} + \varepsilon \left(\frac{a_d b_r - a_r b_d}{b_r} \right), \quad b_r \neq 0.$$

sendo que esse último resultado pode ser obtido multiplicando o nominador e o denominador por $b^* = b_r - \varepsilon b_d$ e fatorando os resultados.

Seja uma função $f(x)$. Podemos calcular a derivada da função em relação a x utilizando uma expansão em série de Taylor

$$f(x_r + \varepsilon x_d) \approx f(x_r) + \frac{df}{dx}\varepsilon x_d + \frac{1}{2}\frac{d^2 f}{dx^2}\varepsilon^2 x_d^2 + \frac{1}{3!}\frac{d^3 f}{dx^3}\varepsilon^2 \varepsilon x_d^3 + ...$$

ou seja, todos os termos com potências maires do que ε serão nulas e podemos escrever a **igualdade**

$$f(x_r + \varepsilon x_d) = f(x_r) + \varepsilon \frac{df}{dx} x_d$$

que é um número dual! Ou seja, a avaliação de uma função em números duais terá a parte real com o valor da função e a parte dual da solução será proporcional à derivada da função em relação a parte real do número.

■ **Exemplo 16.5** Seja a função $f(x) = 3x^2$. Para obtermos a derivada em relação a x, podemos calcular a função em

$$f(x_r + \varepsilon x_d) = 3(x_r + \varepsilon x_d)(x_r + \varepsilon x_d) = 3\left((x_r x_r) + \varepsilon\left(x_r x_d + x_d x_r\right)\right) = 3x_r^2 + \varepsilon\left(2x_r\right)x_d$$

onde $3x^2$ é o valor da função e $2x_r$ é o valor da derivada.

■

Com isso, podemos realizar a derivada para frente de forma bem simples apenas operando com números duais, já que a regra da cadeia irá se aplicar naturalmente ao cálculo exato da derivada.

■ **Exemplo 16.6** Sejam as funções $f(x) = 2x$ e $g(f) = f^2$, tal que $g(f(x)) = (2x)^2 = 4x^2$. Sabemos que a derivada da composição é $8x$, mas vamos avaliar por números duais. Assim,

$$f(x_r + \varepsilon x_d) = 2(x_r + \varepsilon x_d) = 2x_r + \varepsilon(2)x_d$$

e

$$g(f_r + \varepsilon f_d) = (2x_r + \varepsilon(2)x_d)(2x_r + \varepsilon(2)x_d) = 4x_r^2 + \varepsilon\left(8x_r\right)x_d$$

■

Com base nos exemplo, podemos definir operações com funções mais complicadas somente observando o padrão. Por exemplo, para a função seno

$$\sin(x_r + \varepsilon x_d) = \sin(x_r) + \varepsilon \cos(x_r) x_d$$

pois sabemos que a derivada do $\sin(x)$ em relação a x é a função $\cos(x)$. Da mesma forma,

$$\exp(x_r + \varepsilon x_d) = \exp(x_r) + \varepsilon \exp(x_r) x_d$$

e diversas definições de operações sobre números duais podem ser encontradas na biblioteca LDUAL[1].

[1] https://github.com/CodeLenz/LDual.jl

■ **Exemplo 16.7** Vamos calcular a derivada da função $f(x) = \sin(x^2)^3$, mas agora utilizando números duais. Seguindo a lógica para frente do cálculo

$$x^2: \quad (x_r + \varepsilon x_d)^2 = x_r^2 + \varepsilon 2 x_r x_d$$

tal que

$$\sin(x^2): \quad \sin(x_r^2 + \varepsilon 2 x_r x_d) = \sin(x_r^2) + \varepsilon \cos(x_r^2) 2 x_r x_d.$$

Como

$$x^3: \quad (x_r + \varepsilon x_d)^3 = x_r^3 + \varepsilon 3 x_r^2 x_d$$

então

$$\sin(x^2)^3: \quad \sin(x_r^2 + \varepsilon 2 x_r x_d)^3 = \sin(x_r^2)^3 + \varepsilon \left(3 \sin(x_r^2)^2 \cos(x_r^2) 2 x_r\right) x_d$$

tal que a derivada da função será $6x \sin(x^2)^2 \cos(x^2)$. ■

■ **Exemplo 16.8** As operações com números duais também se aplicam a operações com matrizes e vetores. Seja a operação $f : \mathbb{R}^n \to \mathbb{R}^m$

$$\mathbf{y}_{m\times 1} = \mathbf{B}(\mathbf{A}\mathbf{x}) + \mathbf{C}\mathbf{x}$$

onde $\mathbf{A}_{n\times n}, \mathbf{B}_{m\times n}$ e $\mathbf{C}_{m\times n}$ são matrizes. Já estudamos este problema em um exemplo anterior deste mesmo capítulo.

Inicializando o vetor $\mathbf{x}$ como dual

$$\mathbf{x} = \mathbf{x}_r + \varepsilon \mathbf{x}_d$$

teremos

$$\mathbf{a} = \mathbf{A}(\mathbf{x}_r + \varepsilon \mathbf{x}_d) = \mathbf{A}\mathbf{x}_r + \varepsilon \mathbf{A}\mathbf{x}_d = \mathbf{a}_r + \varepsilon \mathbf{a}_d$$

tal que

$$\mathbf{b} = \mathbf{B}(\mathbf{A}\mathbf{x}_r + \varepsilon \mathbf{A}\mathbf{x}_d) = \mathbf{B}\mathbf{A}\mathbf{x}_r + \varepsilon \mathbf{B}\mathbf{A}\mathbf{x}_d.$$

Por fim, verificamos que

$$\mathbf{c} = \mathbf{C}(\mathbf{x}_r + \varepsilon \mathbf{x}_d) = \mathbf{C}\mathbf{x}_r + \varepsilon \mathbf{C}\mathbf{x}_d$$

e, somando as parcelas $\mathbf{b}$ e $\mathbf{c}$, obtemos tanto o resultado da operação quanto da derivada

$$\mathbf{y} = \mathbf{y}_r + \varepsilon \mathbf{y}_d = (\mathbf{B}\mathbf{A}\mathbf{x}_r + \mathbf{C}\mathbf{x}_r) + \varepsilon (\mathbf{B}\mathbf{A} + \mathbf{C}) \mathbf{x}_d$$

tal que

$$\frac{d\mathbf{y}}{d\mathbf{x}} = \mathbf{B}\mathbf{A} + \mathbf{C}.$$

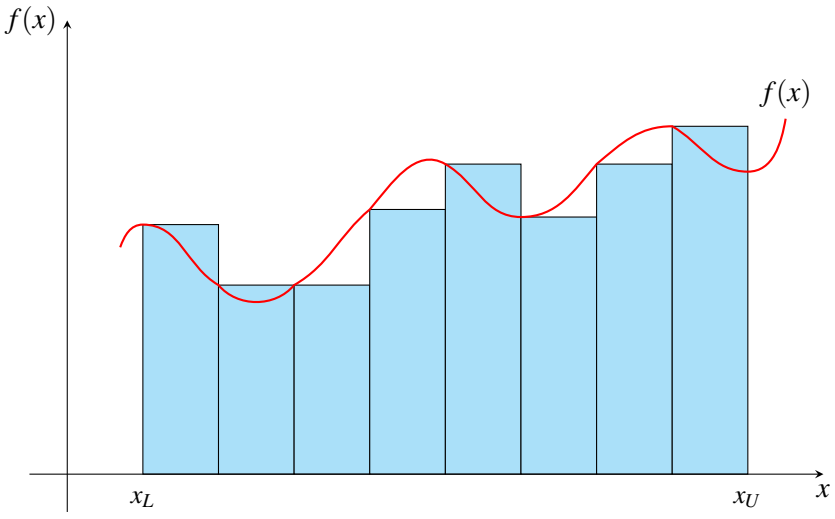
■

17. Integração Numérica

Dada uma função $f(x)$, temos, por definição, que a integral de Cauchy da função no intervalo $[x_L, x_U]$ é dada por

$$\int_{x_L}^{x_U} f(x)\,dx = \lim_{\Delta x \to 0} \sum_{x=x_L}^{x=x_U} f(x)\Delta x$$

ou seja, o somatório da área de infinitos retângulos de base Δx e altura $f(x)$. Obviamente esta definição matemática não é muito prática para uma implementação numérica, uma vez que deveríamos avaliar o valor da função em um número (idealmente) infinito de pontos. Para visualizarmos o erro quando usamos poucos intervalos, considere o gráfico da função abaixo com 8 retângulos



Esse procedimento está ilustrado no Alg. 41.

■ **Exemplo 17.1** Obtenha a integral de $f(x) = 3x^2 - 6x^3$ com $x \in [0,1]$.

Analiticamente esta integral tem como solução

$$\int_0^1 3x^2 - 6x^3\, dx = -\frac{1}{2}.$$

Se dividirmos o intervalo em 10 retângulos iremos obter como aproximação para a integral:

$$\left(3x^2 - 6x^3\big|_0 0.1\right) + \left(3x^2 - 6x^3\big|_{0.1} 0.1\right) + \left(3x^2 - 6x^3\big|_{0.2} 0.1\right) + ... + \left(3x^2 - 6x^3\big|_{1.0} 0.1\right) = -0.66$$

com 100 retângulos -0.5151 e, com 1000 retângulos, -0.501501. Obviamente, esta estratégia, que podemos chamar de método dos retângulos, é muito pobre e só consegue integrar corretamente (isto é, com um número de amostras menor do que ∞) funções constantes. ■

17.1 Regra dos Trapézios

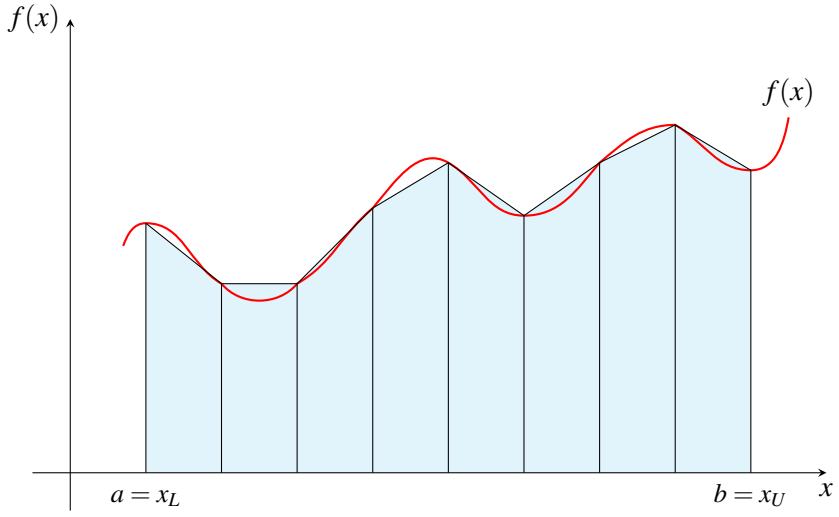
Outra estratégia é a famosa regra dos trapézios. Neste método realizamos uma interpolação linear entre dois pontos da curva (função) e calculamos a área abaixo do trapézio. Desta forma, considerando dois pontos x_i e x_{i+1} e seus respectivos valores $f(x_i)$ e $f(x_{i+1})$ teremos como área abaixo do trapézio

$$dA_i = f(x_i)\Delta x + \frac{1}{2}\left(f(x_{i+1}) - f(x_i)\right)\Delta x$$

e, como área total

$$A = \sum_{i=1}^{n-1} dA_i$$

Da imagem abaixo, podemos ver que essa simples modificação permite uma melhor descrição da área abaixo da curva, para o mesmo número de divisões utilizado no método dos retângulos.



■ **Exemplo 17.2** Considerando os mesmos dados do exemplo anterior (método dos retângulos) e um $\Delta x = 0.1$ teremos

$$f(0)0.1 + \frac{1}{2}\left(f(0.1) - f(0)\right)0.1 + f(0.1)0.1 + \frac{1}{2}\left(f(0.2) - f(0.1)\right)0.1 + ... + f(0.9)0.1 + \frac{1}{2}\left(f(1.0) - f(0.9)\right)0.1 = -0.51$$

ou, com $\Delta x = 1 \times 10^{-2}$ obtemos -0.5001 e, com $\Delta x = 1 \times 10^{-3}$ obtemos -0.500001, mostrando como este método é muito mais acurado do que o método de ordem zero. No entanto, este método consegue integrar corretamente uma função de primeira ordem. ■

Este método é apresentado no Alg. 42.

Desta forma, fica claro que se utilizarmos aproximações de alta ordem para cada uma das áreas diferenciais a serem somadas, iremos obter aproximações cada vez melhores mesmo com um menor número de intervalos. Esta é a ideia por trás dos métodos de integração por interpolação.

17.2 Regra de Simpson de Segunda Ordem (Primeira Regra de Simpson)

Se considerarmos três pontos x_i, x_{i+1} e x_{i+2} e seus respectivos valores de f, podemos realizar uma aproximação quadrática para o comportamento da função, na forma

$$\tilde{f}(\zeta) = a + b\zeta + c\zeta^2$$

em que ζ é uma coordenada local, variando de 0 até $2\Delta x$. Os coeficientes são obtidos por

$$\begin{aligned}
&\tilde{f}(\zeta = 0) = a = f(x_i) \\
&\tilde{f}(\zeta = \Delta x) = a + b\Delta x + c\,(\Delta x)^2 = f(x_{i+1}). \\
&\tilde{f}(\zeta = 2\Delta x) = a + b2\Delta x + c\,(2\Delta x)^2 = f(x_{i+2})
\end{aligned}$$

Como este pequeno pedaço da função tem área conhecida e igual a

$$dA_i = \int_0^{2\Delta x} a + b\zeta + c\zeta^2\, d\zeta$$

podemos realizar a integração analítica e substituir os coeficientes, obtendo

$$dA_i = \frac{\Delta x}{3}\left[f(x_i) + 4f(x_{i+1}) + f(x_{i+2})\right].$$

Este método, por sua natureza, é capaz de integrar corretamente funções constantes, lineares e quadráticas.

■ **Exemplo 17.3** Usando os mesmos dados dos exemplos anteriores e com $\Delta x = 0.1$ teremos

$$\frac{0.1}{3}\left[f(0) + 4f(0.1) + f(0.2)\right] + \frac{0.1}{3}\left[f(0.2) + 4f(0.3) + f(0.4)\right] + \ldots + \frac{0.1}{3}\left[f(0.8) + 4f(0.9) + f(1.0)\right] = -0.5$$

sendo que, para os dados deste exemplo, este resultado já é obtido com $\Delta x = 0.5$.

■

O procedimento está ilustrado no Alg. (43).

17.3 Regra de Simpson de Terceira Ordem (Segunda Regra de Simpson)

Neste caso, aproximamos a função por meio de uma função cúbica, na forma

$$\tilde{f}(\zeta) = a + b\zeta + c\zeta^2 + d\zeta^3$$

e, portanto, necessitamos de 4 pontos para cobrir cada um dos intervalos. Realizando um procedimento análogo ao da Primeira Regra de Simpson obtemos

$$dA_i = \frac{3\Delta x}{8}\left[f(x_i) + 3f(x_{i+1}) + 3f(x_{i+2}) + f(x_{i+3})\right]$$

conseguindo integrar exatamente até funções cúbicas, como é o caso do nosso exemplo. O procedimento está ilustrado no Alg. (44).

Algoritmo 41: Integral pelo método dos retângulos

```
1  Retangulos(f,x_l,x_u,n)
       Entrada: f função
       Entrada: x_l limite inferior
       Entrada: x_l limite superior
       Entrada: n número de divisões
2      Calcula o passo
3      Δx ← (x_u − x_l)/n
4      Inicializa a área
5      area ← 0
6      Laço nos intervalos
7      for i ∈ {0,...,n} do
8          area ← area + f(x_l + i·Δx)·Δx
9      return area
```

Algoritmo 42: Integral pelo método dos trapézios

```
1  Trapezios(f,x_l,x_u,n)
       Entrada: f função
       Entrada: x_l limite inferior
       Entrada: x_l limite superior
       Entrada: n número de divisões
2      Calcula o passo
3      Δx ← (x_u − x_l)/n
4      Inicializa a área
5      area ← 0
6      Laço pelos trapézios
7      for i ∈ {0,...,n} do
8          Ponto inicial do trapézio
9          x_a ← x_l + i·Δx
10         Valor da função no ponto inicial
11         f_a ← f(x_a)
12         Calcula a área
13         area ← area + f_a Δx + 0.5(f(x_l + Δ) − f_a)Δx
14     return area
```

Algoritmo 43: Integral pelo método de Simpson de segunda ordem

```
1  Simpson1(f,x_l,x_u,n)
       Entrada: f função
       Entrada: x_l limite inferior
       Entrada: x_l limite superior
       Entrada: n número de divisões
2      Calcula o passo
3      Δx ← (x_u − x_l)/n
4      Inicializa a área
5      area ← 0
6      Inicializa os pontos
7      x_0 ← x_l
8      x_1 ← x_0 + Δx
9      x_2 ← x_1 + Δx
10     Laço pelas regiões
11     for i ∈ {1 : 2 : n} do
12         Calcula a área
13         area ← area + ((Δx)/3) · (f(x_0) + 4f(x_1) + f(x_2))
14         Atualiza os pontos
15         x_0 ← x_2
16         x_1 ← x_0 + Δx
17         x_2 ← x_1 + Δx
18     return area
```

17.4 Integração por Quadratura

A integração por quadratura tem como objetivo aproximar a integral de uma função por uma soma ponderada da função avaliada em pontos específicos. Tanto as posições dos pontos quanto os valores dos pesos são deduzidos de modo a minimizar o erro de integração. Existem diversas regras de quadratura diferentes, sendo que aqui iremos estudar a mais utilizada na área de mecânica computacional, conhecida como Quadratura de Gauss, ou Gauss-Legendre.

17.4.1 Quadratura de Gauss-Legendre

A quadratura de Gauss-Legendre integra exatamente um polinômio de ordem $2n-1$ com apenas n avaliações do integrando. Neste método aproximamos a integral de um polinômio $p(r)$ com $r \in [-1,1]$ por

$$\int_{-1}^{1} p(r)\,dr = \sum_{i=1}^{n} W_i p(r_i)$$

onde as coordenadas discretas r_i são chamados de **pontos de quadratura** e os escalares W_i são chamados de **pesos de quadratura**.

Vamos explorar a escolha dos pontos e pesos de quadratura utilizando um polinômio cúbico como exemplo.

Considere um polinômio cúbico, com a forma $p(r) = a_0 + a_1 r + a_2 r^2 + a_3 r^3$ com $r \in [-1,1]$. A integral deste polinômio é obtida analiticamente,

$$\int_{-1}^{1} p(r)\,dr = 2a_0 + \frac{2}{3}a_2$$

Algoritmo 44: Integral pelo método de Simpson de terceira ordem

```
1  Simpson2(f,x_l,x_u,n)
       Entrada: f função
       Entrada: x_l limite inferior
       Entrada: x_l limite superior
       Entrada: n número de divisões
2      Calcula o passo
3      Δx ← (x_u − x_l)/n
4      Inicializa a área
5      area ← 0
6      Inicializa os pontos
7      x_0 ← x_l
8      x_1 ← x_0 + Δx
9      x_2 ← x_1 + Δx
10     x_3 ← x_2 + Δx
11     Laço pelas regiões
12     for i ∈ {1 : 3 : n − 2} do
13         Calcula a área
14         area ← area + ((3Δx)/8) · (f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3))
15         Atualiza os pontos
16         x_0 ← x_2
17         x_1 ← x_0 + Δx
18         x_2 ← x_1 + Δx
19         x_3 ← x_2 + Δx
20     return area
```

e, com este resultado, podemos utilizar a forma aproximada proposta por Gauss, obtendo

$$2a_0 + \frac{2}{3}a_2 = \sum_{i=1}^{n} W_i p(r_i)$$

sendo que $n = 2$ implica na integração exata de um polinõmio de ordem $2 \cdot 2 - 1 = 3$. Assim,

$$2a_0 + \frac{2}{3}a_2 = W_1 p(r_1) + W_2 p(r_2)$$

ou, expandindo o polinômio,

$$2a_0 + \frac{2}{3}a_2 = W_1 \left(a_0 + a_1 r_1 + a_2 r_1^2 + a_3 r_1^3\right) + W_2 \left(a_0 + a_1 r_2 + a_2 r_2^2 + a_3 r_2^3\right)$$

de tal forma que podemos agrupar por coeficientes a_i em comum, obtendo

$$\begin{aligned} 2 &= W_1 + W_2 \\ 0 &= r_1 W_1 + r_2 W_2 \\ \frac{2}{3} &= r_1^2 W_1 + r_2^2 W_2 \\ 0 &= r_1^3 W_1 + r_2^3 W_2 \end{aligned}$$

cuja solução é $W_1 = W_2 = 1$, $r_1 = -\frac{1}{\sqrt{3}}$ e $r_2 = \frac{1}{\sqrt{3}}$. Assim, a integral exata de qualquer polinômio de ordem igual ou menor a 3 será obtida com duas avaliações da função, na forma

$$\int_{-1}^{1} p(r)\, dr = 1 \cdot p\left(-\frac{1}{\sqrt{3}}\right) + 1 \cdot p\left(\frac{1}{\sqrt{3}}\right).$$

■ **Exemplo 17.4** Seja a função $p(r) = 2 - 3r + 2^3$ em $x \in [-1, 1]$. Utilizando quadratura com 2 pontos, teremos

$$\int_{-1}^{1} r(r)\, dr = p\left(-\frac{1}{\sqrt{3}}\right) \cdot 1 + p\left(\frac{1}{\sqrt{3}}\right) \cdot 1 = \left(\sqrt{3} + 10\right) + \left(10 - \sqrt{3}\right) = 20$$

que é o valor exato. ■

■ **Exemplo 17.5** Seja a função $f(x) = 3x^2 - 6x^3$ com $x \in [0, 1]$.

Como o intervalo de interesse não está definido no intervalo $[-1, 1]$, precisamos primeiro realizar uma mudança de variável, de tal forma que o intervalo seja corrigido para os limites apropriados. Assim, podemos facilmente verificar que se o intervalo a ser integrado for $[x_L, x_U]$, então

$$r = -1 + 2\frac{x - x_L}{x_U - x_L}$$

e

$$x = \frac{r(x_U - x_L) + (x_U + x_L)}{2}$$

que neste caso permite obter

$$x = \frac{r + 1}{2},$$

de tal forma que o polinômio pode ser escrito como

$$p(r) = 3\left(\frac{r+1}{2}\right)^2 - 6\left(\frac{r+1}{2}\right)^3.$$

É importante salientar que ao mudarmos a variável, estamos mudando também o diferencial da integral. Assim, para convertermos de dr para dx,

$$dr \cdot \frac{dx}{dr} = dx$$

onde

$$\frac{dx}{dr} = \frac{x_U - x_L}{2}$$

é a relação entre os comprimentos do domínio original e do domínio normalizado. Este fator deve ser SEMPRE considerado quando realizamos a mudança de variável. Neste exemplo, o fator de correção é $\frac{1}{2}$.

Desta fora, finalmente podemos obter a integral

$$\frac{1}{2}\left\{\left[3\left(\frac{-\frac{1}{\sqrt{3}}+1}{2}\right)^2 - 6\left(\frac{-\frac{1}{\sqrt{3}}+1}{2}\right)^3\right] + \left[3\left(\frac{\frac{1}{\sqrt{3}}+1}{2}\right)^2 - 6\left(\frac{\frac{1}{\sqrt{3}}+1}{2}\right)^3\right]\right\} = -0.5$$

■

A Tabela 17.1 apresenta pontos e pesos de quadratura para algumas situações, com pontos e pesos arredondados para a décima casa decimal.

n	r_j	W_j
1	0	2
2	±0.5773502692	1
3	±0.7745966692 0	0.5555555556 0.8888888889
4	±0.8611363116 ±0.3399810436	0.3478548451 0.6521451549
5	±0.9061798459 0 ±0.5384693101	0.2369268851 0.5688888889 0.4786286705
6	±0.9324695142 ±0.6612093865 ±0.2386191861	0.1713244924 0.3607615730 0.4679139346

Tabela 17.1: Algumas regras de quadratura de Gauss-Legendre

O Julia tem o pacote *FastGaussQuadrature*, que permite obter os pontos e pesos de forma muito ágil

```
julia> using FastGaussQuadrature

julia> nodes, weights = gausslegendre( 6 );
```

```
    julia> nodes
6-element Vector{Float64}:
 -0.932469514203152
 -0.6612093864662645
 -0.2386191860831969
  0.2386191860831969
  0.6612093864662645
  0.932469514203152

julia> weights
6-element Vector{Float64}:
 0.17132449237917025
 0.3607615730481385
 0.46791393457269126
 0.46791393457269126
 0.3607615730481385
 0.17132449237917025

julia> f(r) = 1 + 5*r^3 - 2*r^4
f (generic function with 1 method)

julia> sum(f.(nodes).*weights)
1.2000000000000006
```

17.4.2 Integrais duplas e triplas

Os conceitos vistos em uma dimensão podem ser estendidos para integrais em múltiplas dimensões. Desta forma, para um polinômio definido em duas variáveis r e s, podemos calcular a integral com

$$\int_{-1}^{1}\int_{-1}^{1} p(r,s)drds = \sum_{i=1}^{npg}\sum_{j=1}^{npg} p(r_i,s_j)W_iW_j, \tag{17.1}$$

e, para um polinômio definido em três variáveis r, s e t, , podemos calcular a integral com

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} p(r,s,t)drdsdt = \sum_{i=1}^{npg}\sum_{j=1}^{npg}\sum_{j=1}^{npg} p(r_i,s_j,t_k)W_iW_jW_k. \tag{17.2}$$

Uma questão que deve ser observada é no cálculo da correção entre a área do domínio original e do domínio normalizado. De forma análoga ao que foi feito no caso 1D, quando realizamos uma mudança de variável entre um domínio $[x_l,x_u]\times[y_l,y_u]$ para um problema 2D $[-1,1]\times[-1,1]$, devemos multiplicar as integrais obtidas pela quadratura pela correção $|J|$, que é a área de domínio original pelo domínio normalizado (2×2). Este cálculo é trivial se o domínio original é um retângulo (sem distorções), mas deve ser calculado com cuidado no caso de domínios distorcidos.

■ **Exemplo 17.6** Seja o polinômio $p(x,y) = 20x^2 + 10xy - 5y^3$ no domínio $[0,5]\times[0,6]$. A integral deste polinômio é -850 e, se fizermos as mudanças de variáveis

$$r = -1 + 2\frac{x}{5}$$
$$s = -1 + 2\frac{y}{6}$$

tal que o polinômio nas coordenadas r e s pode ser escrito como

$$p(r,s) = -135s^3 - 405s^2 + (75r - 330)s + 125r^2 + 325r + 65.$$

A integral por quadratura, ainda sem considerarmos a correção de área, é obtida por

$$\begin{aligned}\int_{-1}^{1}\int_{-1}^{1} p(r,s)\,dr\,ds = &\left(-135s_1^3-405s_1^2+(75r_1-330)s_1+125r_1^2+325r_1+65\right)W_1W_1+\\ &\left(-135s_2^3-405s_2^2+(75r_1-330)s_2+125r_1^2+325r_1+65\right)W_1W_2+\\ &\left(-135s_1^3-405s_1^2+(75r_2-330)s_1+125r_2^2+325r_2+65\right)W_2W_1+\\ &\left(-135s_2^3-405s_2^2+(75r_2-330)s_2+125r_2^2+325r_2+65\right)W_2W_2,\end{aligned}$$

com

$$\begin{aligned}r_1, s_1 &= \left(\frac{1}{\sqrt{3}}\right)\\ r_2, s_2 &= -\left(\frac{1}{\sqrt{3}}\right)\\ W_1 &= W_2 = 1.\end{aligned}$$

Esta integral dá $-\frac{340}{3}$ e, se multiplicarmos este valor pela proporção entre as áreas, obtemos o resultado correto

$$-\frac{340}{3}\frac{6\cdot 5}{2\cdot 2} = -850. \tag{17.3}$$

■

17.4.3 Relação entre os pontos de quadratura e os polinômios de Legendre.

Os polinômios de Legendre são obtidos como solução para a EDO

$$\frac{d}{dx}\left[(1-x^2)\frac{dP_n(x)}{dx}\right]+n(n+1)P_n(x)=0$$

para $n\in\mathbb{N}$ e $x\in[-1,1]$ e tem a forma

$$P_n(x)=\frac{1}{2^n n!}\frac{d^n}{dx^n}\left[(x^2-1)^n.\right]$$

Estes polinômios são ortogonais

$$\int_{-1}^{1} P_n(x)P_m(x)dx = \frac{2}{2n+1}\delta_{mn}$$

e tem como raízes as coordenadas dos pontos de quadratura de ordem n.

Os pesos, por sua vez, são calculados com

$$W_j = \frac{2}{(1-r_j^2)\left(\frac{dP_n(r)}{dr}\Big|_{r_j}\right)^2}.$$

Exercício 17.1 Deduza as quadraturas para $n=1$ (um ponto de Gauss) e para $n=4$. Quais são os graus dos polinômios que podem ser corretamente integrados por estas quadraturas ? ■

Exercício 17.2 Integre a função $f(x) = e^{3x}$ para $x = [2,5]$ utilizando todos os métodos de interpolação apresentados e compare com a solução analítica. ■

Exercício 17.3 Obtenha a integral da função do exercício 2 utilizando Quadratura de Gauss para $n = 2$, $n = 3$ e $n = 4$. Explique o que está acontecendo. ■

18. Decomposição em Valor Singular

Dada uma matriz $\mathbf{A}_{m\times n}$, chamamos σ de valor singular de $\mathbf{A}$ se e somente se existem dois vetores $\mathbf{u}_{m\times 1}$ e $\mathbf{v}_{n\times 1}$ que satisfazem

$$\mathbf{A}\mathbf{v} = \sigma\mathbf{u}$$

e

$$\mathbf{A}^T\mathbf{u} = \sigma\mathbf{v}.$$

Os vetores $\mathbf{u}$ e $\mathbf{v}$ são chamados, respectivamente, de vetor singular a esquerda e de vetor singular a direita. Uma matriz $\mathbf{A}$ tem ao mínimo 1 e no máximo $\min(m,n)$ valores singulares distintos. É interessante notar que se considerarmos todos os valores e vetores singulares ao mesmo tempo obteremos

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{18.1}$$

onde $\mathbf{U}_{m\times m}$ armazena em suas colunas os m vetores singulares a esquerda, $\mathbf{S}_{m\times n}$ contém somente termos não negativos em suas posições diagonais (demais termos são nulos) e $\mathbf{V}_{n\times n}$ contém os n vetores singulares a direita. A Eq. (18.1) é chamada de decomposição em valor singular da matriz $\mathbf{A}$, ou, em inglês, *Singular Value Decomposition* ou SVD. É importante salientar que as matrizes $\mathbf{U}$ e $\mathbf{V}$ tem colunas com norma unitária e não são únicas. Desta forma, as colunas de $\mathbf{U}$ e $\mathbf{V}$ formam bases ortonormais para $\mathbf{A}$ tal que $\mathbf{U}\mathbf{U}^T = \mathbf{I}$ e $\mathbf{V}\mathbf{V}^T = \mathbf{I}$. Com isto

$$\mathbf{A}^{-1} = \mathbf{V}\mathbf{S}^{-1}\mathbf{U}^T$$

o que torna a inversão trivial. Esta inversão é conhecida como pseudo-inversa da matriz, sendo uma generalização do conceito de inversa para matrizes não quadradas.

Desta forma, se tivermos um sistema de equações lineares com um número diferente de equações e incógnitas, na forma

$$\mathbf{A}_{m\times n}\mathbf{x}_{n\times 1} = \mathbf{b}_{m\times 1}$$

podemos escrever a "solução" utilizando a inversa da decomposição em valor singular, tal que

$$\mathbf{x} = \mathbf{V}\mathbf{S}^{-1}\mathbf{U}^T\mathbf{b}$$

e a solução será a de menor erro de acordo com a norma 2, ou seja,

$$\mathbf{x} = \arg\min \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 .$$

Esta solução pode ser utilizada quando temos um problema mal condicionado (mesmo no caso de matrizes quadradas).

O procedimento para obtermos a decomposição singular de uma matriz $\mathbf{A}$ é:

- Calcular $\mathbf{A}\mathbf{A}^T$;
- Determinar os autovalores e autovetores de $\mathbf{A}\mathbf{A}^T$: $\mathbf{\Phi}_{AA^T}$;
- Aplicar Gramm-Schmidt nas colunas de $\mathbf{\Phi}_{AA^T}$, gerando $\mathbf{U}$;
- Calcular $\mathbf{A}^T\mathbf{A}$;
- Determinar os autovalores e autovetores de $\mathbf{A}^T\mathbf{A}$: $\mathbf{\Phi}_{A^TA}$;
- Aplicar Gramm-Schmidt nas colunas de $\mathbf{\Phi}_{A^TA}$, gerando $\mathbf{V}$;
- Gerar $\mathbf{S}$, tal que $s_{ii} = \sqrt{\lambda_i}$, em ordem decrescente ($\lambda_i \neq 0$);

■ **Exemplo 18.1** Seja a matriz

$$\mathbf{A} = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix}.$$

Gerando a forma a esquerda

$$\mathbf{A}\mathbf{A}^T = \begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix},$$

obtemos $\lambda_1 = 12$, $\lambda_2 = 10$ e

$$\mathbf{\Phi}_{AA^T} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

que uma vez ortogonalizada por Gramm-Schmidt resulta em

$$\mathbf{U} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}.$$

Gerando a forma a direita

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 10 & 0 & 2 \\ 0 & 10 & 4 \\ 2 & 4 & 2 \end{bmatrix},$$

obtemos $\lambda_3 = 12$, $\lambda_4 = 10$, $\lambda_5 = 0$ e

$$\mathbf{V} = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{30}} \\ \frac{2}{\sqrt{6}} & \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{30}} \\ \frac{1}{\sqrt{6}} & 0 & \frac{-5}{\sqrt{30}} \end{bmatrix}.$$

e, por fim,

$$\mathbf{S} = \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{bmatrix},$$

tal que

$$\begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{30}} \\ \frac{2}{\sqrt{6}} & \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{30}} \\ \frac{1}{\sqrt{6}} & 0 & \frac{-5}{\sqrt{30}} \end{bmatrix}^T.$$

Observem que os autovalores foram colocados em ordem decrescente.

■

18.1 Relação entre SVD, *Rank* e Espaço Nulo de um Operador

Algumas informações importantes podem ser obtidas a partir das matrizes **U**, **S** e **V**.

- A dimensão do espaço nulo do operador, *nullity*, pode ser obtida pelo número de colunas de **V** correspondentes as posições onde os valores singulares são nulos, ie, $S_{ii} = 0$;
- A dimensão do *Rank* do operador, pode ser obtida pelo número de colunas de **U** correspondentes as posições não nulas dos valores singulares, ie, $S_{ii} \neq 0$;

Assim, baseado no *Rank* do operador, obtido por SVD, é possível avaliar o *Rank* numérico efetivo de um operador, pois erros numéricos podem levar a um *Rank* diferente do analítico. No exemplo anterior, observamos que o espaço nulo tem dimensão 1 e que o *Rank* é 2.

■ **Exemplo 18.2** A matriz

$$\mathbf{A} = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

tem como matriz de valores singulares

$$\sigma = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

de tal forma que seu *Rank* é 1 e a dimensão de seu espaço nulo é 2. ■

Exercício 18.1 Solucione um problema mal condicionado dos capítulos anteriores utilizando o SVD. ■

18.2 Compressão de Imagens Utilizando o SVD - *Low-Rank Matrix Approximation*

Uma imagem é armazenada no computador como uma matriz retangular de dimensão $n \times m$. Assim, considerando que cada posição é ocupada por um *byte*, são necessários $m * n$ bytes de espaço para armazenar esta informação. Se aplicarmos o SVD em uma matriz que contém as informações de uma imagem, teremos n valores singulares (considerando que $n \leqslant m$) não nulos, tal que a imagem pode ser reconstruída por uma combinação linear da forma

$$\mathbf{A} = \sum_{i=1}^{n} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

onde **A** é a matriz que contém a imagem reconstruída. Como ordenamos os valores principais em ordem decrescente, temos que os primeiros termos do somatório são formados pelos valores singulares mais significativos da imagem. Desta forma, apenas alguns valores singulares são necessários para obter uma boa aproximação da imagem original. Neste caso, são necessários apenas $N < n \leqslant m$ valores principais, com seus respectivos vetores a direita e a esquerda, tal que o armazenamento total será de $N + N \cdot n + N \cdot m$ *bytes*.

■ **Exemplo 18.3** Considere a imagem[1].

Figura 18.1: Imagem em tons de cinza com 512×512 pixels.

Os valores singulares da matriz **A** que representa a imagem são ilustrados na Fig. 18.3, de onde podemos ver que os primeiros valores singulares são muito mais significativos do que os demais. De fato, os 20 primeiros valores, somados, correspondem a 66% da soma de todos os 512 valores singulares.

Se reconstruirmos a imagem utilizando a combinação linear dos 20 primeiros valores e vetores singulares, obteremos a imagem da Fig. 18.3. Pode-se notar que, mesmo com um número pequeno de termos na combinação linear, podemos reconhecer claramente as características (informações) da imagem original.

Por fim, se utilizarmos os 60 primeiros valores singulares (contendo 82% das informações da imagem original), obteremos a imagem da Fig. 18.3.

O código em Julia utilizado para os cálculos e operações deste exemplo estão listados no Alg. 18.1.

[1] https://raw.githubusercontent.com/JuliaImages/TestImages.jl/images/images/jetplane.tiff

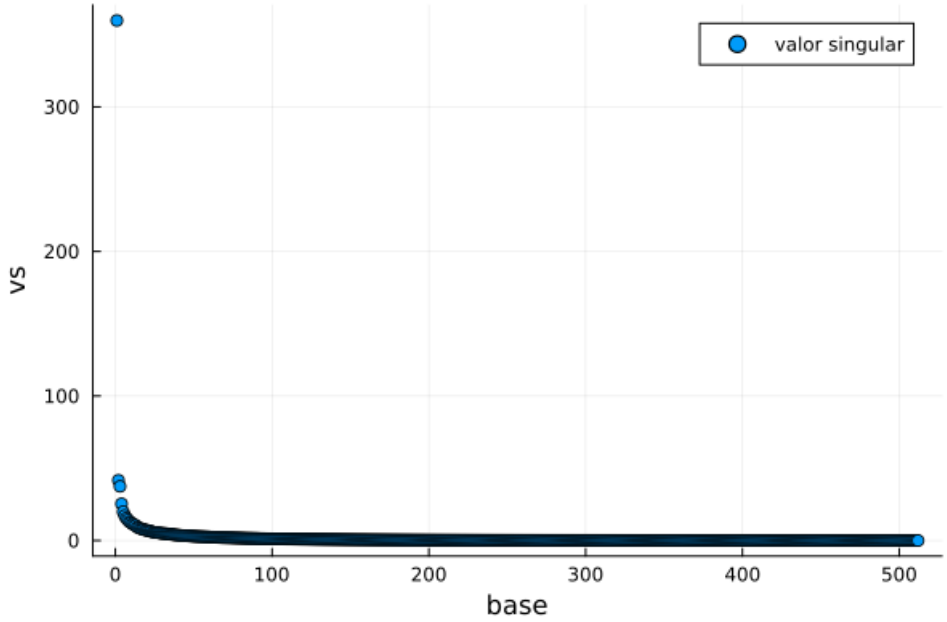


Figura 18.2: Valores singulares correspondentes a matriz **A** que representa a imagem da Fig. 18.3.

Figura 18.3: Imagem reconstruída com os 20 primeiros valores singulares.

Listing 18.1: Código em Julia utilizado no exemplo de compressão de imagem utilizando SVD.

Figura 18.4: Imagem reconstruída com os 60 primeiros valores singulares.

```
using LinearAlgebra, Plots
using TestImages, Images, ImageIO

# Carrega a imagem
img = testimage("jetplane")

# Converte a matriz para números e escala de cinza
A = float64.(Gray.(img))

# Calcula a decomp em valor singular da matriz
U,S,V = svd(A);

# Dimen. da imagem
dim = size(A,1)

# Vamos ver os valores singulares
scatter(S,label="valor singular",xlabel="base",ylabel="vs")

# Vamos montar a imagem original usando os N maiores termos de base
N = 60;

# Evita números inválidos (maiores do que a imagem e menores do que 1)
N = max(1,min(N,dim))

# Calcula a porcentagem dos primeiros N valores principais
@show sum(S[1:N])/sum(S)

# Inicializa a imagem reconstruída
B = zeros(dim,dim);

# Loop para reconstruir a imagem
for i=1:N
```

```
    B .= B .+ S[i]*U[:,i]*V[:,i]'
end

# Mostra a imagem reconstruída
C = convert(typeof(A),B)

# Salva a figura reconstruída
save("reconstruida.png",map(clamp01nan,C))
```

■

19. Transformada Discreta de Fourier - DFT

Na primeira parte da matéria - Fundamentos de Álgebra - vimos que a Série de Fourier de uma função contínua periódica de período τ[1] é descrita pela projeção ortogonal da função na base

$$B = \left\{\frac{1}{2}\right\} \bigcup \{\cos(k\omega t)\} \bigcup \{\sin(k\omega t)\}, k \in \mathbb{N}$$

onde

$$\tau = \frac{2\pi}{\omega}.$$

Se a função for definida no intervalo $t \in [0, T_f]\,s$, então a série de Fourier é obtida com

$$f(t) \cong a_0 \frac{1}{2} + \sum_{k=1}^{\infty} \{a_k \cos(k\omega t) + b_k \sin(k\omega t)\} \tag{19.1}$$

sendo que os coeficientes a_0, a_k e b_k são obtidos por meio das projeções ortogonais

$$a_0 = \frac{\langle f(t), 1/2\rangle}{\langle 1/2, 1/2\rangle} = \frac{2}{\tau}\int_0^{\tau} f(t)\,dt,$$

$$a_k = \frac{\langle f(t), \cos(k\omega t)\rangle}{\langle \cos(k\omega t), \cos(k\omega t)\rangle} = \frac{2}{\tau}\int_0^{\tau} f(t)\cos(k\omega t)\,dt$$

e

$$b_k = \frac{\langle f(t), \sin(k\omega t)\rangle}{\langle \sin(k\omega t), \sin(k\omega t)\rangle} = \frac{2}{\tau}\int_0^{\tau} f(t)\sin(k\omega t)\,dt.$$

Desta forma, a equação 19.1 permite escrever uma função no tempo, $f(t)$, em termos da combinação linear de funções trigonométricas com diferentes frequências $k\omega$, sendo que os

[1] isto é, uma função com a propriedade $f(t) = f(t+\tau)$

coeficientes a_k e b_k permitem associar uma magnitude (importância) para cada uma destas componentes de frequência. Por isto, tal operação é dita um mapeamento do domínio da frequência para o domínio do tempo.

A equação 19.1 pode ser escrita em formas alternativas, porém diferentes. Uma forma mais compacta de obter esta mesma descrição é utilizando a identidade de Euler,

$$e^{ik\omega t} = \cos(k\omega t) + i\sin(k\omega t)$$

tal que a série de Fourier pode ser escrita como

$$f(t) \cong a_0\frac{1}{2} + \sum_{k=1}^{\infty} c_k e^{ik\omega t},$$

com coeficientes complexos c_k. Como a exponencial de zero é um, podemos incluir o termo constante no somatório, bastando para isto modificar o valor inicial de k, tal que

$$f(t) \cong \sum_{k=0}^{K} F_k e^{i\omega_k t}, \tag{19.2}$$

onde F_k são as amplitudes complexas e, para simplificarmos a notação, definimos ω_k como sendo o k-ésimo múltiplo de ω. Da mesma forma, devido a impossibilidade de trabalharmos com infinitos valores no somatório, estimamos um valor limite K para o número de termos na aproximação.

É interessante notar que a Equação 19.1 também pode ser escrita em outra forma alternativa, pois

$$a_k\cos(k\omega t) + b_k\sin(k\omega t) = R_k\cos(k\omega t - \phi_k)$$

onde $R_k = \sqrt{a_k^2 + b_k^2}$ é o módulo de um cosseno defasado em um ângulo $\tan(\phi_k) = b_k/a_k$. Assim, considerando este resultado, podemos identificar que a notação da Eq. 19.2 encapsula a informação de amplitude e fase de vários cossenos de frequência ω_k.

As equações obtidas acima dizem respeito a uma função que varia continuamente no tempo t. No entanto, quando adquirimos um sinal, obtemos um conjunto discreto de valores ao longo do tempo. Desta forma, se o tempo for discretizado em N pacotes (ou valores discretos), tal que

$$\Delta t = \frac{T_f}{N},$$

onde T_F é o tempo final, podemos escrever que o tempo em uma posição discreta n é dado por

$$t_n = n\Delta t,$$

tal que a equação 19.2 se torna

$$f_n \cong \sum_{k=0}^{K} F_k e^{i\omega_k n\frac{T_f}{N}} \tag{19.3}$$

para $n \in [0, N-1]$. Se ainda considerarmos que cada frequência ω_k pode ser descrita por

$$\omega_k = \frac{2\pi}{\tau_k}$$

e que os períodos associados a cada ω_k são pacotes discretos do tempo total, na forma

$$\tau_k = \frac{T_F}{k}$$

então podemos escrever a equação 19.3 como

$$f_n \cong \sum_{k=0}^{K} F_k e^{2\pi n \frac{k}{N} i}. \tag{19.4}$$

A operação inversa, chamada de Discrete Fourier Transform - DFT - permite obter os F_k a partir de um sinal no tempo. Para isto, simplesmente invertemos a relação da Eq. 19.4, resultando em

$$F_k = \sum_{n=0}^{N-1} f_n e^{-2\pi n \frac{k}{N} i} \tag{19.5}$$

com $k \in [0,K]$ e $n \in [0,N-1]$. Nessa equação, podemos ver que cada F_k está associado a uma componente de frequência $\omega_k = \frac{2\pi}{T_f} k$ $[rad/s]$, tal que esta operação é um mapeamento do domínio do tempo para o domínio da frequência. O procedimento está descrito no Alg. 45.

Algoritmo 45: Transformada Discreta de Fourier

```
1 DFT(f,N)
    Entrada: f vetor com os valores do sinal nos tempos discretos
    Entrada: N número de pacotes no tempo
2   Aloca o vetor com as amplitudes complexas
3   F ← 0_N
4   Laço por cada freq/tempo
5   for k ∈ {0,...,N−1} do
6       for n ∈ {0,...,N−1} do
7           F(k+1) ← F(k+1) + f(n+1) exp(−2iπnk/N)
8   return F
```

Se o sinal f_n for real, então F_k é simétrico conjugado, e necessitamos somente dos primeiros $K/2$ pontos para caracterizar o sinal, pois a outra parte do vetor é simétrica (espelhada).

É importante salientar que, mesmo no caso de f_n reais, temos que os F_k são complexos. Isto é interessante, pois as partes reais e complexas de F_k contém as informações sobre amplitude e fase do sinal f_n. Assim, se

$$F_k = R_k + I_k i$$

temos que

$$M_k = \sqrt{R_k^2 + I_k^2} \tag{19.6}$$

é o módulo do sinal na frequência ω_k e

$$\tan(\phi_k) = \frac{I_k}{R_k} \tag{19.7}$$

onde ϕ_k é uma fase.

■ **Exemplo 19.1** Seja um sinal com a forma

$$f(t) = 10\cos(2\pi 10t) + 5\sin(2\pi 15t),$$

que é a sobreposição de um cosseno de amplitude 10 em 10Hz e um seno de amplitude 5 em 15Hz. Se gerarmos o sinal em um intervalo de 2s iremos obter o gráfico ilustrado na figura 19.1.

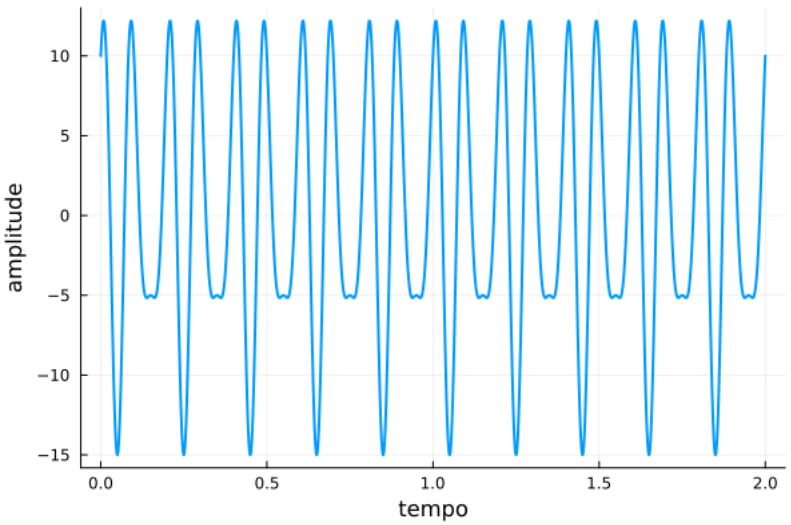


Figura 19.1: Sinal de referência para o exemplo 1.

Se o sinal for amostrado com $N = 128$ pacotes no tempo, então teremos uma resolução de $\Delta t = 2/128 = 0.0155625$s e uma taxa de amostragem de 64Hz (64 leituras por segundo). Se sobrepusermos esses pontos sobre o gráfico (contínuo) da função de referência, obtemos o gráfico da Fig. 19.1

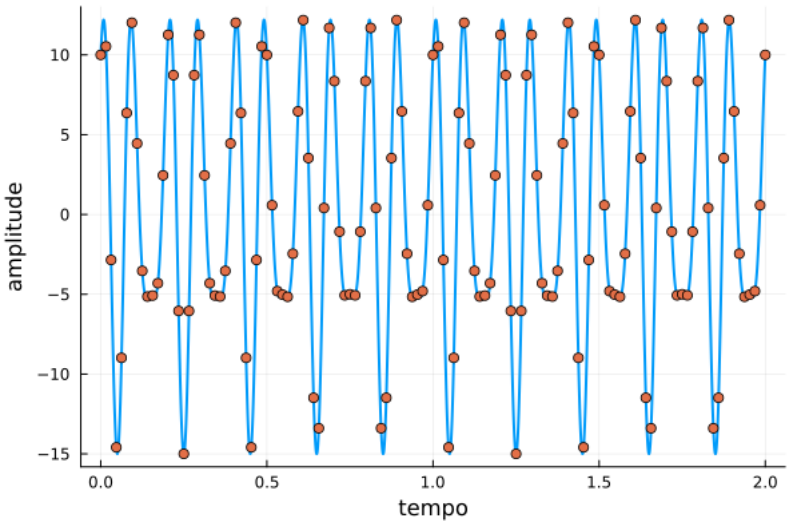


Figura 19.2: Sobreposição dos valores discretos ($N = 128$) e da função de referência.

Neste caso, se utilizarmos $K = N - 1 = 127$, então obteremos a DFT do sinal com o algoritmo Alg. 45. O gráfico das figura 19.1 ilustra o valor real de **F** para cada valor de

k, de onde podemos verificar que $\mathbf{F}$ é realmente simétrico. Da mesma forma, a Fig. 19.1 mostra que os resultados são simétrico-conjugados (reflexão e inversão do valor da parte complexa).

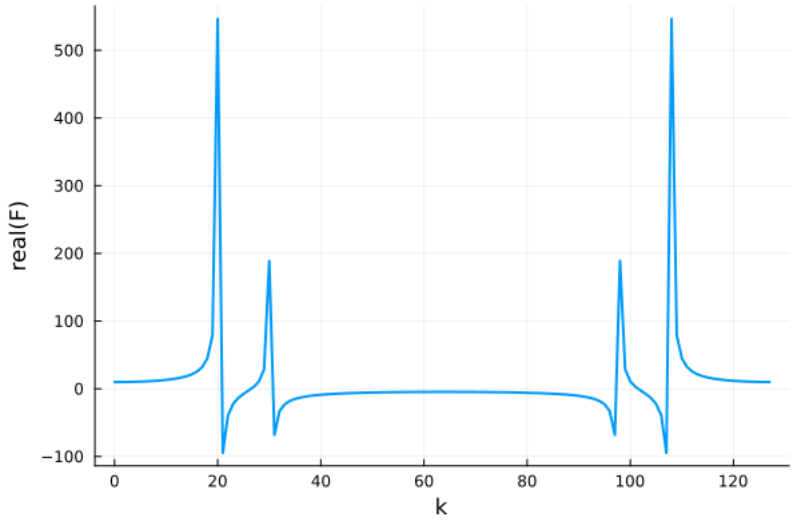


Figura 19.3: Parte real da DFT do sinal amostrado com $N = 128$.

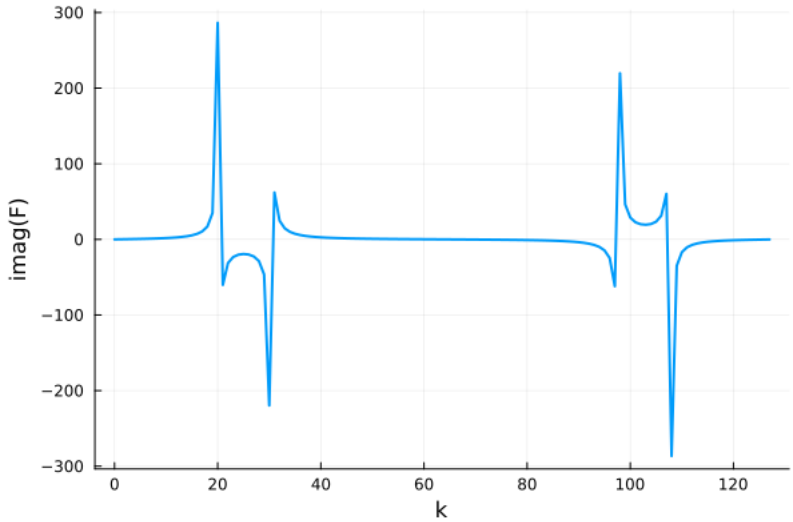


Figura 19.4: Parte imaginária da DFT do sinal amostrado com $N = 128$.

■

Como podemos ver nos resultados do exemplo, a simetria faz com que sejam necessárias apenas os primeiros $K/2$ valores da DFT, sendo que isto está associado ao fato de **não termos utilizado uma extensão periódica para avaliar o sinal em um intervalo consistente de $n \in [-N/2, N/2]$, conforme apresentado no capítulo de séries de Fourier**.

Outro resultado interessante que vemos nos gráficos é o fato das amplitudes serem muito maiores do que as observadas no sinal original (domínio do tempo). Isto se deve ao fato de estarmos realizando um somatório (Loop) de dimensão N para cada K, somado ao

fato de termos a questão da simetria. Desta forma, observamos que as amplitudes devem ser escalonadas (divididas) por um fator de $N/2$.

Assim, utilizando somente metade do gráfico e escalonando (dividindo) os resultados por $N/2$ obtemos os gráficos ilustrados nas figuras 8 e 8.

O gráfico do valor absoluto, Fig. 8 ilustra amplitudes muito próximas do sinal que foi utilizado, pois temos ≈ 10 e ≈ 5.

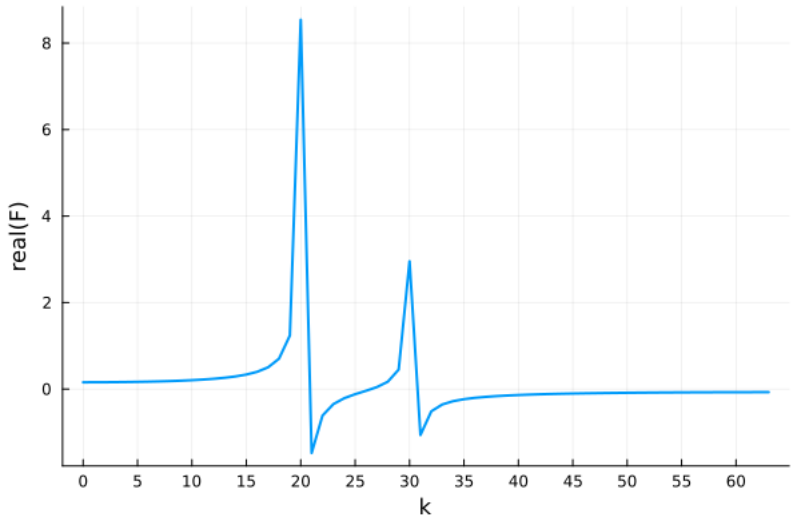


Figura 19.5: Parte real da DFT do sinal amostrado com $N = 128$, somente as primeiras $N/2$ posições de $\mathbf{F}$ divididas por $N/2$.

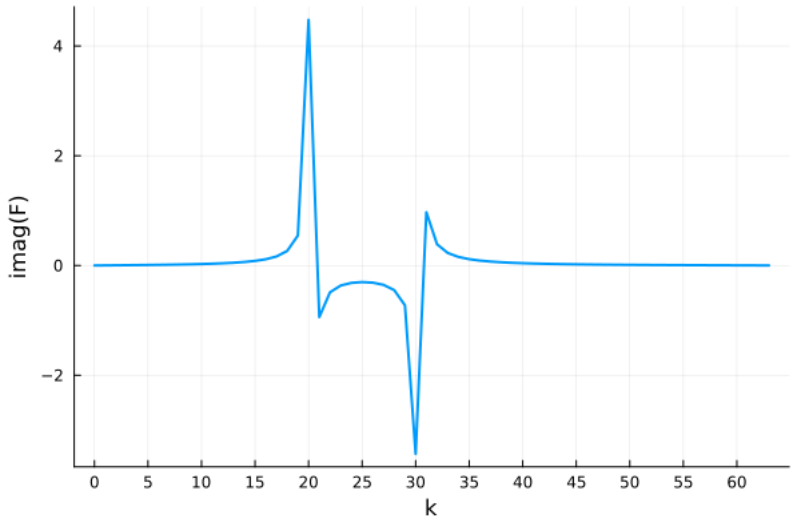


Figura 19.6: Parte imaginária da DFT do sinal amostrado com $N = 128$, somente as primeiras $N/2$ posições de $\mathbf{F}$ divididas por $N/2$.

Por fim, podemos converter os valores do eixo da abcissa (k) para valores de frequência. Para isso, lembramos que a variável k é um contador de frequências angulares discretas $\omega_k = \frac{2\pi}{\tau_k}$ onde $\tau_k = \frac{T_f}{k}$. Com isso, para cada k temos uma frequência angular

$$\omega_k = k\frac{2\pi}{T_f}$$

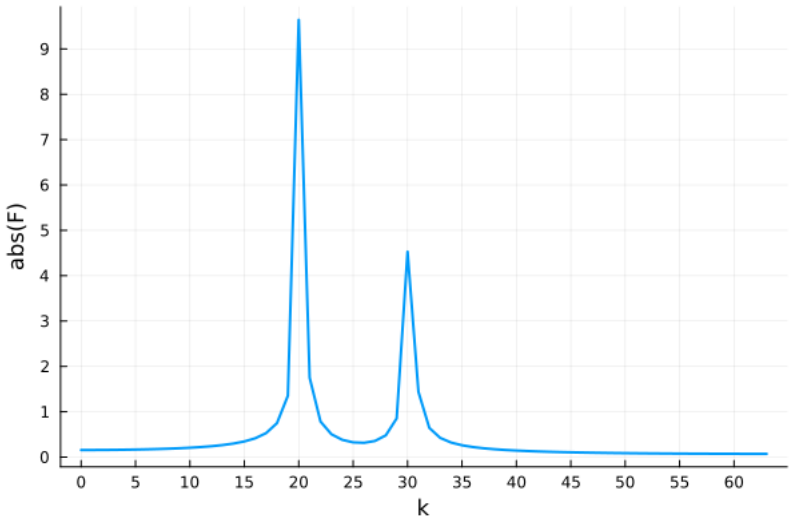


Figura 19.7: Valor absoluto da DFT do sinal amostrado com $N = 128$, somente as primeiras $N/2$ posições de $\mathbf{F}$ divididas por $N/2$.

ou, em Hz

$$f_k = \frac{k}{T_f}, \quad k = 0...N/2 - 1.$$

O resultado está ilustrado na Fig. 8, de onde podemos identificar as frequências do seno e do cosseno originais do sinal.

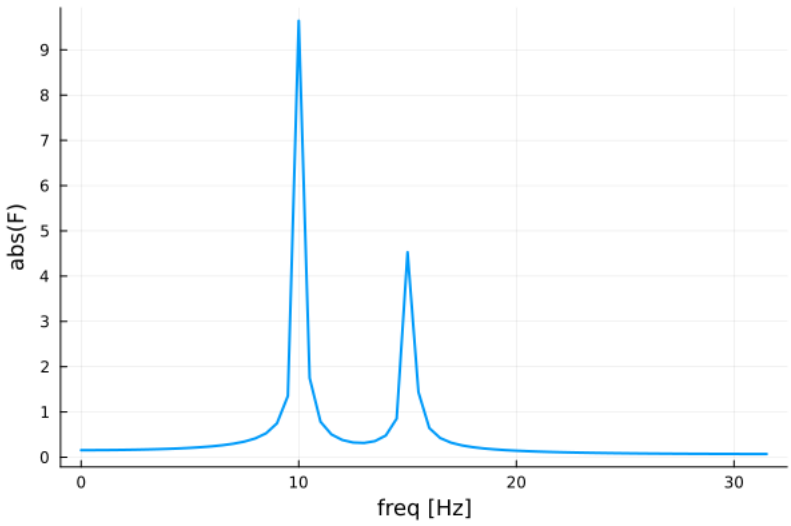


Figura 19.8: Valor absoluto da DFT do sinal amostrado com $N = 128$, somente as primeiras $N/2$ posições de $\mathbf{F}$ divididas por $N/2$. Eixo das abcissas corrigido para Hz.

O gráfico da parte complexa, juntamente com o gráfico da fase, permite avaliar uma característica interessante da DFT quando aplicada à análise modal. Quando temos um padrão "para cima" e um "para baixo", podemos interpretar a parte para cima como sendo uma componente de um cosseno e a para baixo como a componente de um seno. Assim, cada pico terá a parte complexa com sinais diferentes.

19.1 Influência da taxa de amostragem - *aliasing*

O sinal do exemplo anterior foi gerado com um cosseno de frequência 10Hz e um seno de frequência 15Hz. Como o número de pontos utilizados para amostrar o tempo foi de 128 pacotes em $2s$, utilizamos uma taxa de amostragem de 64 pacotes por segundo. Esta taxa é 4.26 maior do que a maior frequência do sinal.

Para ilustrarmos o quanto a taxa de aquisição do sinal é importante, vamos utilizar uma taxa de 16 pacotes por segundo, tal que $N = 32$. Neste caso, o sinal amostrado tem a forma ilustrada no gráfico da figura 19.1, que quando comparado com o (mesmo) sinal obtido com $N = 128$, figura 19.1, ilustra a falta de informações sobre o fenômeno.

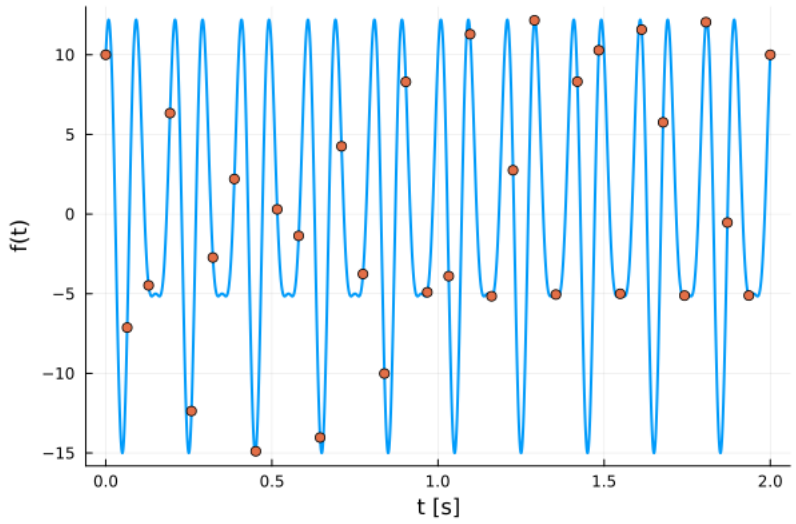


Figura 19.9: Efeito de uma baixa frequência de amostragem, ou *aliasing*.

Se aplicarmos a DFT neste sinal, iremos obter, por exemplo, o gráfico de magnitudes ilustrado na figura 19.1, que mostra claramente que a faixa de frequências que a DFT consegue calcular é menor do que as frequências existentes no sinal. O que está sendo calculado é uma resposta fictícia, causada pela falta de informação sobre o sinal verdadeiro.

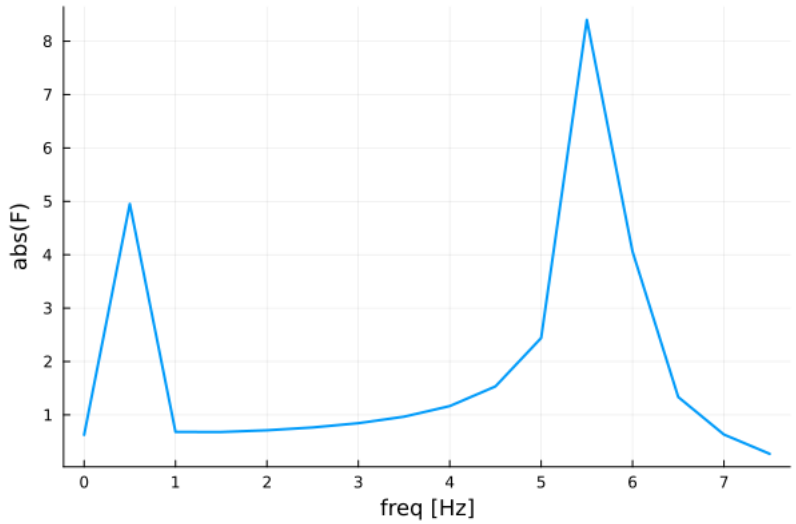


Figura 19.10: Efeito de uma baixa frequência de amostragem, ou *aliasing*.

Desta forma, podemos definir a famosa Regra de Nyquist, que diz que **a taxa de amostragem do sinal deve ser no mínimo duas vezes a maior frequência que ser quer obter**.

Novamente, podemos utilizar esta regra e o conhecimento sobre o nosso exemplo para definir que como a maior frequência é de $15Hz$ e estamos amostrando um sinal de $2s$, então $N \geqslant 15 \cdot 2 \cdot 2$ deve ser suficiente. Utilizando a potência de 2 mais próxima, $N = 64$, obtemos o sinal discreto ilustrado na figura 19.1.

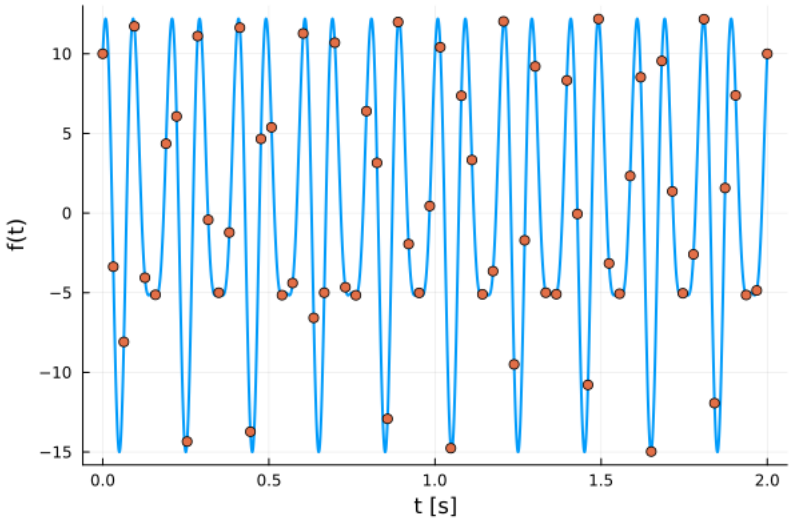


Figura 19.11: Amostragem mínima para capturar 15 Hz.

A figura mostra que já é possível identificar as características de mais alta frequência do sinal. No entanto, como a faixa de frequência que estamos aptos a descrever é de 16Hz, pois $64\ pacotes/2\ segundos = 32$, e como utilizamos somente metade do espectro, $f_{max} = 32/2 = 16$Hz, estamos bem no limite da resolução da DFT, Fig. 19.1.

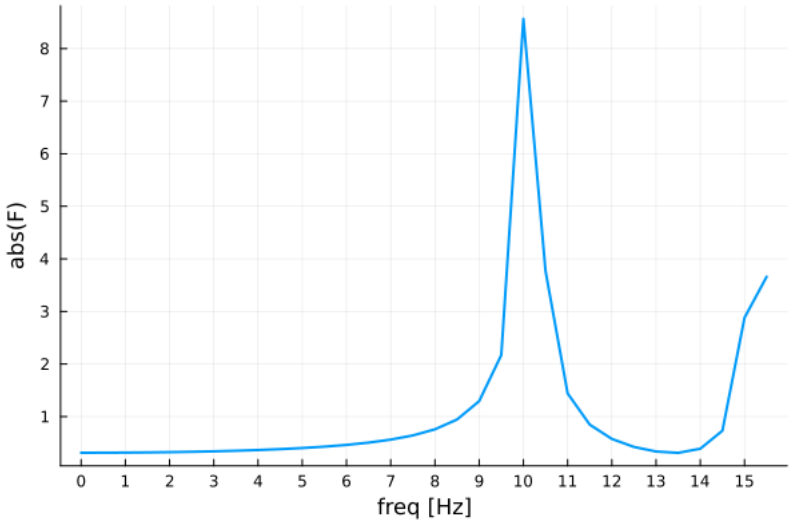


Figura 19.12: Amostragem mínima para capturar 15 Hz.

■ **Exemplo 19.2** Um exemplo de como o *aliasing* pode ser problemático é a amostragem de

um seno puro

$$f(t) = \sin(2\pi 10t)$$

a uma taxa de amostragem de 5Hz. Se a amostra for de 10s, isso corresponde a 5 leituras por segundo, totalizando 50 leituras. Pode-se ver claramente que essa taxa de aquisição faz com que o sinal adquirido seja um seno com frequência de 5 Hz.

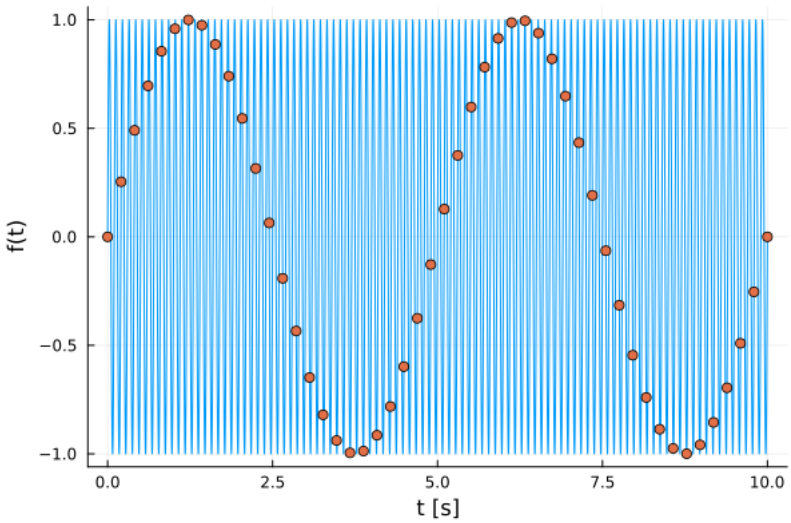


Figura 19.13: Exemplo de *aliasing* clássico.

■