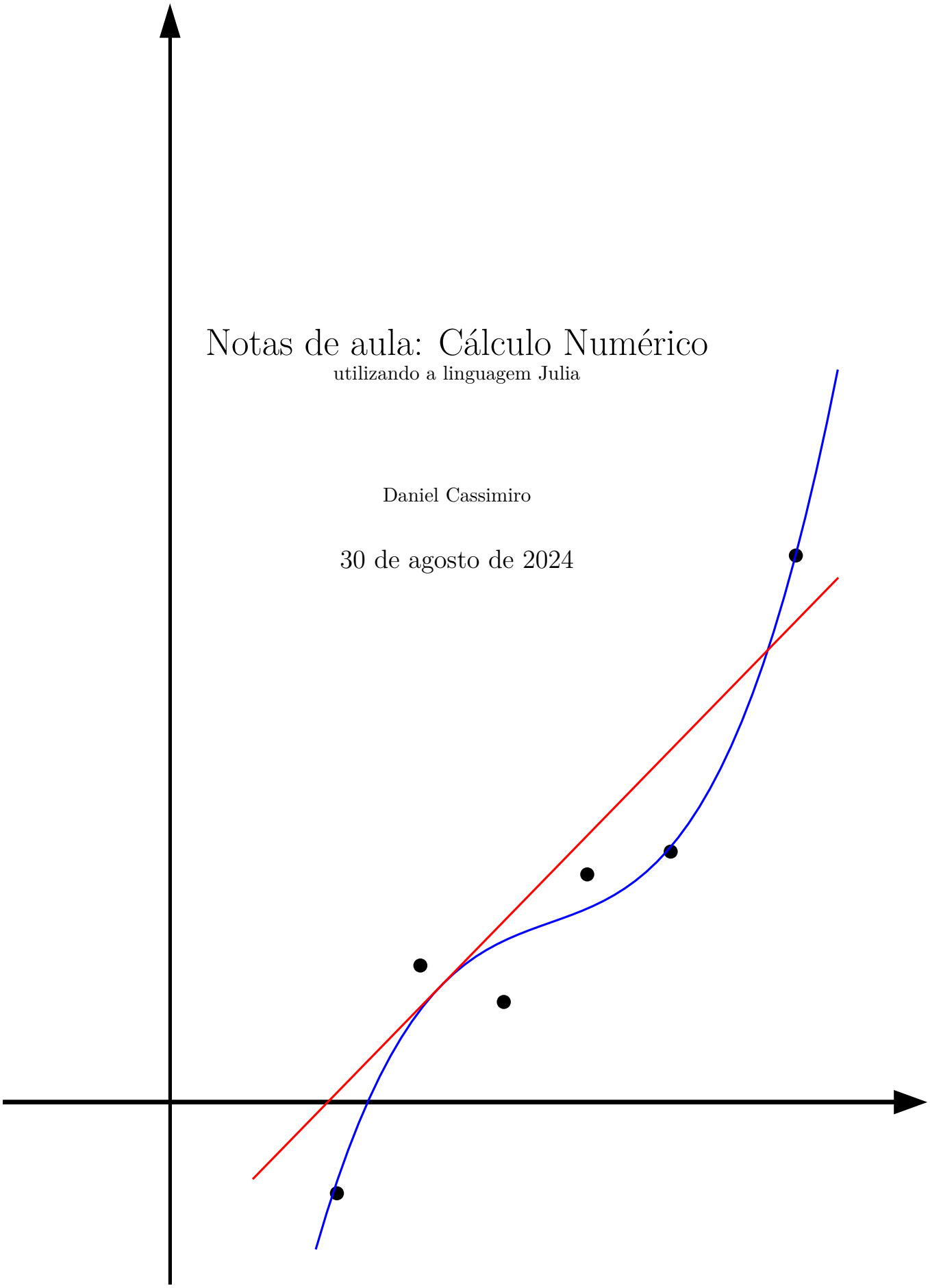
Notas de aula: Cálculo Numérico

utilizando a linguagem Julia

Daniel Cassimiro

30 de agosto de 2024



Licença

Sumário

Capítulo 1

Introdução

Cálculo numérico é a disciplina que estuda as técnicas para a solução aproximada de problemas matemáticos. Estas técnicas são de natureza analítica e computacional. As principais preocupações normalmente envolvem exatidão e desempenho.

Aliado ao aumento contínuo da capacidade de computação disponível, o desenvolvimento de métodos numéricos tornou a simulação computacional de problemas matemáticos uma prática usual nas mais diversas áreas científicas e tecnológicas. As então chamadas simulações numéricas são constituídas de um arranjo de vários esquemas numéricos dedicados a resolver problemas específicos como, por exemplo: resolver equações algébricas, resolver sistemas de equações lineares, interpolar e ajustar pontos, calcular derivadas e integrais, resolver equações diferenciais ordinárias etc. Neste livro, abordamos o desenvolvimento, a implementação, a utilização e os aspectos teóricos de métodos numéricos para a resolução desses problemas.

Trabalharemos com problemas que abordam aspectos teóricos e de utilização dos métodos estudados, bem como com problemas de interesse na engenharia, na física e na matemática aplicada.

A necessidade de aplicar aproximações numéricas decorre do fato de que esses problemas podem se mostrar intratáveis se dispomos apenas de meios puramente analíticos, como aqueles estudados nos cursos de cálculo e álgebra linear. Por exemplo, o teorema de Abel-Ruffini nos garante que não existe uma fórmula algébrica, isto é, envolvendo apenas operações aritméticas e radicais, para calcular as raízes de uma equação polinomial de qualquer grau, mas apenas casos particulares:

- Simplesmente isolar a incógnita para encontrar a raiz de uma equação do primeiro grau;
- Fórmula de Bhaskara para encontrar raízes de uma equação do segundo grau;
- Fórmula de Cardano para encontrar raízes de uma equação do terceiro grau;

- Existe expressão para equações de quarto grau;
- Casos simplificados de equações de grau maior que 4 onde alguns coeficientes são nulos também podem ser resolvidos.

Equações não polinomiais podem ser ainda mais complicadas de resolver exatamente, por exemplo:

$$\cos(x) = x \qquad \text{ou} \qquad xe^x = 10 \tag{1.1}$$

A maioria dos problemas envolvendo fenômenos reais produzem modelos matemáticos cuja solução analítica é difícil (ou impossível) de obter, mesmo quando provamos que a solução existe. Nesse curso propomos calcular aproximações numéricas para esses problemas, que apesar de, em geral, serem diferentes da solução exata, mostraremos que elas podem ser bem próximas.

Para entender a construção de aproximações é necessário estudar como funciona a aritmética implementada nos computadores e erros de arredondamento. Como computadores, em geral, usam uma base binária para representar números, começaremos falando em mudança de base.

Capítulo 2

Representação de números e aritmética de máquina

Neste capítulo, abordaremos formas de representar números reais em computadores. Iniciamos com uma discussão sobre representação posicional e mudança de base. Então, enfatizaremos a representação de números com quantidade finita de dígitos, mais especificamente, as representações de números inteiros, ponto fixo e ponto flutuante em computadores.

A representação de números e a aritmética em computadores levam aos chamados erros de arredondamento e de truncamento. Ao final deste capítulo, abordaremos os efeitos do erro de arredondamento na computação científica.

2.1 Sistema de numeração e mudança de base

Usualmente, utilizamos o sistema de numeração decimal, isto é, base 10, para representar números. Esse é um sistema de numeração em que a posição do algarismo indica a potência de 10 pela qual seu valor é multiplicado.

Exemplo 2.1.1. O número 293 é decomposto como

$$\begin{aligned} 293 &= 2 \text{ centenas} + 9 \text{ dezenas } + 3 \text{ unidades} \\ &= 2 \cdot 10^2 + 9 \cdot 10^1 + 3 \cdot 10^0. \end{aligned} \tag{2.1}$$

O sistema de numeração posicional também pode ser usado com outras bases. Vejamos a seguinte definição.

Definição 2.1.1 (Sistema de numeração de base b)**.** *Dado um número natural $b > 1$ e o conjunto de símbolos $\{\pm, \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots, \boldsymbol{b-1}\}$[1], a sequência de símbolos*

$$(d_n d_{n-1} \cdots d_1 d_0, d_{-1} d_{-2} \cdots)_b \tag{2.2}$$

[1]Para $b > 10$, veja a Observação 2.1.1.

representa o número positivo

$$d_n \cdot b^n + d_{n-1} \cdot b^{n-1} + \cdots + d_0 \cdot b^0 + d_{-1} \cdot b^{-1} + d_{-2} \cdot b^{-2} + \cdots \tag{2.3}$$

Para representar números negativos usamos o símbolo $-$ à esquerda do numeral[2].

Observação 2.1.1 ($b \geq 10$)**.** Para sistemas de numeração com base $b \geq 10$ é usual utilizar as seguintes notações:

- No sistema de numeração decimal ($b = 10$), costumamos representar o número sem os parênteses e o subíndice, ou seja,

$$\pm\, d_n d_{n-1} \ldots d_1 d_0{,}d_{-1} d_{-2} \ldots := \pm(d_n d_{n-1} \ldots d_1 d_0{,}d_{-1} d_{-2} \ldots)_{10}. \tag{2.4}$$

- Se $b > 10$, usamos as letras $A, B, C, \cdots$ para denotar os algarismos: $A = 10$, $B = 11$, $C = 12$, $D = 13$, $E = 14$, $F = 15$.

Exemplo 2.1.2 (Sistema binário)**.** O sistema de numeração em base dois é chamado de binário e os algarismos binários são conhecidos como *bits* (do inglês **binary digits**). Um *bit* pode assumir dois valores distintos: 0 ou 1. Por exemplo:

$$\begin{aligned} x &= (1001{,}101)_2 \\ &= 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} \\ &= 8 + 0 + 0 + 1 + 0{,}5 + 0 + 0{,}125 = 9{,}625. \end{aligned} \tag{2.5}$$

Ou seja, $(1001{,}101)_2$ é igual a 9,625 no sistema decimal.

Em `Julia` podemos converter o número $(1001{,}101)_2$ para a base decimal computando

```
julia> 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 + 1*2^-1 + 0*2^-2 + 1*2^-3
9.625
```

Exemplo 2.1.3 (Sistema quaternário)**.** No sistema quaternário a base b é igual a 4 e, portanto, temos o seguinte conjunto de algarismos $\{0, 1, 2, 3\}$. Por exemplo:

$$(301{,}2)_4 = 3 \cdot 4^2 + 0 \cdot 4^1 + 1 \cdot 4^0 + 2 \cdot 4^{-1} = 49{,}5. \tag{2.6}$$

Verifique no computador!

Exemplo 2.1.4 (Sistema octal)**.** No sistema octal a base é $b = 8$. Por exemplo:

$$\begin{aligned} (1357{,}24)_8 &= 1 \cdot 8^3 + 3 \cdot 8^2 + 5 \cdot 8^1 + 7 \cdot 8^0 + 2 \cdot 8^{-1} + 4 \cdot 8^{-2} \\ &= 512 + 192 + 40 + 7 + 0{,}25 + 0{,}0625 = 751{,}3125. \end{aligned} \tag{2.7}$$

Verifique no computador!

[2] O uso do símbolo $+$ é opcional na representação de números positivos.

Exemplo 2.1.5 (Sistema hexadecimal)**.** O sistema de numeração cuja base é $b = 16$ é chamado de sistema hexadecimal. Neste, temos o conjunto de algarismos $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$. Convertendo o número $(E2AC)_{16}$ para a base 10 temos

$$\begin{aligned}(E2AC)_{16} &= 14 \cdot 16^3 + 2 \cdot 16^2 + 10 \cdot 16^1 + 12 \cdot 16^0 \\ &= 57344 + 512 + 160 + 12 = 58028.\end{aligned} \tag{2.8}$$

Verifique no computador!

Observação 2.1.2. A linguagem `Julia` tem prefixos para representar números nas bases 2, 8 e 10. Por exemplo, temos:

```
>>> print(0b1001) # bin -> dec
9
>>> print(0o157) # oct -> dec
111
>>> print(0xbeba) # hex -> dec
48826
>>> string(9,base=2) # dec -> bin
1001
```

Também é possível usar a função int() para interpretar a representação de um inteiro em uma base com b entre 2 e 36. Por exemplo, temos:

```
>>> print(int('1001', 2)) # Base 2
9
>>> print(int('1001', 5)) # Base 5
126
>>> print(int('ABCD', 20)) # Base 20
84653
>>> print(int('zz', 36)) # Base 36
1295
```

Nos exemplos acima vimos como converter números representados em um sistema de numeração de base b para o sistema decimal. Agora, vamos estudar como fazer o processo inverso. Isto é, dado um número decimal $(X)_{10}$ queremos escrevê-lo em uma outra base b, isto é, queremos obter a seguinte representação:

$$\begin{aligned}(X)_{10} &= (d_n d_{n-1} \cdots d_0, d_{-1} \cdots)_b \\ &= d_n \cdot b^n + d_{n-1} \cdot b^{n-1} + \cdots + d_0 \cdot b^0 + d_{-1} \cdot b^{-1} + d_{-2} \cdot b^{-2} + \cdots\end{aligned} \tag{2.9}$$

Separando as partes inteira e fracionária de X, isto é, $X = X^{\mathrm{i}} + X^{\mathrm{f}}$, temos

$$X^{\mathrm{i}} = d_n \cdot b^n + \cdots + d_{n-1} b^{n-1} \cdots + d_1 \cdot b^1 + d_0 \cdot b^0 \tag{2.10}$$

e

$$X^{\mathrm{f}} = \frac{d_{-1}}{b^1} + \frac{d_{-2}}{b^2} + \cdots \tag{2.11}$$

Nosso objetivo é determinar os algarismos $\{d_n, d_{n-1}, ...\}$.

Primeiramente, vejamos como tratar a parte inteira X^{i}. Calculando o quociente de X^{i} por b, temos:

$$\frac{X^{\mathrm{i}}}{b} = \frac{d_0}{b} + d_1 + d_2 \cdot b^1 + \cdots + d_{n-1} \cdot b^{n-2} + d_n \cdot b^{n-1}. \tag{2.12}$$

Observe que d_0 é o resto da divisão de X^{i} por b, pois $d_1 + d_2 \cdot b^1 + \cdots + d_{n-1} \cdot b^{n-2} + d_n \cdot b^{n-1}$ é inteiro e $\frac{d_0}{b}$ é uma fração com $d_0 < b$. Da mesma forma, o resto da divisão de $d_1 + d_2 \cdot b^1 + \cdots + d_{n-1} \cdot b^{n-2} + d_n \cdot b^{n-1}$ por b é d_1. Ou seja, repetindo este processo encontramos os algarismos d_0, d_1, d_2, ..., d_n.

Vamos, agora, converter a parte fracionária X^{f} do número decimal X para o sistema de base b. Multiplicando X^{f} por b, temos

$$bX^{\mathrm{f}} = d_{-1} + \frac{d_{-2}}{b} + \frac{d_{-3}}{b^2} + \cdots \tag{2.13}$$

Observe que a parte inteira desse produto é d_{-1} e $\frac{d_{-2}}{b} + \frac{d_{-3}}{b^2} + \cdots$ é a parte fracionária. Quando multiplicamos $\frac{d_{-2}}{b} + \frac{d_{-3}}{b^2} + \cdots$ por b novamente, encontramos d_{-2}. Repetindo este processo encontramos os demais algarismos.

Exemplo 2.1.6. Vamos converter o número 9,625 para a base binária ($b = 2$). Primeiramente, decompomos 9,625 na soma de suas partes inteira e fracionária.

$$9{,}625 = 9 + 0{,}625. \tag{2.14}$$

Conversão da parte inteira. Para converter a parte inteira, fazemos sucessivas divisões por $b = 2$ obtendo

$$\begin{aligned} 9 &= 4 \cdot 2 + 1 && (2.15)\\ &= (2 \cdot 2 + 0) \cdot 2 + 1 && (2.16)\\ &= 2^3 + 1. && (2.17) \end{aligned}$$

Ou seja, temos que $9 = (1001)_2$.

Em `Julia`, podemos usar os comandos `int` (truncamento) e a operação `%` (resto da divisão) para computar esta conversão da seguinte forma

```
>>> x = 9
>>> d0 = x%2; x = int(x/2); print(f"d0 = {d0}, x = {x}")
d0 = 1, x = 4
```

```
>>> d1 = x%2; x = int(x/2); print(f"d1 = {d1}, x = {x}")
d1 = 0, x = 2
>>> d2 = x%2; x = int(x/2); print(f"d2 = {d2}, x = {x}")
d2 = 0, x = 1
>>> d3 = x%2; x = int(x/2); print(f"d3 = {d3}, x = {x}")
d3 = 1, x = 0
```

Conversão da parte fracionária. Para converter a parte fracionária, fazemos sucessivas multiplicações por $b = 2$ obtendo

$$0{,}625 = 1{,}25 \cdot 2^{-1} = 1 \cdot 2^{-1} + 0{,}25 \cdot 2^{-1} \tag{2.18}$$
$$= 1 \cdot 2^{-1} + (0{,}5 \cdot 2^{-1}) \cdot 2^{-1} = 1 \cdot 2^{-1} + 0{,}5 \cdot 2^{-2} \tag{2.19}$$
$$= 1 \cdot 2^{-1} + (1 \cdot 2^{-1}) \cdot 2^{-2} = 1 \cdot 2^{-1} + 1 \cdot 2^{-3}. \tag{2.20}$$

Ou seja, temos que $0{,}625 = (0{,}101)_2$.

No `GNU Octave`, podemos computar esta conversão da parte fracionária da seguinte forma

```
>> x = 0.625
x =  0.62500
>> d = fix(2*x), x = 2*x - d
d =  1
x =  0.25000
>> d = fix(2*x), x = 2*x - d
d = 0
x =  0.50000
>> d = fix(2*x), x = 2*x - d
d =  1
x = 0
```

Conclusão. Da conversão das partes inteira e fracionária de 9,625, obtemos $9 = (1001)_2$ e $0{,}625 = (0{,}101)_2$. Logo, concluímos que $9{,}625 = (1001{,}101)_2$.

Observação 2.1.3. O `GNU Octave` oferece algumas funções para a conversão de números inteiros em base decimal para uma base dada. Por exemplo, temos:

```
>> dec2base(9,2)
ans = 1001
>> dec2base(111,8)
ans = 157
>> dec2base(48826,16)
ans = BEBA
```

Observação 2.1.4. Uma maneira de converter um número dado em uma base b_1 para uma base b_2 é fazer em duas partes: primeiro converter o número dado na base b_1 para base decimal e depois converter para a base b_2.

Exercícios resolvidos

Esta seção carece de exercícios resolvidos. Participe da sua escrita.

Veja como em:

https://github.com/livroscolaborativos/CalculoNumerico

ER 2.1.1. Obtenha a representação do número $125{,}58\overline{3}$ na base 6.

Solução. Decompomos $125{,}58\overline{3}$ nas suas partes inteira 125 e fracionária $0{,}58\overline{3}$. Então, convertemos cada parte.

Conversão da parte inteira. Vamos escrever o número 125 na base 6. Para tanto, fazemos sucessivas divisões por 6 como segue:

$$\begin{aligned} 125 &= 20 \cdot 6 + 5 \quad (125 \text{ dividido por } 6 \text{ é igual a } 20 \text{ e resta } 5) \\ &= (3 \cdot 6 + 2) \cdot 6 + 5 = 3 \cdot 6^2 + 2 \cdot 6 + 5, \end{aligned} \tag{2.21}$$

logo $125 = (325)_6$.

Estes cálculos podem ser feitos no `GNU Octave` com o auxílio das funções `mod` e `fix`. A primeira calcula o resto da divisão entre dois números, enquanto que a segunda retorna a parte inteira de um número dado. No nosso exemplo, temos:

```
>> x = 125
x =  125
>> d = mod(x,6), x = fix(x/6)
d =  5
x =  20
>> d = mod(x,6), x = fix(x/6)
d =  2
x =  3
>> d = mod(x,6), x = fix(x/6)
d =  3
x = 0
```

Verifique!

Conversão da parte fracionária. Para converter $0{,}58\overline{3}$ para a base 6, fazemos sucessivas multiplicações por 6 como segue:

$$\begin{aligned} 0{,}58\overline{3} &= 3{,}5 \cdot 6^{-1} \quad (0{,}58\overline{3} \text{ multiplicado por } 6 \text{ é igual a } 3{,}5) \\ &= 3 \cdot 6^{-1} + 0{,}5 \cdot 6^{-1} \\ &= 3 \cdot 6^{-1} + (3 \cdot 6^{-1}) \cdot 6^{-1} \\ &= 3 \cdot 6^{-1} + 3 \cdot 6^{-2}, \end{aligned} \tag{2.22}$$

logo $0{,}58\overline{3} = (0{,}33)_6$.

No `GNU Octave`, podemos computar esta conversão da parte fracionária da seguinte forma

```
>> x = 0.58 + 1/3/100
x =  0.58333
>> d = fix(6*x), x = 6*x - d
d =  3
x =  0.50000
>> x = 0.5 #isso é realmente necessário?
x =  0.50000
>> d = fix(6*x), x = 6*x - d
d =  3
x = 0
```

$\Diamond$

ER 2.1.2. Obtenha a representação na base 4 do número $(101{,}01)_2$.

Solução. Começamos convertendo $(101{,}01)_2$ para a base decimal:

$$(101{,}01)_2 = 1 \cdot 2^2 + 1 \cdot 2^0 + 1 \cdot 2^{-2} = 5{,}25. \tag{2.23}$$

Então, convertemos 5,25 para a base 4. Para sua parte inteira, temos

$$5 = 1 \cdot 4 + 1 = (11)_4. \tag{2.24}$$

Para sua parte fracionária, temos

$$0{,}25 = 1 \cdot 4^{-1} = (0{,}1)_4. \tag{2.25}$$

Logo, $(101{,}01)_2 = (11{,}1)_4$. Verifique estas contas no computador! $\Diamond$

Exercícios

E 2.1.1. Converta para base decimal cada um dos seguintes números:

a) $(100)_2$

b) $(100)_3$

c) $(100)_b$

d) $(12)_5$

e) $(AA)_{16}$

f) $(7{,}1)_8$

g) $(3{,}12)_5$

E 2.1.2. Escreva os números abaixo na base decimal.

a) $(25{,}13)_8$

b) $(101{,}1)_2$

c) $(12F{,}4)_{16}$

d) $(11{,}2)_3$

E 2.1.3. Escreva o número 5,5 em base binária.

E 2.1.4. Escreva o número 17,109375 em base hexadecimal ($b = 16$).

E 2.1.5. Escreva cada número decimal na base b.

a) $7{,}\overline{6}$ na base $b = 5$

b) $29{,}1\overline{6}$ na base $b = 6$

E 2.1.6. Escreva $(12.4)_8$ em base decimal e binária.

E 2.1.7. Escreva cada número dado para a base b.

a) $(45{,}1)_8$ para a base $b = 2$

b) $(21{,}2)_8$ para a base $b = 16$

c) $(1001{,}101)_2$ para a base $b = 8$

d) $(1001{,}101)_2$ para a base $b = 16$

E 2.1.8. Quantos algarismos são necessários para representar o número 937163832173947 em base binária? E em base 7? Dica: Qual é o menor e o maior inteiro que pode ser escrito em dada base com N algarismos?

2.2 Notação científica e notação normalizada

Como vimos, no sistema posicional usual um número x na base b é representado por

$$x = \pm(d_n d_{n-1} \cdots d_0{,}d_{-1}d_{-2}d_{-3}\cdots)_b, \tag{2.26}$$

onde $d_n \neq 0$ e $d_i \in \{0, 1, \ldots, b-1\}$ é o dígito da i-ésima posição. Alternativamente, é costumeiro usarmos a chamada notação científica. Nesta, o número x é representado como

$$x = \pm(M)_b \times b^e, \tag{2.27}$$

onde $(M)_b = (d_m d_{m-1} \cdots d_0{,}d_{-1}d_{-2}d_{-3}\cdots)_b$ é chamada de mantissa e $e \in \mathbb{Z}$ é chamado de expoente de x.

Exemplo 2.2.1. a) O número 602,2141 em notação científica pode ser escrito como

$$602{,}2141 \times 10^0 = 60{,}22141 \times 10^1 = 0{,}6022141 \times 10^3. \tag{2.28}$$

b) O número $(1010{,}10)_2$ pode ser escrito em notação científica como $(10{,}1010)_2 \times 2^2$.

Observamos que um número pode ser representado de várias formas equivalentes em notação científica. Para termos uma representação única introduzimos o conceito de notação normalizada.

Definição 2.2.1. *Um número x na base b é dito estar representado em notação (científica) normalizada quando está escrito na forma*

$$x = (-1)^s (M)_b \times b^E, \tag{2.29}$$

onde $(M)_b = (d_0{,}d_{-1}d_{-2}d_{-3}\cdots)_b$, com $d_0 \neq 0$[3][4], s é 0 para positivo e 1 para negativo, E é o expoente.

Exemplo 2.2.2. Vejamos os seguintes casos:

a) O número 602,2141 em notação (científica) normalizada é representado por $6{,}022141 \times 10^2$.

b) O número $(1010{,}10)_2$ escrito em notação normalizada é $(1{,}01010)_2 \times 2^3$.

Observação 2.2.1. No `GNU Octave`, podemos controlar a impressão de números usando o comando `printf`. Por exemplo:

[3]Em algumas referências é usado $M_b = (0{,}d_{-1}d_{-2}d_{-3}\cdots)_b$.
[4]No caso de $x = 0$, $M_b = (0{,}00\cdots)_b$.

```
>> printf('%1.5f\n',-pi)
-3.14159
>> printf('%1.5e\n',-pi)
-3.14159e+00
```

No primeiro caso, obtemos a representação em ponto flutuante decimal com 6 dígitos do número $-\pi$. No segundo caso, obtemos a representação em notação científica normalizada com 6 dígitos.

Exercícios resolvidos

Esta seção carece de exercícios resolvidos. Participe da sua escrita.

Veja como em:

https://github.com/livroscolaborativos/CalculoNumerico

Exercícios

Esta seção carece de exercícios. Participe da sua escrita.

Veja como em:

https://github.com/livroscolaborativos/CalculoNumerico

E 2.2.1. Represente os seguintes números em notação científica normalizada:

$$\begin{aligned} &a)\ 299792{,}458 \qquad b)\ 66{,}2607 \times 10^{-35} \\ &c)\ 0{,}6674 \times 10^{-7} \quad d)\ 9806{,}65 \times 10^{1} \end{aligned} \tag{2.30}$$

(2.31)

E 2.2.2. Use o computador para verificar as respostas do Exercício 2.2.1.

2.3 Representação decimal finita

Em computadores, é usual representarmos números usando uma quantidade de dígitos finita. A quantidade a ser usada normalmente depende da precisão com que as computações estão sendo feitas. Ocorre que quando restringimos a representação a um número finito de dígitos, muitos números não podem ser representado

de forma exata, por exemplo, as dízimas infinitas e os números irracionais. Este fenômeno nos leva aos conceitos de número de dígitos significativos e de arredondamento.

Definição 2.3.1 (Número de dígitos significativos)**.** *Um número decimal* $x = \pm d_0{,}d_{-1}\cdots d_{-i}d_{-i-1}\cdots d_{-i-n}d_{-i-n-1}\cdot\times 10^E$ *é dito ter* n *dígitos significativos quando* $d_j = 0$ *para* $j \geq -i$ *e* $j \leq -i-n-1$.

Exemplo 2.3.1. O número $0{,}0602100 \times 10^{-3}$ tem 4 dígitos significativos.

2.3.1 Arredondamento de números

Quando representamos um número x com uma quantidade de dígitos menor que a de dígitos significativos acabamos com uma aproximação deste. Este procedimento é chamado arredondamento de um número. Mais precisamente, seja dado

$$x = \pm d_0{,}d_1d_2\ldots d_{k-1}d_kd_{k+1}\ldots d_n \times 10^e \tag{2.33}$$

em notação normalizada, isto é, $d_0 \neq 0$[5]. Podemos representar x com k dígitos da seguinte forma:

1. **Arredondamento por truncamento** (ou corte): aproximamos x por

$$\bar{x} = \pm d_0{,}d_1d_2\ldots d_k \times 10^e \tag{2.34}$$

   simplesmente descartando os dígitos d_j com $j > k$.

2. **Arredondamento por proximidade**[6]: se $d_{k+1} < 5$ aproximamos x por

$$\bar{x} = \pm d_0{,}d_1d_2\ldots d_k \times 10^e \tag{2.35}$$

   senão aproximamos x por[7]

$$\bar{x} = \pm(d_0{,}d_1d_2\ldots d_k + 10^{-k}) \times 10^e \tag{2.37}$$

[5] caso $x \neq 0$.

[6] com desempate infinito.

[7] Note que essas duas opções são equivalentes a somar 5 no dígito à direita do corte e depois arredondar por corte, ou seja, arredondar por corte

$$\pm\,(d_0{,}d_1d_2\ldots d_kd_{k+1} + 5\times 10^{-(k+1)}) \times 10^e \tag{2.36}$$

3. **Arredondamento por proximidade com desempate par**: se $d_{k+1} < 5$ aproximamos x por
$$\bar{x} = \pm d_0{,}d_1 d_2 \dots d_k \times 10^e. \tag{2.38}$$
Se $d_{k+1}{,}d_{d+2}d_{k+3} \dots > 5$ aproximamos x por
$$\bar{x} = \pm(d_0{,}d_1 d_2 \dots d_k + 10^{-k}) \times 10^e. \tag{2.39}$$
Agora, no caso de empate, i.e. $d_{k+1}{,}d_{k+2}d_{k+3} \dots = 5$, então x é aproximado por
$$\bar{x} = \pm d_0{,}d_1 d_2 \dots d_k \times 10^e \tag{2.40}$$
caso d_k seja par e, caso contrário, por
$$\bar{x} = \pm(d_0{,}d_1 d_2 \dots d_k + 10^{-k}) \times 10^e. \tag{2.41}$$

Observação 2.3.1. O arredondamento por proximidade é frequentemente empregado para arredondamentos de números reais para inteiros. Por exemplo:

- $x = 1{,}49$ arredonda-se para o inteiro 1.
- $x = 1{,}50$ arredonda-se para o inteiro 2.
- $x = 2{,}50$ arredonda-se para o inteiro 3.

```
>> round(1.49)
ans =  1
>> round(1.50)
ans =  2
>> round(2.50)
ans =  3
```

Exemplo 2.3.2. Represente os números $x_1 = 0{,}567$, $x_2 = 0{,}233$, $x_3 = -0{,}675$ e $x_4 = 0{,}314159265\ldots \times 10^1$ com dois dígitos significativos por truncamento e arredondamento.

Solução. Vejamos cada caso:

- Por truncamento:
$$x_1 = 0{,}56, \quad x_2 = 0{,}23, \quad x_3 = -0{,}67 \quad \text{e} \quad x_4 = 3{,}1. \tag{2.42}$$
No `GNU Octave`, podemos obter a representação de $x_3 = -0{,}675$ fazendo:

```
>> printf("%1.2f\n", ceil(-0.675*1e2)/1e2)
-0.67
```

e, em notação normalizada, temos:

```
>> printf("%1.1e\n", ceil(-0.675*1e2)/1e2)
-6.7e-01
```

Em `GNU Octave`, a representação de números por arredondamento por proximidade com desempate par é o padrão. Assim, para obtermos a representação desejada de $x_3 = 0{,}675$ fazemos:

```
>> printf("%1.2f\n", -0.675)
-0.68
```

e, em notação normalizada, temos:

```
>> printf("%1.1e\n", -0.675)
-6.8e-01
```

$\Diamond$

Observação 2.3.2. Observe que o arredondamento pode mudar todos os dígitos e o expoente da representação em ponto flutuante de um número dado. Por exemplo, o arredondamento de $0{,}9999 \times 10^{-1}$ com 3 dígitos significativos é $0{,}1 \times 10^{0}$.

Exercícios resolvidos

Esta seção carece de exercícios resolvidos. Participe da sua escrita.

Veja como em:

https://github.com/livroscolaborativos/CalculoNumerico

Exercícios

E 2.3.1. Aproxime os seguintes números para 2 dígitos significativos por arredondamento por truncamento.

(a) 1,159

(b) 7,399

(c) $-5{,}901$

E 2.3.2. Aproxime os seguintes números para 2 dígitos significativos por arredondamento por proximidade com desempate par.

(a) 1,151

(b) 1,15

(c) 2,45

(d) $-2{,}45$

E 2.3.3. O `GNU Octave` usa arredondamento por proximidade com desempate par como padrão. Assim sendo, por exemplo

```
>> printf('%1.1e\n', 1.25)
1.2e+00
```

Agora:

```
>> printf('%1.1e\n', 2.45)
2.5e+00
```

Não deveria ser 2.4? Explique o que está ocorrendo.

2.4 Representação de números em máquina

Os computadores, em geral, usam a base binária para representar os números, onde as posições, chamadas de bits, assumem as condições "verdadeiro" ou "falso", ou seja, 1 ou 0, respectivamente. Os computadores representam os números com uma quantidade fixa de bits, o que se traduz em um conjunto finito de números representáveis. Os demais números são tomados por proximidade àqueles conhecidos, gerando erros de arredondamento. Por exemplo, em aritmética de computador, o número 2 tem representação exata, logo $2^2 = 4$, mas $\sqrt{3}$ não tem representação finita, logo $(\sqrt{3})^2 \neq 3$.

Veja isso no `GNU Octave`:

```
>> 2^2 == 4
ans =  1
>> sqrt(3)^2 == 3
ans = 0
```

2.4.1 Números inteiros

Tipicamente, um número inteiro é armazenado em um computador como uma sequência de dígitos binários de comprimento fixo denominado **registro**.

Representação sem sinal

Um registro com n bits da forma

d_{n-1}	d_{n-2}	$\cdots$	d_1	d_0

representa o número $(d_{n-1}d_{n-2}...d_1d_0)_2$.

Assim, é possível representar números inteiros entre $2^n - 1$ e 0, sendo

$$\begin{aligned}
[111\ldots111] &= 2^{n-1} + 2^{n-2} + \cdots + 2^1 + 2^0 = 2^n - 1,\\
&\vdots\\
[000\ldots011] &= 3,\\
[000\ldots010] &= 2,\\
[000\ldots001] &= 1,\\
[000\ldots000] &= 0.
\end{aligned} \tag{2.43}$$

Exemplo 2.4.1. No `Scilab`,

```
-->uint8( bin2dec('00000011') )
   ans = 3
-->uint8( bin2dec('11111110') )
   ans = 254
```

Representação com bit de sinal

O bit mais significativo (o primeiro à esquerda) representa o sinal: por convenção, 0 significa positivo e 1 significa negativo. Um registro com n bits da forma

s	d_{n-2}	$\cdots$	d_1	d_0

representa o número $(-1)^s(d_{n-2}\ldots d_1d_0)_2$. Assim, é possível representar números inteiros entre $1 - 2^{n-1}$ e $2^{n-1} - 1$, com duas representações para o zero: $(1000\ldots000)_2$ e $(00000\ldots000)_2$.

Exemplo 2.4.2. Em um registro com 8 bits, teremos os números

$$\begin{aligned}
&[11111111] = -(2^6 + \cdots + 2 + 1) = -127,\\
&\qquad\vdots\\
&[10000001] = -1,\\
&[10000000] = -0,\\
&[01111111] = 2^6 + \cdots + 2 + 1 = 127,\\
&\qquad\vdots\\
&[00000010] = 2,\\
&[00000001] = 1,\\
&[00000000] = 0.
\end{aligned} \tag{2.44}$$

Representação complemento de dois

O bit mais significativo (o primeiro à esquerda) representa o coeficiente de -2^{n-1}. Um registro com n bits da forma:

d_{n-1}	d_{n-2}	$\cdots$	d_1	d_0

representa o número $-d_{n-1}2^{n-1} + (d_{n-2} \ldots d_1 d_0)_2$.

Observação 2.4.1. Note que todo registro começando com 1 será um número negativo.

Exemplo 2.4.3. O registro com 8 bits [01000011] representa o número:

$$-0(2^7) + (1000011)_2 = 2^6 + 2 + 1 = 67. \tag{2.45}$$

Agora, o registro [10111101] representa:

$$-1(2^7) + (0111101)_2 = -128 + 2^5 + 2^4 + 2^3 + 2^2 + 1 = -67. \tag{2.46}$$

Note que podemos obter a representação de -67 invertendo os dígitos de 67 em binário e somando 1.

Exemplo 2.4.4. Em um registro com 8 bits, teremos os números

$$\begin{aligned}
[11111111] &= -2^7 + 2^6 + \cdots + 2 + 1 = -1 \\
&\vdots \\
[10000001] &= -2^7 + 1 = -127 \\
[10000000] &= -2^7 = -128 \\
[01111111] &= 2^6 + \cdots + 2 + 1 = 127 \\
&\vdots \\
[00000010] &= 2 \\
[00000001] &= 1 \\
[00000000] &= 0
\end{aligned} \tag{2.47}$$

O `GNU Octave` trabalha com representação complemento de 2 de números inteiros. O comando `bitunpack` mostra o barramento de um número dado, por exemplo:

```
>> bitunpack(int8(3))
ans =
   1   1   0   0   0   0   0   0
```

mostra o barramento com 8 **bits** do número inteiro 3. Note que a ordem dos **bits** é inversa daquela apresentada no texto acima. Aqui, o **bit** mais à esquerda fornece o coeficiente de 2^0, enquanto o **bit** mais à direita fornece o coeficiente de -2^7.

O comando `bitpack` converte um barramento para o número em decimal, por exemplo:

```
>> bitpack(logical([1 1 0 0 0 0 0 0]),'int8')
ans = 3
```

2.4.2 Sistema de ponto fixo

O sistema de ponto fixo representa as partes inteira e fracionária do número com uma quantidade fixa de dígitos.

Exemplo 2.4.5. Em um computador de 32 bits que usa o sistema de ponto fixo, o registro

d_{31}	d_{30}	d_{29}	$\cdots$	d_1	d_0

pode representar o número

- $(-1)^{d_{31}}(d_{30}d_{29}\cdots d_{17}d_{16}, d_{15}d_{14}\cdots d_1d_0)_2$ se o sinal for representado por um dígito. Observe que, neste caso, o zero possui duas representações possíveis:

$$[10000000000000000000000000000000] \tag{2.48}$$

e

$$[00000000000000000000000000000000] \tag{2.49}$$

- $(d_{30}d_{29}\cdots d_{17}d_{16})_2 - d_{31}(2^{15} - 2^{-16}) + (0{,}d_{15}d_{14}\cdots d_1d_0)_2$ se o sinal do número estiver representado por uma implementação em complemento de um. Observe que o zero também possui duas representações possíveis:

$$[11111111111111111111111111111111] \tag{2.50}$$

e

$$[00000000000000000000000000000000] \tag{2.51}$$

- $(d_{30}d_{29}\cdots d_{17}d_{16})_2 - d_{31}2^{15} + (0{,}d_{15}d_{14}\cdots d_1d_0)_2$ se o sinal do número estiver representado por uma implementação em complemento de dois. Nesse caso o zero é unicamente representado por

$$[00000000000000000000000000000000] \tag{2.52}$$

Observe que 16 dígitos são usados para representar a parte fracionária, 15 são para representar a parte inteira e um dígito, o d_{31}, está relacionado ao sinal do número.

2.4.3 Sistema de ponto flutuante

O sistema de ponto flutuante não possui quantidade fixa de dígitos para as partes inteira e fracionária do número.

Podemos definir uma máquina F em ponto flutuante de dois modos:

$$F(\beta,|M|,|E|,BIAS) \text{ ou } F(\beta,|M|,E_{MIN},E_{MAX}) \tag{2.53}$$

onde

- β é a base (em geral 2 ou 10),
- $|M|$ é o número de dígitos da mantissa,
- $|E|$ é o número de dígitos do expoente,
- $BIAS$ é um valor de deslocamento do expoente (veja a seguir),

- E_{MIN} é o menor expoente,
- E_{MAX} é o maior expoente.

Considere uma máquina com um registro de 64 bits e base $\beta = 2$. Pelo padrão IEEE754, 1 bit é usado para o sinal, 11 bits para o expoente e 52 bits são usados para o significando tal que

s	c_{10}	c_9	$\cdots$	c_0	m_1	m_2	$\cdots$	m_{51}	m_{52}

represente o número (o $BIAS = 1023$ por definição)

$$x = (-1)^s M \times 2^{c-BIAS}, \tag{2.54}$$

onde a **característica** é representada por

$$c = (c_{10}c_9 \cdots c_1c_0)_2 = c_{10}2^{10} + \cdots + c_1 2^1 + c_0 2^0 \tag{2.55}$$

e o significando por

$$M = (1.m_1m_2 \cdots m_{51}m_{52})_2. \tag{2.56}$$

Observação 2.4.2. Em base 2 não é necessário armazenar o primeiro dígito (por quê?).

Exemplo 2.4.6. O registro

$$[0|100\ 0000\ 0000|1010\ 0000\ 0000 \ldots 0000\ 0000] \tag{2.57}$$

representa o número

$$(-1)^0(1 + 2^{-1} + 2^{-3}) \times 2^{1024-1023} = (1 + 0.5 + 0.125)2 = 3.25. \tag{2.58}$$

Observação 2.4.3. No `GNU Octave`, podemos usar os comandos `bitpack` e `bitunpack` transformar um registro de ponto flutuante de 64 **bits** em decimal e vice-versa. Entretanto, um tal registro no `GNU Octave` tem a seguinte estrutura

$$[m_{52}m_{51}m_{50} \ldots m_1|c_0c_1c_2 \cdots c_{10}|s]. \tag{2.59}$$

Desta forma, o decimal 3.25 tem, aqui, o registro

$$[000 \ldots 0101|000 \ldots 01|0]. \tag{2.60}$$

O que podemos verificar com o comando

```
>> bitpack(logical([zeros(1,49) 1 0 1 zeros(1,10) 1 0]),'double')
ans =  3.2500
```

O expoente deslocado

Uma maneira de representar os expoentes inteiros é deslocar todos eles uma mesma quantidade. Desta forma permitimos a representação de números negativos e a ordem deles continua crescente. O expoente é representado por um inteiro sem sinal do qual é deslocado o **BIAS**.

Tendo $|E|$ dígitos para representar o expoente, geralmente o $BIAS$ é predefinido de tal forma a dividir a tabela ao meio de tal forma que o expoente *um* seja representado pelo sequência $[100\ldots000]$.

Exemplo 2.4.7. Com 64 bits, pelo padrão $IEEE754$, temos que $|E| := 11$. Assim, $(100\ 0000\ 0000)_2 = 2^{10} = 1024$. Como queremos que esta sequência represente o 1, definimos $BIAS := 1023$, pois

$$1024 - BIAS = 1. \tag{2.61}$$

Com 32 bits, temos $|E| := 8$ e $BIAS := 127$. E com 128 bits, temos $|E| := 15$ e $BIAS := 16383$.

Com $|E| = 11$ temos

$$\begin{aligned}
&[111\ 1111\ 1111] = \text{reservado}\\
&[111\ 1111\ 1110] = 2046 - BIAS = 1023_{10} = E_{MAX}\\
&\qquad\qquad\vdots =\\
&[100\ 0000\ 0001] = 2^{10} + 1 - BIAS = 2_{10}\\
&[100\ 0000\ 0000] = 2^{10} \quad - BIAS = 1_{10}\\
&[011\ 1111\ 1111] = 1023 \quad - BIAS = 0_{10}\\
&[011\ 1111\ 1110] = 1022 \quad - BIAS = -1_{10}\\
&\qquad\qquad\vdots =\\
&[000\ 0000\ 0001] = 1 - BIAS = -1022 = E_{MIN}\\
&[000\ 0000\ 0000] = \text{reservado}
\end{aligned} \tag{2.62}$$

O maior expoente é dado por $E_{MAX} = 1023$ e o menor expoente é dado por $E_{MIN} = -1022$.

O menor número representável positivo é dado pelo registro

$$[0|000\ 0000\ 0001|0000\ 0000\ 0000\ldots0000\ 0000] \tag{2.63}$$

quando $s = 0$, $c = 1$ e $M = (1.000...000)_2$, ou seja,

$$MINR = (1+0)_2 \times 2^{1-1023} \approx 0.2225 \times 10^{-307}. \tag{2.64}$$

O maior número representável é dado por

$$[0|111\ 1111\ 1110|1111\ 1111\ \cdots\ 1111\ 1111] \tag{2.65}$$

quando $s = 0$, $c = 2046$ e $M = (1.1111\ 1111 \cdots\ 1111)_2 = 2 - 2^{-52}$, ou seja,

$$MAXR = (2 - 2^{-52}) \times 2^{2046-1023} \approx 2^{1024} \approx 0.17977 \times 10^{309}. \tag{2.66}$$

Observação 2.4.4. No `GNU Octave`, podemos obter o maior e o menor `double` positivo não nulo com os comandos:

```
>> realmax
ans =   1.7977e+308
>> realmin
ans =   2.2251e-308
```

Casos especiais

O **zero** é um caso especial representado pelo registro

$$[0|000\ 0000\ 0000|0000\ 0000\ 0000...0000\ 0000] \tag{2.67}$$

Os expoentes **reservados** são usados para casos especiais:

- $c = [0000...0000]$ é usado para representar o zero (se $m = 0$) e os números subnormais (se $m \neq 0$).
- $c = [1111...1111]$ é usado para representar o infinito (se $m = 0$) e NaN (se $m \neq 0$).

Os números subnormais[8] tem a forma

$$x = (-1)^s (0.m_1 m_2 \cdots m_{51} m_{52})_2 \times 2^{1-BIAS}. \tag{2.68}$$

2.4.4 Precisão e épsilon de máquina

A **precisão** p de uma máquina é o número de dígitos significativos usado para representar um número. Note que $p = |M| + 1$ em binário e $p = |M|$ para outras bases.

O **épsilon de máquina**, $\epsilon_{mach} = \epsilon$, é definido de forma que $1 + \epsilon$ seja o menor número representável maior que 1, isto é, $1 + \epsilon$ é representável, mas não existem números representáveis em $(1, 1 + \epsilon)$.

[8] Note que poderíamos definir números um pouco menores que o $MINR$.

Exemplo 2.4.8. Com 64 bits, temos que o épsilon será dado por

$$\begin{aligned} 1 \rightarrow \quad & (1.0000\ 0000....0000)_2 \times 2^0 \\ \epsilon \rightarrow + & (0.0000\ 0000....0001)_2 \times 2^0 = 2^{-52} \\ & (1.0000\ 0000....0001)_2 \times 2^0 \neq 1 \end{aligned} \tag{2.69}$$

Assim, $\epsilon = 2^{-52}$.

Observação 2.4.5. No `GNU Octave`, o épsilon de máquina é representado pela constante `eps`. Observe os seguintes resultados:

```
>> 1 + 1e-16 == 1
ans =  1
>> 1 + eps == 1
ans = 0
```

2.4.5 Distribuição dos números

Utilizando uma máquina em ponto flutuante, temos um número finito de números que podemos representar.

Um número muito pequeno geralmente é aproximado por zero (**underflow**) e um número muito grande (**overflow**) geralmente faz o cálculo parar. Além disso, os números não estão uniformemente espaçados no eixo real. Números pequenos estão bem próximos enquanto que números com expoentes grandes estão bem distantes.

Se tentarmos armazenar um número que não é representável, devemos utilizar o número mais próximo, gerando os erros de arredondamento.

Observação 2.4.6. Veja como o `GNU Octave` se comporta nos seguintes casos de exceção:

```
>> 0/0
warning: division by zero
ans = NaN
>> 1/0
warning: division by zero
ans = Inf
>> 1/-0
warning: division by zero
ans = -Inf
>> 2*realmax
ans = Inf
>> 1/2^9999
ans = 0
```

Exercícios

E 2.4.1. Usando a representação complemento de dois de números inteiros com 8 **bits**, escreva o número decimal que corresponde aos seguintes barramentos:

a) [01100010].

b) [00011101].

c) [10000000].

d) [11100011].

e) [11111111]

E 2.4.2. Usando a representação complemento de dois de números inteiros com 16 **bits**, escreva o número decimal que corresponde aos seguintes barramentos:

a) [0110001001100010].

b) [0001110100011101].

c) [1110001011100011].

d) [1111111111111111].

E 2.4.3. Usando a representação complemento de dois de números inteiros com 8 **bits** no `GNU Octave`, escreva o número decimal que corresponde aos seguintes barramentos:

a) [01100010].

b) [00011101].

c) [00010010].

E 2.4.4. Usando a representação complemento de dois de números inteiros com 16 **bits** no `GNU Octave`, escreva o número decimal que corresponde aos seguintes barramentos:

a) [0110001001100010].

b) [0001110100011101].

c) [0001001011100010].

E 2.4.5. Usando a representação de ponto flutuante com 64 **bits**, escreva o número decimal que corresponde aos seguintes barramentos:

a) [0|10000000000|111000 . . . 0].

b) [1|10000000001|0111000 . . . 0].

E 2.4.6. Explique a diferença entre o sistema de ponto fixo e ponto flutuante.

E 2.4.7. Usando a representação de `double` no `GNU Octave`, escreva o número decimal que corresponde aos seguintes barramentos:

a) [000 . . . 0111|00000000001|0].

b) [000 . . . 01110|10000000001|1].

E 2.4.8. Considere a seguinte rotina escrita para ser usada no `GNU Octave`:

```
x=1
while x+1>x
    x=x+1
end
```

Explique se esta rotina finaliza em tempo finito, em caso afirmativo calcule a ordem de grandeza do tempo de execução supondo que cada passo do laço demore $10^{-7}s$. Justifique sua reposta.

2.5 Tipos de erros

Em geral, os números não são representados de forma exata nos computadores. Isto nos leva ao chamado erro de arredondamento. Quando resolvemos problemas com técnicas numéricas, estamos sujeitos a este e outros tipos de erros. Nesta seção, veremos quais são estes erros e como controlá-los, quando possível.

Quando fazemos aproximações numéricas, os erros são gerados de várias formas, sendo as principais delas as seguintes:

1. **Incerteza dos dados** são devidos aos erros nos dados de entrada. Quando o modelo matemático é oriundo de um problema físico, existe incerteza nas medidas feitas pelos instrumentos de medição, que possuem acurácia finita.

2. **Erros de Arredondamento** são aqueles relacionados com as limitações existentes na forma de representar números em máquina.

3. **Erros de Truncamento** surgem quando aproximamos um conceito matemático formado por uma sequência infinita de passos por de um procedimento finito. Por exemplo, a definição de integral é dada por um processo de limite de somas. Numericamente, aproximamos por um soma finita. O erro de truncamento deve ser estudado analiticamente para cada método empregado e normalmente envolve matemática mais avançada que a estudado em um curso de graduação.

Uma questão fundamental é a quantificação dos erros imbricados na computação da solução de um dado problema. Para tanto, precisamos definir medidas de erros (ou de exatidão). As medidas de erro mais utilizadas são o **erro absoluto** e o **erro relativo**.

Definição 2.5.1 (Erro absoluto e relativo)**.** *Seja x um número real e $\overline{x}$, sua aproximação. O* ***erro absoluto*** *da aproximação $\overline{x}$ é definido como*

$$|x - \overline{x}|. \tag{2.70}$$

O ***erro relativo*** *da aproximação $\overline{x}$ é definido como*

$$\frac{|x - \overline{x}|}{|x|}, \quad x \neq 0. \tag{2.71}$$

Observação 2.5.1. Observe que o erro relativo é adimensional e, muitas vezes, é expresso em porcentagens. Mais precisamente, o erro relativo em porcentagem da aproximação $\overline{x}$ é dado por

$$\frac{|x - \bar{x}|}{|x|} \times 100\%. \tag{2.72}$$

Exemplo 2.5.1. Sejam $x = 123456{,}789$ e sua aproximação $\bar{x} = 123000$. O erro absoluto é

$$|x - \bar{x}| = |123456{,}789 - 123000| = 456{,}789 \tag{2.73}$$

e o erro relativo é

$$\frac{|x - \bar{x}|}{|x|} = \frac{456{,}789}{123456{,}789} \approx 0{,}00369999 \text{ ou } 0{,}36\% \tag{2.74}$$

Exemplo 2.5.2. Sejam $y = 1{,}23456789$ e $\bar{y} = 1{,}13$. O erro absoluto é

$$|y - \bar{y}| = |1{,}23456789 - 1{,}13| = 0{,}10456789 \tag{2.75}$$

que parece pequeno se compararmos com o exemplo anterior. Entretanto o erro relativo é

$$\frac{|y-\bar{y}|}{|y|} = \frac{0{,}10456789}{1{,}23456789} \approx 0{,}08469999 \text{ ou } 8{,}4\% \tag{2.76}$$

Note que o erro relativo leva em consideração a escala do problema.

Exemplo 2.5.3. Observe os erros absolutos e relativos em cada caso a seguir:

x	$\bar{x}$	Erro absoluto	Erro relativo
$0{,}\overline{3} \times 10^{-2}$	$0{,}3 \times 10^{-2}$	$0{,}\overline{3} \times 10^{-3}$	10%
$0{,}\overline{3}$	$0{,}3$	$0{,}\overline{3} \times 10^{-2}$	10%
$0{,}\overline{3} \times 10^{2}$	$0{,}3 \times 10^{2}$	$0{,}\overline{3} \times 10^{1}$	10%

Outra forma de medir a exatidão de uma aproximação numérica é contar o **número de dígitos significativos corretos** em relação ao valor exato.

Definição 2.5.2 (Número de dígitos significativos corretos)**.** *A aproximação $\overline{x}$ de um número x tem s* ***dígitos significativos corretos*** *quando*[9]

$$\frac{|x-\overline{x}|}{|x|} < 5 \times 10^{-s}. \tag{2.78}$$

Exemplo 2.5.4. Vejamos os seguintes casos:

a) A aproximação de $x = 0{,}333333$ por $\overline{x} = 0{,}333$ tem 3 dígitos significativos corretos, pois

$$\frac{|x-\overline{x}|}{|x|} = \frac{0{,}000333}{0{,}333333} \approx 0{,}000999 \leq 5 \times 10^{-\mathbf{3}}. \tag{2.79}$$

b) Considere as aproximações $\bar{x}_1 = 0{,}666$ e $\bar{x}_2 = 0{,}667$ de $x = 0{,}666888$. Os erros relativos são

$$\frac{|x-\bar{x}_1|}{|x|} = \frac{|0{,}666888 - 0{,}666|}{0{,}666888} \approx 0{,}00133... < 5 \times 10^{-3}. \tag{2.80}$$

[9]Esta definição é apresentada em [3]. Não existe uma definição única na literatura para o conceito de dígitos significativos corretos, embora não precisamente equivalentes, elas transmitem o mesmo conceito. Uma maneira de interpretar essa regra é: calcula-se o erro relativo na forma normalizada e a partir da ordem do expoente temos o número de dígitos significativos corretos. Como queremos o expoente, podemos estimar s por

$$DIGSE(x,\bar{x}) = s \approx int\left|\log_{10}\frac{|x-\overline{x}|}{|x|}\right|. \tag{2.77}$$

$$\frac{|x-\bar{x}_2|}{|x|} = \frac{|0{,}666888 - 0{,}667|}{0{,}666888} \approx 0{,}000167... < 5 \times 10^{-4}. \tag{2.81}$$

Note que $\bar{x}_1$ possui 3 dígitos significativos corretos e $\bar{x}_2$ possui 4 dígitos significativos (o quarto dígito é o dígito 0 que não aparece à direita, i.e, $\bar{x}_2 = 0.6670$. Isto também leva a conclusão que x_2 aproxima melhor o valor de x do que x_1 pois está mais próximo de x.

c) $\overline{x} = 9{,}999$ aproxima $x = 10$ com 4 dígitos significativos corretos, pois

$$\frac{|x-\overline{x}|}{|x|} = \frac{|10 - 9{,}999|}{10} \approx 0{,}0000999... < 5 \times 10^{-4}. \tag{2.82}$$

d) Considere as aproximações $\overline{x}_1 = 1{,}49$ e $\overline{x}_2 = 1{,}5$ de $x = 1$. Da definição, temos que 1,49 aproxima 1 com um dígito significativo correto (verifique), enquanto 1,5 tem zero dígito significativo correto, pois:

$$\frac{|1 - 1{,}5|}{|1|} = 5 \times 10^{-1} < 5 \times 10^{0}. \tag{2.83}$$

Exercícios

E 2.5.1. Calcule os erros absoluto e relativo das aproximações $\bar{x}$ para x em cada caso:

a) $x = \pi = 3{,}14159265358979\ldots$ e $\bar{x} = 3{,}141$

b) $x = 1{,}00001$ e $\bar{x} = 1$

c) $x = 100001$ e $\bar{x} = 100000$

E 2.5.2. Arredonde os seguintes números para cinco algarismos significativos:

a) 1,7888544

b) 1788,8544

c) 0,0017888544

d) 0,004596632

e) $2{,}1754999 \times 10^{-10}$

f) $2{,}1754999 \times 10^{10}$

E 2.5.3. Represente os seguintes números com três dígitos significativos usando arredondamento por truncamento e arredondamento por proximidade.

a) 3276.

b) 42,55.

c) 0,00003331.

E 2.5.4. Usando a Definição 2.5.2, verifique quantos são os dígitos significativos corretos na aproximação de x por $\bar{x}$.

a) $x = 2{,}5834$ e $\bar{x} = 2{,}6$

b) $x = 100$ e $\bar{x} = 99$

E 2.5.5. Resolva a equação $0{,}1x - 0{,}01 = 12$ usando arredondamento com três dígitos significativos em cada passo e compare com o resultado exato.

E 2.5.6. Calcule o erro relativo e absoluto envolvido nas seguintes aproximações e expresse as respostas com três algarismos significativos corretos.

a) $x = 3{,}1415926535898$ e $\tilde{x} = 3{,}141593$

b) $x = \frac{1}{7}$ e $\tilde{x} = 1{,}43 \times 10^{-1}$

2.6 Erros nas operações elementares

O erro relativo presente nas operações elementares de adição, subtração, multiplicação e divisão é da ordem do épsilon de máquina. Se estivermos usando o sistema de numeração *binary64* ou *double*, temos $\epsilon = 2^{-52} \approx 2{,}22 \cdot 10^{-16}$.

Este erro é bem pequeno para a maioria das aplicações! Assumindo que x e y são representados com todos dígitos corretos, esperamos ter aproximadamente 15 dígitos significativos corretos quando fazemos uma das operações $x + y$, $x - y$, $x \times y$ ou x/y.

Mesmo que fizéssemos, por exemplo, 1000 operações elementares sucessivas em ponto flutuante, teríamos, no pior dos casos, acumulado todos esses erros e perdido 3 casas decimais ($1000 \times 10^{-15} \approx 10^{-12}$).

Entretanto, existem situações em que o erro se propaga de forma muito catastrófica, em especial, quando subtraímos números positivos muito próximos.

2.7 Cancelamento catastrófico

Quando fazemos subtrações com números muito próximos entre si, ocorre o que chamamos de "cancelamento catastrófico", onde podemos perder vários dígitos de precisão em uma única subtração.

Exemplo 2.7.1. Efetue a operação

$$0{,}987624687925 - 0{,}987624 = 0{,}687925 \times 10^{-6} \tag{2.87}$$

usando arredondamento com seis dígitos significativos e observe a diferença se comparado com resultado sem arredondamento.

Solução. Os números arredondados com seis dígitos para a mantissa resultam na seguinte diferença

$$0{,}987625 - 0{,}987624 = 0{,}100000 \times 10^{-5} \tag{2.88}$$

Observe que os erros relativos entre os números exatos e aproximados no lado esquerdo são bem pequenos,

$$\frac{|0{,}987624687925 - 0{,}987625|}{|0{,}987624687925|} = 0{,}00003159 \tag{2.89}$$

e

$$\frac{|0{,}987624 - 0{,}987624|}{|0{,}987624|} = 0\%, \tag{2.90}$$

enquanto no lado direito o erro relativo é enorme:

$$\frac{|0{,}100000 \times 10^{-5} - 0{,}687925 \times 10^{-6}|}{0{,}687925 \times 10^{-6}} = 45{,}36\%. \tag{2.91}$$

$\Diamond$

Exemplo 2.7.2. Considere o problema de encontrar as raízes da equação de segundo grau

$$x^2 + 300x - 0{,}014 = 0, \tag{2.92}$$

usando seis dígitos significativos.

Aplicando a fórmula de Bhaskara com $a = 0{,}100000 \times 10^{1}$, $b = 0{,}300000 \times 10^{3}$ e $c = 0{,}140000 \times 10^{-1}$, temos o discriminante:

$$\begin{aligned}
\Delta &= b^2 - 4 \cdot a \cdot c && (2.93)\\
&= 0{,}300000 \times 10^{3} \times 0{,}300000 \times 10^{3} && (2.94)\\
&+ 0{,}400000 \times 10^{1} \times 0{,}100000 \times 10^{1} \times 0{,}140000 \times 10^{-1} && (2.95)\\
&= 0{,}900000 \times 10^{5} + 0{,}560000 \times 10^{-1} && (2.96)\\
&= 0{,}900001 \times 10^{5} && (2.97)
\end{aligned}$$

e as raízes:

$$x_1,x_2 = \frac{-0{,}300000 \times 10^3 \pm \sqrt{\Delta}}{0{,}200000 \times 10^1} \tag{2.98}$$

$$= \frac{-0{,}300000 \times 10^3 \pm \sqrt{0{,}900001 \times 10^5}}{0{,}200000 \times 10^1} \tag{2.99}$$

$$= \frac{-0{,}300000 \times 10^3 \pm 0{,}300000 \times 10^3}{0{,}200000 \times 10^1} \tag{2.100}$$

$$\tag{2.101}$$

Então, as duas raízes obtidas com erros de arredondamento, são:

$$\begin{aligned} \tilde{x}_1 &= \frac{-0{,}300000 \times 10^3 - 0{,}300000 \times 10^3}{0{,}200000 \times 10^1} \\ &= -\frac{0{,}600000 \times 10^3}{0{,}200000 \times 10^1} = -0{,}300000 \times 10^3 \end{aligned} \tag{2.102}$$

e

$$\tilde{x}_2 = \frac{-0{,}300000 \times 10^3 + 0{,}300000 \times 10^3}{0{,}200000 \times 10^1} = 0{,}000000 \times 10^0 \tag{2.103}$$

No entanto, os valores das raízes com seis dígitos significativos livres de erros de arredondamento, são:

$$x_1 = -0{,}300000 \times 10^3 \quad \text{e} \quad x_2 = 0{,}466667 \times 10^{-4}. \tag{2.104}$$

Observe que a primeira raiz apresenta seis dígitos significativos corretos, mas a segunda não possui nenhum dígito significativo correto.

Observe que isto acontece porque b^2 é muito maior que $4ac$, ou seja, $b \approx \sqrt{b^2 - 4ac}$, logo a diferença

$$-b + \sqrt{b^2 - 4ac} \tag{2.105}$$

estará próxima de zero. Uma maneira de evitar o cancelamento catastrófico é aplicar procedimentos analíticos na expressão para eliminar essa diferença. Um técnica padrão consiste usar uma expansão em série de Taylor em torno da origem, tal como:

$$\sqrt{1-x} = 1 - \frac{1}{2}x + O(x^2). \tag{2.106}$$

Substituindo esta aproximação na fórmula de Bhaskara, temos:

$$\begin{aligned} x &= \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} && (2.107)\\ &= \frac{-b \pm b\sqrt{1 - \frac{4ac}{b^2}}}{2a} && (2.108)\\ &\approx \frac{-b \pm b\left(1 - \frac{4ac}{2b^2}\right)}{2a} && (2.109)\\ & && (2.110) \end{aligned}$$

Observe que $\frac{4ac}{b^2}$ é um número pequeno e por isso a expansão faz sentido. Voltamos no exemplo anterior e calculamos as duas raízes com o nova expressão

$$\begin{aligned} \tilde{x}_1 &= \frac{-b - b + \frac{4ac}{2b}}{2a} = -\frac{b}{a} + \frac{c}{b} && (2.111)\\ &= -\frac{0{,}300000 \times 10^3}{0{,}100000 \times 10^1} - \frac{0{,}140000 \times 10^{-1}}{0{,}300000 \times 10^3} && (2.112)\\ &= -0{,}300000 \times 10^3 - 0{,}466667 \times 10^{-4} && (2.113)\\ &= -0{,}300000 \times 10^3 && (2.114) \end{aligned}$$

$$\begin{aligned} \tilde{x}_2 &= \frac{-b + b - \frac{4ac}{2b}}{2a} && (2.115)\\ &= -\frac{4ac}{4ab} && (2.116)\\ &= -\frac{c}{b} = -\frac{-0{,}140000 \times 10^{-1}}{0{,}300000 \times 10^3} = 0{,}466667 \times 10^{-4} && (2.117)\\ & && (2.118) \end{aligned}$$

Observe que o efeito catastrófico foi eliminado.

Observação 2.7.1. O cancelamento catastrófico também poderia ter sido evitado através do seguinte truque analítico:

$$\begin{aligned} x_2 &= \frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \cdot \frac{-b - \sqrt{b^2 - 4ac}}{-b - \sqrt{b^2 - 4ac}} && (2.119)\\ &= \frac{b^2 - (b^2 - 4ac)}{2a\left(-b - \sqrt{b^2 - 4ac}\right)} = \frac{4ac}{2a\left(-b - \sqrt{b^2 - 4ac}\right)} && (2.120)\\ &= -\frac{2c}{\left(b + \sqrt{b^2 - 4ac}\right)} && (2.121) \end{aligned}$$

2.8 Condicionamento de um problema

Nesta seção, utilizaremos a seguinte descrição abstrata para o conceito de "resolver um problema": dado um conjunto de dados de entrada, encontrar os dados de saída. Se denotamos pela variável x os dados de entrada e pela variável y os dados de saída, resolver o problema significa encontrar y dado x. Em termos matemáticos, a resolução de um problema é realizada pelo mapeamento $f : x \to y$, ou simplesmente $y = f(x)$.

É certo que, na maioria das aplicações, os dados de entrada do problema — isto é, x — não são conhecidos com total exatidão, devido a diversas fontes de erros, como incertezas na coleta dos dados e erros de arredondamento. O conceito de condicionamento está relacionado à forma como os erros nos dados de entrada influenciam os dados de saída.

Para fins de análise, denotaremos por x, os dados de entrada com precisão absoluta e por x^*, os dados com erro. Definiremos também a solução y^*, do problema com dados de entrada x^*, ou seja, $y^* = f(x^*)$.

Estamos interessados em saber se os erros cometidos na entrada $\Delta x = x^* - x$ influenciaram na saída do problema $\Delta y = y^* - y$. No caso mais simples, temos que $x \in \mathbb{R}$ e $y \in \mathbb{R}$. Assumindo que f seja diferenciável, a partir da série de Taylor

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x \tag{2.122}$$

obtemos (subtraindo $f(x)$ dos dois lados)

$$\Delta y = f(x + \Delta x) - f(x) \approx f'(x)\Delta x \tag{2.123}$$

Para relacionarmos os erros relativos, dividimos o lado esquerdo por y, o lado direito por $f(x) = y$ e obtemos

$$\frac{\Delta y}{y} \approx \frac{f'(x)}{f(x)} \frac{x\Delta x}{x} \tag{2.124}$$

sugerindo a definição de número de condicionamento de um problema.

Definição 2.8.1. *Seja f uma função diferenciável. O* ***número de condicionamento*** *de um problema é definido como*

$$\kappa_f(x) := \left| \frac{xf'(x)}{f(x)} \right| \tag{2.125}$$

e fornece uma estimativa de quanto os erros relativos na entrada $\left|\frac{\Delta x}{x}\right|$ serão amplificados na saída $\left|\frac{\Delta y}{y}\right|$.

De modo geral, quando f depende de várias variáveis, podemos obter

$$\delta_f = |f(x_1,x_2,...,x_n) - f(\bar{x}_1,\bar{x}_2,...,\bar{x}_n)| \approx \sum_{i=1}^{n} \left|\frac{\partial f}{\partial x_i}(x_1,x_2,...,x_n)\right| \delta_{x_i} \tag{2.126}$$

Uma matriz de números de condicionamento também poderia ser obtida como em [5].

Exemplo 2.8.1. Considere o problema de calcular $\sqrt{x}$ em $x = 2$. Se usarmos $x^* = 1{,}999$, quanto será o erro relativo na saída? O erro relativo na entrada é

$$\left|\frac{\Delta x}{x}\right| = \left|\frac{2 - 1{,}999}{2}\right| = 0{,}0005 \tag{2.127}$$

O número de condicionamento do problema calcular a raiz é

$$\kappa_f(x) := \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x\frac{1}{2\sqrt{x}}}{\sqrt{x}}\right| = \frac{1}{2} \tag{2.128}$$

Ou seja, os erros na entrada serão diminuídos pela metade. De fato, usando $y = \sqrt{2} = 1{,}4142136...$ e $y^* = \sqrt{1{,}999} = 1{,}41386...$, obtemos

$$\frac{\Delta y}{y} = \frac{\sqrt{2} - \sqrt{1{,}999}}{\sqrt{2}} \approx 0{,}000250031... \tag{2.129}$$

Exemplo 2.8.2. Considere a função $f(x) = \frac{10}{1-x^2}$ e $x^* = 0{,}9995$ com um erro absoluto na entrada de 0,0001.

Calculando $y^* = f(x^*)$ temos

$$y^* = \frac{10}{1 - (0{,}9995)^2} \approx 10002{,}500625157739705173 \tag{2.130}$$

Mas qual é a estimativa de erro nessa resposta? Quantos dígitos significativos temos nessa resposta?

Sabendo que $f'(x) = 20x/(1 - x^2)^2$, o número de condicionamento é

$$\kappa_f(x) := \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{2x^2}{1 - x^2}\right| \tag{2.131}$$

o que nos fornece para $x^* = 0{,}9995$,

$$\kappa_f(0{,}9995) \approx 1998{,}5 \tag{2.132}$$

Como o erro relativo na entrada é

$$\left|\frac{\Delta x}{x}\right| = \left|\frac{0{,}0001}{0{,}9995}\right| \approx 0{,}00010005... \tag{2.133}$$

temos que o erro na saída será aproximadamente

$$\left|\frac{\Delta y}{y}\right| \approx \kappa_f(x)\left|\frac{\Delta x}{x}\right| \approx 1998{,}5 \times 0{,}00010005... \approx 0{,}1999 \tag{2.134}$$

ou seja um erro relativo de aproximadamente 19,99%.

Note que se usarmos $x_1 = 0{,}9994$ e $x_2 = 0{,}9996$ (ambos no intervalo do erro absoluto da entrada) encontramos

$$y_1^* \approx 8335{,}83 \tag{2.135}$$
$$y_2^* \approx 12520{,}50 \tag{2.136}$$

confirmando a estimativa de 19,99%.

Exemplo 2.8.3. Seja $f(x) = x\exp(x)$. Calcule o erro absoluto ao calcular $f(x)$ sabendo que $x = 2 \pm 0{,}05$.

Solução. Temos que $x \approx 2$ com erro absoluto de $\delta_x = 0{,}05$. Neste caso, calculamos δ_f, isto é, o erro absoluto ao calcular $f(x)$, por:

$$\delta_f = |f'(x)|\delta_x. \tag{2.137}$$

Como $f'(x) = (1+x)e^x$, temos:

$$\delta_f = |(1+x)e^x| \cdot \delta_x \tag{2.138}$$
$$= |3e^2| \cdot 0{,}05 = 1{,}1084. \tag{2.139}$$

Portanto, o erro absoluto ao calcular $f(x)$ quando $x = 2 \pm 0{,}05$ é de 1,1084. $\Diamond$

Exemplo 2.8.4. Calcule o erro relativo ao medir $f(x,y) = \frac{x^2+1}{x^2}e^{2y}$ sabendo que $x \approx 3$ é conhecido com 10% de erro e $y \approx 2$ é conhecido com 3% de erro.

Solução. Calculamos as derivadas parciais de f:

$$\frac{\partial f}{\partial x} = \frac{2x^3 - (2x^3+2x)}{x^4}e^{2y} = -\frac{2e^{2y}}{x^3} \tag{2.140}$$

e

$$\frac{\partial f}{\partial y} = 2\frac{x^2+1}{x^2}e^{2y} \tag{2.141}$$

Calculamos o erro absoluto em termos do erro relativo:

$$\frac{\delta_x}{|x|} = 0{,}1 \Rightarrow \delta_x = 3 \cdot 0{,}1 = 0{,}3 \tag{2.142}$$

$$\frac{\delta_y}{|y|} = 0{,}03 \Rightarrow \delta_y = 2 \cdot 0{,}03 = 0{,}06 \tag{2.143}$$

Aplicando a expressão para estimar o erro em f temos

$$\delta_f = \left|\frac{\partial f}{\partial x}\right| \delta_x + \left|\frac{\partial f}{\partial y}\right| \delta_y \tag{2.144}$$

$$= \frac{2e^4}{27} \cdot 0{,}3 + 2\frac{9+1}{9} e^4 \cdot 0{,}06 = 8{,}493045557 \tag{2.145}$$

Portanto, o erro relativo ao calcular f é estimado por

$$\frac{\delta f}{|f|} = \frac{8{,}493045557}{\frac{9+1}{9} e^4} = 14\% \tag{2.146}$$

$\Diamond$

Exemplo 2.8.5. No exemplo anterior, reduza o erro relativo em x pela metade e calcule o erro relativo em f. Depois, repita o processo reduzindo o erro relativo em y pela metade.

Solução. Na primeira situação temos $x = 3$ com erro relativo de 5% e $\delta_x = 0{,}05 \cdot 3 = 0{,}15$. Calculamos $\delta_f = 7{,}886399450$ e o erro relativo em f de 13%. Na segunda situação, temos $y = 2$ com erro de 1,5% e $\delta_y = 2 \cdot 0{,}015 = 0{,}03$. Calculamos $\delta_f = 4{,}853168892$ e o erro relativo em f de 8%. Observe que mesma o erro relativo em x sendo maior, o erro em y é mais significante na função. $\Diamond$

Exemplo 2.8.6. Considere um triângulo retângulo onde a hipotenusa e um dos catetos são conhecidos a menos de um erro: hipotenusa $a = 3 \pm 0{,}01$ metros e cateto $b = 2 \pm 0{,}01$ metros. Calcule o erro absoluto ao calcular a área dessa triângulo.

Solução. Primeiro vamos encontrar a expressão para a área em função da hipotenusa a e um cateto b. A tamanho de segundo cateto c é dado pelo teorema de Pitágoras, $a^2 = b^2 + c^2$, ou seja, $c = \sqrt{a^2 - b^2}$. Portanto a área é

$$A = \frac{bc}{2} = \frac{b\sqrt{a^2 - b^2}}{2}. \tag{2.147}$$

Agora calculamos as derivadas

$$\frac{\partial A}{\partial a} = \frac{ab}{2\sqrt{a^2 - b^2}}, \tag{2.148}$$

$$\frac{\partial A}{\partial b} = \frac{\sqrt{a^2 - b^2}}{2} - \frac{b^2}{2\sqrt{a^2 - b^2}}, \tag{2.149}$$

e substituindo na estimativa para o erro δ_A em termos de $\delta_a = 0{,}01$ e $\delta_b = 0{,}01$:

$$\delta_A \approx \left|\frac{\partial A}{\partial a}\right| \delta_a + \left|\frac{\partial A}{\partial b}\right| \delta_b \tag{2.150}$$

$$\approx \frac{3\sqrt{5}}{5} \cdot 0{,}01 + \frac{\sqrt{5}}{10} \cdot 0{,}01 = 0{,}01565247584 \tag{2.151}$$

Em termos do erro relativo temos erro na hipotenusa de $\frac{0{,}01}{3} \approx 0{,}333\%$, erro no cateto de $\frac{0{,}01}{2} = 0{,}5\%$ e erro na área de

$$\frac{0{,}01565247584}{\frac{2\sqrt{3^2-2^2}}{2}} = 0{,}7\% \tag{2.152}$$

$\Diamond$

Exercícios

E 2.8.1. Considere que a variável $x \approx 2$ é conhecida com um erro relativo de 1% e a variável $y \approx 10$ com um erro relativo de 10%. Calcule o erro relativo associado a z quando:

$$z = \frac{y^4}{1+y^4} e^x. \tag{2.153}$$

Suponha que você precise conhecer o valor de z com um erro de 0,5%. Você propõe uma melhoria na medição da variável x ou y? Explique.

E 2.8.2. A corrente I em ampéres e a tensão V em volts em uma lâmpada se relacionam conforme a seguinte expressão:

$$I = \left(\frac{V}{V_0}\right)^{\alpha}, \tag{2.154}$$

onde α é um número entre 0 e 1 e V_0 é tensão nominal em volts. Sabendo que $V_0 = 220 \pm 3\%$ e $\alpha = -0{,}8 \pm 4\%$, calcule a corrente e o erro relativo associado quando a tensão vale $220 \pm 1\%$.
Obs:. Este problema pode ser resolvido de duas formas distintas: usando a expressão aproximada para a propagação de erro e inspecionando os valores máximos e mínimos que a expressão pode assumir. Pratique os dois métodos.**Dica:** lembre que $x^\alpha = e^{\alpha \ln(x)}$

2.9 Exemplos selecionados de cancelamento catastrófico

Exemplo 2.9.1. Considere o seguinte processo iterativo:

$$x^{(1)} = \frac{1}{3} \tag{2.155}$$

$$x^{(n+1)} = 4x^{(n)} - 1, \quad n = 1, 2, \ldots \tag{2.156}$$

Observe que $x^{(1)} = \frac{1}{3}$, $x^{(2)} = 4 \cdot \frac{1}{3} - 1 = \frac{1}{3}$, $x^{(3)} = \frac{1}{3}$, ou seja, temos uma sequência constante igual a $\frac{1}{3}$. No entanto, ao calcularmos no computador, usando o sistema de numeração `double`, a sequência obtida não é constante e, de fato, diverge.

Faça o teste no `GNU Octave`, colocando:

```
>> x = 1/3
```

e itere algumas vezes a linha de comando:

```
>> x = 4*x-1
```

Para compreender o que acontece, devemos levar em consideração que o número $\frac{1}{3} = 0{,}\overline{3}$ possui um representação infinita tanto na base decimal quanto na base binária. Logo, sua representação de máquina inclui um erro de arredondamento. Seja ϵ a diferença entre o valor exato de $\frac{1}{3}$ e sua representação de máquina, isto é, $\tilde{x}^{(1)} = \frac{1}{3} + \epsilon$. A sequência efetivamente calculada no computador é:

$$\tilde{x}^{(1)} = \frac{1}{3} + \epsilon \tag{2.157}$$

$$\tilde{x}^{(2)} = 4x^{(1)} - 1 = 4\left(\frac{1}{3} + \epsilon\right) - 1 = \frac{1}{3} + 4\epsilon \tag{2.158}$$

$$\tilde{x}^{(3)} = 4x^{(2)} - 1 = 4\left(\frac{1}{3} + 4\epsilon\right) - 1 = \frac{1}{3} + 4^2\epsilon \tag{2.159}$$

$$\vdots \tag{2.160}$$

$$\tilde{x}^{(n)} = \frac{1}{3} + 4^{(n-1)}\epsilon \tag{2.161}$$

Portanto o limite da sequência diverge,

$$\lim_{x \to \infty} |\tilde{x}^{(n)}| = \infty \tag{2.162}$$

Qual o número de condicionamento desse problema?

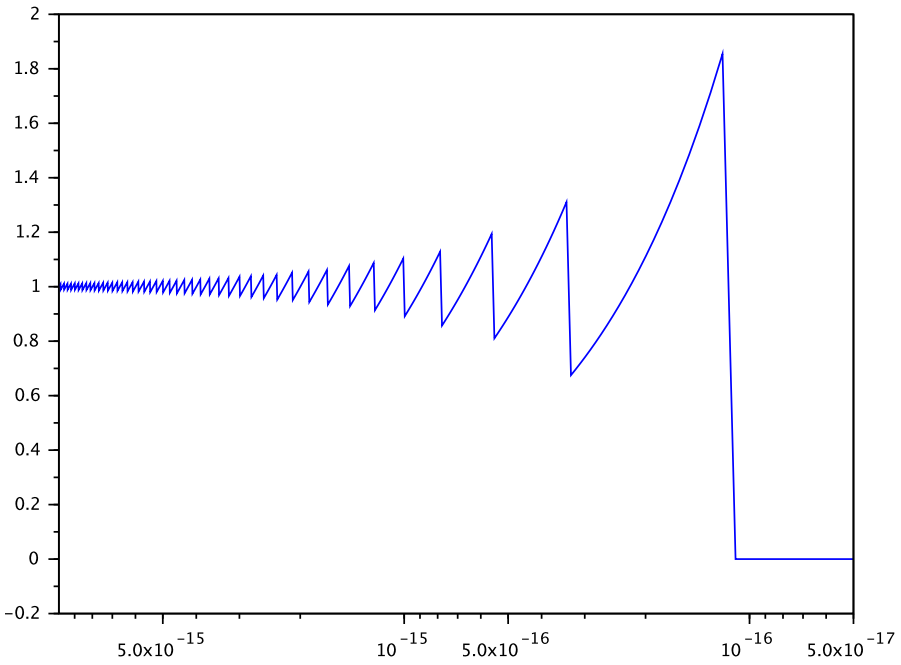


Figura 2.1: Gráfico na função do Exemplo 2.9.2.

Exemplo 2.9.2. Observe a seguinte identidade

$$f(x) = \frac{(1+x)-1}{x} = 1 \tag{2.163}$$

Calcule o valor da expressão à esquerda para $x = 10^{-12}$, $x = 10^{-13}$, $x = 10^{-14}$, $x = 10^{-15}$, $x = 10^{-16}$ e $x = 10^{-17}$. Observe que quando x se aproxima do ϵ de máquina a expressão perde o significado. Veja a Figura 2.1 com o gráfico de $f(x)$ em escala logarítmica.

Exemplo 2.9.3. Neste exemplo, estamos interessados em compreender mais detalhadamente o comportamento da expressão

$$\left(1+\frac{1}{n}\right)^n \tag{2.164}$$

quando n é um número grande ao computá-la em sistemas de numeral de ponto flutuante com acurácia finita. Um resultado bem conhecido do cálculo nos diz que o limite de (2.164) quando n tende a infinito é o número de Euler:

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = e = 2{,}718281828459... \tag{2.165}$$

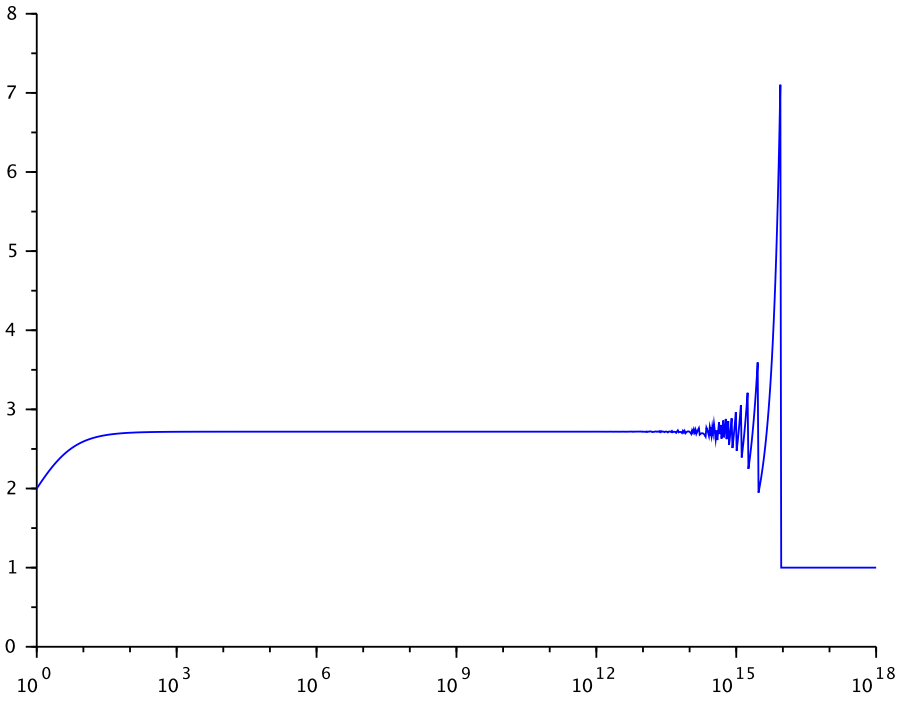


Figura 2.2: Gráfico de $\left(1+\frac{1}{n}\right)^n$ em função de n em escala linear-logarítmica variando de 10^0 até 10^{18}. Veja o Exemplo 2.9.3.

Sabemos também que a sequência produzida por (2.164) é crescente, isto é:

$$\left(1+\frac{1}{1}\right)^1 < \left(1+\frac{1}{2}\right)^2 < \left(1+\frac{1}{3}\right)^3 < \cdots \tag{2.166}$$

No entanto, quando calculamos essa expressão no `Julia`, nos defrontamos com o seguinte resultado:

n	$\left(1+\frac{1}{n}\right)^n$		n	$\left(1+\frac{1}{n}\right)^n$
1	2,0000000000000		10^2	2,7048138294215
2	2,2500000000000		10^4	2,7181459268249
3	2,3703703703704		10^6	2,7182804690957
4	2,4414062500000		10^8	2,7182817983391
5	2,4883200000000		10^{10}	2,7182820532348
6	2,5216263717421		10^{12}	2,7185234960372
7	2,5464996970407		10^{14}	2,7161100340870
8	2,5657845139503		10^{16}	1,0000000000000
9	2,5811747917132		10^{18}	1,0000000000000
10	2,5937424601000		10^{20}	1,0000000000000

Podemos resumir esses dados no gráfico de $\left(1+\frac{1}{n}\right)^n$ em função de n, veja a Figura 2.9.

Observe que quando n se torna grande, da ordem de 10^{15}, o gráfico da função deixa de ser crescente e apresenta oscilações. Observe também que a expressão se torna identicamente igual a 1 depois de um certo limiar. Tais fenômenos não são intrínsecos da função $f(n) = (1+1/n)^n$, mas **oriundas de erros de arredondamento**, isto é, são resultados numéricos espúrios. A fim de pôr o comportamento numérico de tal expressão, apresentamos abaixo o gráfico da mesma função, porém restrito à região entre 10^{14} e 10^{16}.

Para compreendermos melhor por que existe um limiar N que, quando atingido torna a expressão do exemplo acima identicamente igual a 1, observamos a sequência de operações realizadas pelo computador:

$$n \;\to\; 1/n \;\to\; 1+1/n \;\to\; (1+1/n)^n \tag{2.167}$$

Devido ao limite de precisão da representação de números em ponto flutuante, existe um menor número representável que é maior do que 1. Este número é 1+`eps`, onde `eps` é chamado de **épsilon de máquina** e é o menor número que somado a 1 produz um resultado superior a 1 no sistema de numeração usado. O épsilon de máquina no sistema de numeração **double** vale aproximadamente $2{,}22 \times 10^{-16}$.

No `GNU Octave`, o épsilon de máquina é a constante `eps`. Observe que:

```
>> printf('%1.25f\n', 1+eps)
1.0000000000000002220446049
```

Quando somamos a 1 um número positivo inferior ao épsilon de máquina, obtemos o número 1. Dessa forma, o resultado obtido pela operação de ponto flutuante $1 + n$ para $0 < n < 2{,}22 \times 10^{-16}$ é 1.

Portanto, quando realizamos a sequência de operações dada em (2.167), toda informação contida no número n é perdida na soma com 1 quando $1/n$ é menor que o épsilon de máquina, o que ocorre quando $n > 5 \times 10^{15}$. Assim, $(1 + 1/n)$ é aproximado para 1 e a última operação se resume a 1^n, o que é igual a 1 mesmo quando n é grande.

Um erro comum é acreditar que o perda de significância se deve ao fato de $1/n$ ser muito pequeno para ser representado e é aproximando para 0. Isto é falso, o sistema de ponto de flutuante permite representar números de magnitude muito inferior ao épsilon de máquina. O problema surge da limitação no tamanho da mantissa. Observe como a seguinte sequência de operações não perde significância para números positivos x muito menores que o épsilon de máquina:

$$n \;\rightarrow\; 1/n \;\rightarrow\; 1/(1/n) \tag{2.168}$$

compare o desempenho numérico desta sequência de operações para valores pequenos de n com o da seguinte sequência:

$$n \;\rightarrow\; 1+n \;\rightarrow\; (1+n)-1. \tag{2.169}$$

Finalmente, notamos que quando tentamos calcular $\left(1+\frac{1}{n}\right)^n$ para n grande, existe perda de significância no cálculo de $1 + 1/n$.

Para entendermos isso melhor, vejamos o que acontece no `GNU Octave` quando $n = 7 \times 10^{13}$:

```
>> format('long')
>> n=7e13
n =  70000000000000
>> 1/n
```

```
ans =    1.42857142857143e-14
>> y=1+1/n
y =  1.00000000000001
```

Observe a perda de informação ao deslocar a mantissa de $1/n$. Para evidenciar o fenômenos, observamos o que acontece quando tentamos recalcular n subtraindo 1 de $1 + 1/n$ e invertendo o resultado:

```
>> y-1
ans =    1.42108547152020e-14
>> 1/(y-1)
ans =  70368744177664
```

Exemplo 2.9.4 (Analogia da balança)**.** Observe a seguinte comparação interessante que pode ser feita para ilustrar os sistemas de numeração com ponto fixo e flutuante: o sistema de ponto fixo é como uma balança cujas marcas estão igualmente espaçadas; o sistema de ponto flutuante é como uma balança cuja distância entre as marcas é proporcional à massa medida. Assim, podemos ter uma balança de ponto fixo cujas marcas estão sempre distanciadas de 100g (100g, 200g, 300g, ..., 1kg, 1,1kg,...) e outra balança de ponto flutuante cujas marcas estão distanciadas sempre de aproximadamente um décimo do valor lido (100g, 110g, 121g, 133g, ..., 1kg, 1,1kg, 1,21kg, ...) A balança de ponto fixo apresenta uma resolução baixa para pequenas medidas, porém uma resolução alta para grandes medidas. A balança de ponto flutuante distribui a resolução de forma proporcional ao longo da escala.

Seguindo nesta analogia, o fenômeno de perda de significância pode ser interpretado como a seguir: imagine que você deseje obter o peso de um gato (aproximadamente 4kg). Dois processos estão disponíveis: colocar o gato diretamente na balança ou medir seu peso com o gato e, depois, sem o gato. Na balança de ponto flutuante, a incerteza associada à medida do peso do gato (sozinho) é aproximadamente 10% de 4kg, isto é, 400g. Já a incerteza associada à medida da uma pessoa (aproximadamente 70kg) com o gato é de 10% do peso total, isto é, aproximadamente 7kg. Esta incerteza é da mesma ordem de grandeza da medida a ser realizada, tornado o processo impossível de ser realizado, já que teríamos uma incerteza da ordem de 14kg (devido à dupla medição) sobre uma grandeza de 4kg.

Exercícios resolvidos

ER 2.9.1. Deseja-se medir a concentração de dois diferentes oxidantes no ar. Três sensores eletroquímicos estão disponíveis para a medida e apresentam a seguintes respostas:

$$v_1 = 270[A] + 30[B], \quad v_2 = 140[A] + 20[B] \text{ e } v_3 = 15[A] + 200[B] \qquad (2.170)$$

as tensões v_1, v_2 e v_3 são dadas em mV e as concentrações em $milimol/l$.

a) Encontre uma expressão para os valores de $[A]$ e $[B]$ em termos de v_1 e v_2 e, depois, em termos de v_1 e v_3. Dica: Se $ad \neq bc$, então a matriz A dada por

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{2.171}$$

é inversível e sua inversa é dada por

$$A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}. \tag{2.172}$$

b) Sabendo que incerteza relativa associada às sensibilidades dos sensores 1 e 2 é de 2% e que a incerteza relativa associada às sensibilidades do sensor 3 é 10%, verifique a incerteza associada à medida feita com o par $1-2$ e o par $1-3$. Use $[A] = [B] = 10milimol/l$. Dica: Você deve diferenciar as grandezas $[A]$ e $[B]$ em relação aos valores das tensões.

Solução. Em ambos casos, temos a seguinte estrutura:

$$\begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} [A] \\ [B] \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{2.173}$$

De forma que

$$\begin{bmatrix} [A] \\ [B] \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix}^{-1} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \frac{1}{S_{11}S_{22} - S_{12}S_{21}} \begin{bmatrix} S_{22} & -S_{12} \\ -S_{21} & S_{11} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{2.174}$$

Portanto

$$[A] = \frac{S_{22}v_1 - S_{12}v_2}{S_{11}S_{22} - S_{12}S_{21}} \tag{2.175}$$

$$[B] = \frac{-S_{21}v_1 + S_{11}v_2}{S_{11}S_{22} - S_{12}S_{21}} \tag{2.176}$$

Usando derivação logarítmica, temos

$$\frac{1}{[A]}\frac{\partial[A]}{\partial S_{11}} = -\frac{S_{22}}{S_{11}S_{22}-S_{12}S_{21}} \tag{2.177}$$

$$\frac{1}{[A]}\frac{\partial[A]}{\partial S_{12}} = -\frac{v_2}{S_{22}v_1-S_{12}v_2}+\frac{S_{21}}{S_{11}S_{22}-S_{12}S_{21}} \tag{2.178}$$

$$= -\frac{[A]}{[B]}\cdot\frac{S_{22}}{S_{11}S_{22}-S_{12}S_{21}} \tag{2.179}$$

$$\frac{1}{[A]}\frac{\partial[A]}{\partial S_{21}} = \frac{S_{12}}{S_{11}S_{22}-S_{12}S_{21}} \tag{2.180}$$

$$\frac{1}{[A]}\frac{\partial[A]}{\partial S_{22}} = \frac{v_1}{S_{22}v_1-S_{12}v_2}-\frac{S_{11}}{S_{11}S_{22}-S_{12}S_{21}} \tag{2.181}$$

$$= \frac{[A]}{[B]}\cdot\frac{S_{12}}{S_{11}S_{22}-S_{12}S_{21}} \tag{2.182}$$

e

$$\frac{1}{[B]}\frac{\partial[B]}{\partial S_{11}} = \frac{v_2}{-S_{21}v_1+S_{11}v_2}-\frac{S_{22}}{S_{11}S_{22}-S_{12}S_{21}} \tag{2.183}$$

$$= \frac{[B]}{[A]}\frac{S_{21}}{S_{11}S_{22}-S_{12}S_{21}} \tag{2.184}$$

$$\frac{1}{[B]}\frac{\partial[B]}{\partial S_{12}} = \frac{S_{21}}{S_{11}S_{22}-S_{12}S_{21}} \tag{2.185}$$

$$\frac{1}{[B]}\frac{\partial[B]}{\partial S_{21}} = -\frac{v_1}{-S_{21}v_1+S_{11}v_2}+\frac{S_{21}}{S_{11}S_{22}-S_{12}S_{21}} \tag{2.186}$$

$$= -\frac{[B]}{[A]}\frac{S_{11}}{S_{11}S_{22}-S_{12}S_{21}} \tag{2.187}$$

$$\frac{1}{[B]}\frac{\partial[B]}{\partial S_{22}} = -\frac{S_{11}}{S_{11}S_{22}-S_{12}S_{21}} \tag{2.188}$$

$$\tag{2.189}$$

E o erro associado às medidas pode ser aproximado por

$$\frac{1}{[A]}\delta_{[A]} = \left|\frac{1}{[A]}\frac{\partial[A]}{\partial S_{11}}\right|\delta_{S_{11}}+\left|\frac{1}{[A]}\frac{\partial[A]}{\partial S_{12}}\right|\delta_{S_{12}} \tag{2.190}$$

$$+ \left|\frac{1}{[A]}\frac{\partial[A]}{\partial S_{21}}\right|\delta_{S_{21}}+\left|\frac{1}{[A]}\frac{\partial[A]}{\partial S_{22}}\right|\delta_{S_{22}} \tag{2.191}$$

$$= \frac{1}{|\det S|}\left[S_{22}\delta_{S_{11}}+\frac{[A]}{[B]}S_{22}\delta_{S_{12}}+S_{12}\delta_{S_{21}}+\frac{[A]}{[B]}S_{12}\delta_{S_{22}}\right] \tag{2.192}$$

Analogamente, temos:

$$\frac{1}{[B]}\delta_{[B]} = \frac{1}{|\det S|}\left[\frac{[B]}{[A]}S_{21}\delta_{S_{11}} + S_{21}\delta_{S_{11}} + \frac{[B]}{[A]}S_{11}\delta_{S_{21}} + S_{11}\delta_{S_{22}}\right] \quad (2.193)$$

onde não se indicou $|S_{ij}|$ nem $|[.]|$ pois são todos positivos.

Fazemos agora a aplicação numérica:

Caso do par 1-2:

$$\det S = \begin{vmatrix} 270 & 30 \\ 140 & 20 \end{vmatrix} = 1200 \quad (2.194)$$

$$\begin{aligned} \frac{1}{[A]}\delta_{[A]} &= \frac{1}{1200}\left[20 \times 270 \times 2\% + 20 \times 30 \times 2\%\right. && (2.195)\\ &\qquad \left. + 30 \times 140 \times 2\% + 30 \times 20 \times 2\%\right] && (2.196)\\ &= \frac{216}{1200} = 0.18 = 18\% && (2.197)\\ \frac{1}{[B]}\delta_{[B]} &= \frac{1}{1200}\left[140 \times 270 \times 2\% + 140 \times 30 \times 2\%\right. && (2.198)\\ &\qquad \left. + 270 \times 140 \times 2\% + 270 \times 20 \times 2\%\right] && (2.199)\\ &= \frac{426}{1200} = 0.355 = 35.5\% && (2.200) \end{aligned}$$

Caso do par 1-3:

$$\det S = \begin{vmatrix} 270 & 30 \\ 15 & 200 \end{vmatrix} = 53550 \quad (2.201)$$

$$\begin{aligned} \frac{1}{[A]}\delta_{[A]} &= \frac{1}{53550}\left[200 \times 270 \times 2\% + 200 \times 30 \times 2\%\right. && (2.202)\\ &\qquad \left. + 30 \times 15 \times 10\% + 30 \times 200 \times 10\%\right] && (2.203)\\ &= \frac{1804{,}6}{52550} \approx 0.0337 = 3.37\% && (2.204)\\ \frac{1}{[B]}\delta_{[B]} &= \frac{1}{53550}\left[15 \times 270 \times 2\% + 15 \times 30 \times 2\%\right. && (2.205)\\ &\qquad \left. + 270 \times 15 \times 10\% + 270 \times 200 \times 10\%\right] && (2.206)\\ &= \frac{5895}{53550} \approx 0.11 = 11\% && (2.207) \end{aligned}$$

Conclusão, apesar de o sensor 3 apresentar uma incerteza cinco vezes maior na sensibilidade, a escolha do sensor 3 para fazer par ao sensor 1 parece mais adequada. ◊

Exercícios

E 2.9.1. Considere as expressões:

$$\frac{\exp(1/\mu)}{1+\exp(1/\mu)} \tag{2.208}$$

e

$$\frac{1}{\exp(-1/\mu)+1} \tag{2.209}$$

com $\mu > 0$. Verifique que elas são idênticas como funções reais. Teste no computador cada uma delas para $\mu = 0{,}1$, $\mu = 0{,}01$ e $\mu = 0{,}001$. Qual dessas expressões é mais adequada quando μ é um número pequeno? Por quê?

E 2.9.2. Encontre expressões alternativas para calcular o valor das seguintes funções quando x é próximo de zero.

a) $f(x) = \frac{1-\cos(x)}{x^2}$

b) $g(x) = \sqrt{1+x} - 1$

c) $h(x) = \sqrt{x+10^6} - 10^3$

d) $i(x) = \sqrt{1+e^x} - \sqrt{2}$ Dica: Faça $y = e^x - 1$

E 2.9.3. Use uma identidade trigonométrica adequada para mostrar que:

$$\frac{1-\cos(x)}{x^2} = \frac{1}{2}\left(\frac{\operatorname{sen}(x/2)}{x/2}\right)^2. \tag{2.210}$$

Analise o desempenho destas duas expressões no computador quando x vale 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}, 10^{-9}, 10^{-200} e 0. Discuta o resultado. **Dica:** Para $|x| < 10^{-5}$, $f(x)$ pode ser aproximada por $1/2 - x^2/24$ com erro de truncamento inferior a 10^{-22}.

E 2.9.4. Reescreva as expressões:

$$\sqrt{e^{2x}+1} - e^x \qquad \text{e} \qquad \sqrt{e^{2x}+x^2} - e^x \tag{2.211}$$

de modo que seja possível calcular seus valores para $x = 100$ utilizando a aritmética de ponto flutuante ("Double") no computador.

E 2.9.5. Na teoria da relatividade restrita, a energia cinética de uma partícula e sua velocidade se relacionam pela seguinte fórmula:

$$E = mc^2 \left(\frac{1}{\sqrt{1-(v/c)^2}} - 1 \right), \tag{2.215}$$

onde E é a energia cinética da partícula, m é a massa de repouso, v o módulo da velocidade e c a velocidade da luz no vácuo dada por $c = 299792458m/s$. Considere que a massa de repouso $m = 9{,}10938291 \times 10^{-31}kg$ do elétron seja conhecida com erro relativo de 10^{-9}. Qual é o valor da energia e o erro relativo associado a essa grandeza quando $v = 0{,}1c$, $v = 0{,}5c$, $v = 0{,}99c$ e $v = 0{,}999c$ sendo que a incerteza relativa na medida da velocidade é 10^{-5}?

Apêndice A

Rápida introdução à linguagem Julia

Neste apêndice, abordaremos a linguagem computacional `Julia` que nos auxiliará no processo de obtenção de resultados de operações e computações utilizando o computador.

A.1 Sobre a linguagem Julia

A linguagem de programação Julia é uma linguagem moderna e poderosa que foi criada para atender aos requisitos da computação numérica de alto desempenho e científica. Seus criadores quiseram reunir na linguagem as principais (melhores) características de outras linguagens:

- Ruby;
- matlab;
- R;
- Julia;
- C.

Julia é uma linguagem de programação de alto nível, interpretada e multi-paradigma.

Para mais informações, consulte:

- Página oficial da linguagem Julia: `https://julialang.org/`
- Manual Julia: `https://docs.julialang.org/en/v1/manual/getting-started/`

- GitHub do Professor Daniel: https://github.com/Daniel-C-Fernandes/Numerico/blob/main/Mini-curso-Julia.ipynb
- Julia for Data science: https://juliadatascience.io/pt/
- Julia Academy: https://juliaacademy.com/
- GitHub Julia: https://github.com/JuliaLang/julia
- Introdução: https://julia.vento.eng.br/
- Curso Julia USP: https://edisciplinas.usp.br/pluginfile.php/7879038/mod_resource/content/4/Tutorial%20para%20a%20Linguagem%20Julia.pdf
- Tutoriais Julia: https://julialang.org/learning/tutorials/

Para saber mais: https://www.youtube.com/watch?v=Xzdg9czOxD8&t=29s

A.1.1 Características Principais

- **Tipagem Dinâmica:** Suas variáveis podem receber qualquer tipo de dado, e sua sintaxe se aproxima mais da linguagem humana do que da linguagem de máquina.
- **Multiparadigma:** Suporta diversos paradigmas de programação, como orientação a objetos e programação funcional.
- **Alto Nível:** Possui uma sintaxe expressiva e amigável.
- **Gratuita e Open Source:** Julia é distribuída sob a licença MIT.
- **Suporte a Unicode e UTF-8:** Permite o uso de símbolos matemáticos durante a escrita de programas. Gerenciador de Pacotes Prático: Facilita a instalação e atualização de pacotes.

Aplicações:

- **Data Science:** Julia é usada para análise de dados e descoberta de conhecimento a partir de grandes conjuntos de dados. Machine Learning: Possui pacotes poderosos para criação de modelos de aprendizado de máquina.
- **Computação Científica:** Ideal para construir modelos matemáticos e soluções numéricas.

- **Desenvolvimento Geral:** Pode ser aplicada no desenvolvimento de aplicações web, desktop e outras áreas. Em resumo, Julia é uma linguagem versátil que combina alta performance com uma sintaxe amigável, tornando-a uma escolha popular entre cientistas de dados, desenvolvedores e entusiastas da programação.

Visão geral sobre linguagens de programação: Qual a melhor linguagem de programação? Veja https://www.youtube.com/watch?v=DjUB-yVWT2A.

Para iniciantes, recomendamos o curso EAD gratuito no site Goggo dot jl:

https://www.youtube.com/watch?v=0oChN11wf_4

A.1.2 Instalação e execução

Para versões `Linux Ubuntu` ou `Debian`, basta utilizar o comando a seguir e esperar a instalação:

```
> sudo apt install julia -y
```

Para versões `Windows`, utilize o Prompt de comando `PowerShell` para inserir o comando de instalação a seguir:

```
> winget install julia -s msstore
```

Execute o modo interativo da linguagem `Julia` simplesmente digitando o nome da linguagem seguida de Enter:

```
> julia
```

Dessa forma, abre-se o interpretador interativo da linguagem `Julia`. Pode-se então, como primeira interação, realizar uma operação matemática:

```
julia> 2 + 3
5
```

No site oficial da linguagem Julia estão disponíveis para *download* os interpretadores para os principais sistemas operacionais, `Linux`, `Mac OS` e `Windows`.

Além disso, no GitHub do professor Daniel, há um tutorial de instalação para a versão Windows do intepretador de linguagem `Julia` juntamente com o ambiente `Jupyter Notebook` e uma introdução à linguagem de programação com exemplos.

A.1.3 Usando Julia

O uso da linguagem `Julia` pode ser feito de três formas básicas:

- usando um **console Julia** de modo interativo;

- executando um código `codigo.jl` no console `Julia`;
- executando um código `Julia codigo.jl` diretamente no terminal;

Execução no terminal de um código salvo em .jl

Para se executar um código diretamente no terminal de comando do sistema operacional, basta escrevermos o código que desejamos em um arquivo texto de extensão **.jl**.

Dessa forma, por exemplo, salvaremos o seguinte código num arquivo chamado **ola.jl**:

```
1  println("Hello world!!")
```

Após o salvamento, estando o prompt de comando no mesmo diretório do arquivo salvo, basta executarmos o seguinte comando, obtendo-se a resposta que se segue:

```
1 > julia ola.jl
2 Hello world!!
```

Utilização do Console interativo Julia

Para executarmos qualquer comando no console interativo da linguagem `Julia`, iniciaremos o console interativo da seguinte forma:

```
1 > julia
```

Estando o modo interativo rodando no prompt de comando, basta inserirmos o comando desejado e verificar o resultado da saída do comando registrado na linha seguinte do prompt:

```
1 julia> println("Hello world!!")
2 Hello world!!
```

Execução no console de um código salvo em .jl

Para executarmos qualquer um código salvo com extensão **.jl** no console interativo da linguagem `Julia`, iniciaremos o console interativo da seguinte forma:

```
1 > julia
```

Estando o modo interativo rodando no prompt de comando, basta inserirmos o seguinte comando e verificar o resultado da saída do arquivo registrado na linha seguinte do prompt (observe que o arquivo deve estar localizado em ...):

```
julia> include ola.jl
Hello world!!
```

A.2 Elementos da linguagem

`Julia` é uma linguagem de alto nível de tipagem dinâmica, ou seja, uma variável é criada quando um valor é atribuído a ela, não sendo necessário especificar explicitamente cada tipo de variável, `Julia` vai inferir o tipo de cada variável por você. Porém, também é possível especificar a declaração da variável a ser criada, se for desejável.

A.2.1 Variáveis

Variáveis são valores armazenados pelo computador atrelados a um nome específico, para seja possível recuperar ou alterar seu valor posteriormente.

Alguns tipos de variáveis em `Julia`:

- Números inteiros: Int64
- Números reais: Float64
- Matrizes inteiras: Matrix{Int64}
- Matrizes reais: Matrix{Float64}
- Booleanas: Bool
- Strings: String

Por padrão, números são armazenados usando 64 bits, sendo possível aumentar ou reduzir a precisão, utilizando os tipos Int8 ou Int128, por exemplo.

Criamos novas variáveis escrevendo o nome da variável à esquerda e seu valor à direita, e no meio usamos o operador de atribuição =.

Vejamos alguns exemplos:

```
julia> x = 1
1
julia> y = x * 2.0
2.0
```

variaveis com emogi

a variável `x` recebe o valor `int` 1 e, logo após, na segunda linha de comando, a variável `y` recebe o valor `double` 2. Observamos que o símbolo `=` significa o operador de atribuição não o de igualdade. O operador lógico de igualdade no `Julia` é `==`. Veja os seguintes comandos:

```
julia> print(x, "-", y)
1-2.0

julia> typeof(x), typeof(y)
(Int64, Float64)
```

Comentários e continuação de linha de comando são usados como no seguinte exemplo:

```
julia> # Isto é um comentário

julia> x = 1
1

julia> print(x)
1
```

A.3 Repositórios

Tartaruga

A.4 Estruturas de ramificação e repetição

A linguagem `Julia` contém estruturas de repetição e ramificação padrões de linguagens estruturadas.

A.4.1 A instrução de ramificação “if”

A instrução “if” permite executar um pedaço do código somente se uma dada condição for satisfeita.

Exemplo A.4.1. Veja o seguinte código Julia:

```
#!/usr/bin/env Julia
# -*- coding:  utf-8 -*-

```

```
i = 2
if (i == 1):
    print("Olá!")
elif (i == 2):
    print("Hallo!")
elif (i == 3):
    print("Hello!")
else:
    print("Ça Va!")
```

Qual é a saída apresentada pelo código? Por quê?

Observamos que, em `Julia`, a identação é obrigatória, pois é ela que defini o escopo da instrução.

A.4.2 A instrução de repetição "for"

A instrução `for` permite que um pedaço de código seja executado repetidamente.

Exemplo A.4.2. Veja o seguinte código:

```
for i in range(6):
    print(i)
```

Qual é a saída deste código? Por quê?

Exemplo A.4.3. Veja o seguinte código:

```
import numpy as np
for i in np.arange(1,8,2):
    print(i)
```

Qual é a saída deste código? Por quê?

Exemplo A.4.4. Veja o seguinte código:

```
for i in np.arange(10,0,-3):
  print(i)
```

O que é mostrado no console do Julia?

Exemplo A.4.5. Veja o seguinte código:

```
import numpy as np
for i in np.arange(10,1,-3):
    print(i)
```

O que é mostrado no console do Julia?

A.4.3 A instrução de repetição “while”

A instrução `while` permite que um pedaço de código seja executado repetidamente até que uma dada condição seja satisfeita.

Exemplo A.4.6. Veja o seguinte código Julia:

```
s = 0
i = 1
while (i <= 10):
    s = s + i
    i = i + 1
```

Qual é o valor de `s` ao final da execução? Por quê?

A.5 Funções

Além das muitas funções disponíveis em `Julia` (e os tantos muitos pacotes livres disponíveis), podemos definir nossas próprias funções. Para tanto, existe a instrução `def`. Veja os seguintes exemplos:

Exemplo A.5.1. O seguinte código:

```
def f(x):
    return x + np.sin(x)
```

define a função $f(x) = x + \text{sen } x$.

Observe que $f(\pi) = \pi$. Confirme isso computando:

```
>>> f(np.pi)
```

Exemplo A.5.2. O seguinte código em `Julia`:

```
def h(x,y):
    if (x < y):
        return y - x
    else:
        return x - y
```

define a função:

$$h(x,y) = \begin{cases} y - x & , x < y \\ x - y & , x \geq y \end{cases} \tag{A.1}$$

Exemplo A.5.3. O seguinte código:

```
def J(x):
    y = np.zeros((2,2))
    y[0,0] = 2*x[0]
    y[0,1] = 2*x[1]

    y[1,0] = -x[1]*np.sin(x[0]*x[1])
    y[1,1] = -x[0]*np.sin(x[0]*x[1])

    return y
```

define a matriz jacobiana $J(x_1,x_2) := \frac{(f_1,f_2)}{(x_1,x_2)}$ da função:

$$\boldsymbol{f}(x_1,x_2) = (x_1^2 + x_2^2,\ \cos(x_1 x_2)). \tag{A.2}$$

A.5.1 Operações matemáticas elementares

Em `Julia`, os operadores matemáticos elementares são os seguintes:

```
+ # adição
- # subtração
* # multiplicação
/ # divisão de a por b
\ # divisão de b por a
% # Resto da divisão euclidiana
÷ # Quociente inteiro da divisão euclidiana (alt + 246)
(ou \div + tab)
^ #potenciação
// # Frações
exp() #potenciação de base e
Log() #Logarítimo Neperiano
Log10() #Logarítimo de base 10
```

A.5.2 Funções e constantes elementares

Várias funções e constantes elementares estão disponíveis no pacote módulo Julia math. Por exemplo:

```
julia> π = pi (\pi + tab)
π = 3.1415926535897...

julia> cos(π)
-1.0

```

```
julia> exp(1)
2.718281828459045

julia> log(exp(1))
1.0
```

Observamos que `log` é a função logaritmo natural, isto é, $f(x) = \ln(x)$, enquanto que a implementação Julia de $f(x) = \log(x)$ é:

```
julia> log10(10)
1.0
```

Veja mais na documentação Julia.

A.5.3 Operadores lógicos

Em `Julia`, o valor lógico verdadeiro é escrito como `True` e o valor lógico falso como `False`. Temos os seguintes operadores lógicos disponíveis:

```
&& # e lógico
|| # ou lógico
! # negação
== # igualdade
!= # diferente
< # menor que
> # maior que
<= # menor ou igual que
>= # maior ou igual que
```

Veja mais em https://acervolima.com/operadores-em-julia/.

A.6 Matrizes

Em `Julia`, temos um ótimo suporte para computação científica com o pacote numpy. Uma matriz $A = [a_{i,j}]_{i,j=1}^{m,n}$ em Julia é definida usando-se a seguinte sintaxe:

```
julia> A = [
          a11 a12 ... a1n,
          a21 a22 ... a2n,
          ⋮
          am1 am2 ... amn
          ]
```

Exemplo A.6.1. Defina a matriz:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \tag{A.3}$$

Solução. Em `Julia`, digitamos:

```
julia> A = [1 2 3,
            4 5 6]

julia> print(A)
2×3 Matrix{Int64}:
1 2 3
4 5 6
```

◊

A seguinte lista contém uma série de funções que geram matrizes particulares:

```
eye # matriz identidade
linspace # vetor de elementos linearmente espaçados
ones # matriz cheia de uns
zeros # matriz nula
```

A.6.1 Obtendo dados de uma matriz

A função `numpy.shape` retorna o tamanho de uma matriz, por exemplo:

```
>>> A = np.ones((3,2))
>>> print(A)
[[ 1. 1.]
 [ 1. 1.]
 [ 1. 1.]]
>>> nl, nc = np.shape(A)
>>> print(nl,nc)
(3, 2)
```

informando que a matriz `A` tem três linhas e duas colunas.

Existem vários métodos para acessar os elementos de uma matriz dada `A`:

- a matriz inteira acessa-se com a sintaxe:

```
A
```

- o elemento da i-ésima linha e j-ésima coluna acessa-se usando a sintaxe:

```
A[i,j]
```

- o bloco formado pelas linhas i_1, i_2 e pelas colunas j_1, j_2 obtém-se usando a sintaxe:

```
A[i1:i2, j1:j2]
```

Exemplo A.6.2. Veja as seguintes linhas de comando:

```
>>> from numpy import random
>>> A = np.random.random((3,4))
>>> A
array([[ 0.39235668, 0.30287204, 0.24379253, 0.98866709],
       [ 0.72049734, 0.99300252, 0.14232844, 0.25604346],
       [ 0.61553036, 0.80615392, 0.22418474, 0.13685148]])
>>> A[2,3]
0.13685147547025989
>>> A[1:3,1:4]
array([[ 0.99300252, 0.14232844, 0.25604346],
       [ 0.80615392, 0.22418474, 0.13685148]])
```

Definida uma matriz A em Julia, as seguintes sintaxes são bastante úteis:

```
A[:,:] toda a matriz
A[i:j,k] os elementos das linhas i até j (exclusive) da k-ésima coluna
A[i,j:k] os elementos da i-ésima linha das colunas j até k (exclusive)
A[i,:] a i-ésima linha da matriz
A[:,j] a j-ésima coluna da matriz
```

Atenção, os índices em `Julia` iniciam-se em 0. Assim, o comando `A[1:3,1:4]` retorna o bloco da matriz A compreendido da segunda à terceira linha e da segunda a quarta coluna desta matriz.

Exemplo A.6.3. Veja as seguintes linhas de comando:

```
>>> B = np.random.random((4,4))
>>> B
array([[ 0.94313432, 0.72650883, 0.55487089, 0.18753526],
       [ 0.02094937, 0.45726099, 0.51925464, 0.8535878 ],
       [ 0.75948469, 0.95362926, 0.77942318, 0.06464183],
       [ 0.91243198, 0.22775889, 0.04061536, 0.14908227]])
>>> aux = np.copy(B[:,2])
>>> B[:,2] = np.copy(B[:,3])
```

```
>>> B[:,3] = np.copy(aux)
>>> B
array([[ 0.94313432, 0.72650883, 0.18753526, 0.55487089],
       [ 0.02094937, 0.45726099, 0.8535878 , 0.51925464],
       [ 0.75948469, 0.95362926, 0.06464183, 0.77942318],
       [ 0.91243198, 0.22775889, 0.14908227, 0.04061536]])
```

A.6.2 Operações matriciais e elemento-a-elemento

Em `Julia` com `numpy`, o operador `*` opera elemento a elemento. Por exemplo:

```
>>> A = np.array([[1,2],[2,1]]); print(A)
[[1 2]
 [2 1]]
>>> B = np.array([[2,1],[2,1]]); print(B)
[[2 1]
 [2 1]]
>>> print(A*B)
[[2 2]
 [4 1]]
```

A multiplicação matricial obtemos com:

```
>>> C = A.dot(B)
>>> print(C)
[[6 3]
 [6 3]]
```

Aqui, temos as sintaxes análogas entre operações elemento-a-elemento:

```
+ # adição
- # subtração
* # multiplicação
/ # divisão
^ # potenciação
```

Exemplo A.6.4. Veja as seguintes linhas de comando:

```
>>> A = np.ones((2,2))
>>> A
array([[ 1., 1.],
       [ 1., 1.]])
>>> B = 2 * np.ones((2,2))
>>> B
```

```
array([[ 2., 2.],
       [ 2., 2.]])
>>> A*B
array([[ 2., 2.],
       [ 2., 2.]])
>>> A.dot(B)
array([[ 4., 4.],
       [ 4., 4.]])
>>> A/B
array([[ 0.5, 0.5],
       [ 0.5, 0.5]])
```

A.7 Gráficos

Para criar um esboço do gráfico de uma função de uma variável real $y = f(x)$, podemos usar a biblioteca `Julia` mathplotlib.A função `matplotlib.pyplot.plot` faz uma representação gráfica de um conjunto de pontos $\{(x_i, y_i)\}$ fornecidos. Existe uma série de opções para esta função de forma que o usuário pode ajustar várias questões de visualização. Veja a documentação.

Exemplo A.7.1. Veja as seguintes linhas de código:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> def f(x): return x**3 + 1
...
>>> x = np.linspace(-2,2)
>>> plt.plot(x, f(x))
[<matplotlib.lines.Line2D object at 0x7f4f6d153510>]
>>> plt.grid()
>>> plt.show()
```

Resposta dos Exercícios

Recomendamos ao leitor o uso criterioso das respostas aqui apresentadas. Devido a ainda muito constante atualização do livro, as respostas podem conter imprecisões e erros.

E 2.1.1. a) 4; b) 9; c) b^2; d) 7; e) 170; f) 7,125; g) 3,28

E 2.1.2. a) 21,172; b) 5,5; c) 303,25; d) $4,\bar{6}$.

E 2.1.3. $(101,1)_2$.

E 2.1.4. $(11,1C)_{16}$.

E 2.1.5. a) $(12,3\bar{1})_5$; b) $(45,1)_6$.

E 2.1.6. 10,5; $(1010,1)_2$.

E 2.1.7. a) $(100101,001)_2$; b) $(11,4)_{16}$; c) $(11,5)_8$; d) $(9,A)_{16}$.

E 2.1.8. 50; 18.

E 2.2.1.

$$\begin{array}{ll} a)\ 2{,}99792458 \times 10^5 & b)\ 6{,}62607 \times 10^{-34} \\ c)\ 6{,}674 \times 10^{-8} & d)\ 9{,}80665 \times 10^4 \end{array} \tag{2.32}$$

E 2.2.2. No `GNU Octave`, temos:

```
>> printf("%1.7e\n", 299792.458)
2.9979458e+04
>> printf("%1.5e\n", 66.2607)
6.62607e+01
>> printf("%1.3e\n", 0.6674)
6.674e-01
>> printf("%1.5e\n", 9806.65e1)
9.80665e+04
```

E 2.3.1. (a) 1,1; (b) 7,3; (c) $-5,9$.

E 2.3.2. (a) 1,2; (b) 1,2; (c) 2,4; (d) $-2,4$.

E 2.3.3.

Este exercício está sem resposta sugerida. Proponha uma resposta. Veja como em:
https://github.com/livroscolaborativos/CalculoNumerico

E 2.4.1. a) $2^6 + 2^5 + 2^1 = 98$; b) $2^4 + 2^3 + 2^2 + 2^0 = 29$; c) -2^7; d) $-2^7 + 2^6 + 2^5 + 2^1 + 2^0 = -29$; e)$-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = -1$. Observe que o dígito mais significativo (mais à esquerda) tem peso negativo.

E 2.4.2. a) 25186; b) 7453; c) -7453; d) -1.

E 2.4.3. a) 70; b) -72; c) 72.

E 2.4.4. a) 17990; b) -18248; c) 18248.

E 2.4.5. a) 3,75; b) $-5{,}75$.

E 2.4.7. a) 3,75; b) $-5{,}75$.

E 2.4.8. Devido à precisão finita do sistema de numeração, o laço para quando x for suficientemente grande em comparação a 1 a ponto de x+1 ser aproximado para 1. Isso acontece quando 1 é da ordem do épsilon de máquina em relação a x, isto é, quando $x \approx 2/\%eps$. O tempo de execução fica em torno de 28 anos.

E 2.5.1. a) $\varepsilon_{abs} = 5{,}9 \times 10^{-4}$, $\varepsilon_{rel} = 1{,}9 \times 10^{-2}\%$; b) $\varepsilon_{abs} = \times 10^{-5}$, $\varepsilon_{rel} = \times 10^{-3}\%$; c) $\varepsilon_{abs} = 1$, $\varepsilon_{rel} = 10^{-5}\%$.

E 2.5.2. a) 1,7889; b) 1788,9; c) 0,0017889; d) 0,0045966; e) $2{,}1755 \times 10^{-10}$; f) $2{,}1755 \times 10^{10}$.

E 2.5.3. a) 3270, 3280; b) 42,5, 42,6; c) 0,0000333, 0,0000333.

E 2.5.4. a) 2; b) 2.

E 2.5.5.

$$0{,}1x - 0{,}01 = 12 \tag{2.84}$$
$$0{,}1x = 12 + 0{,}01 = 12{,}01 \tag{2.85}$$
$$x = 120{,}1 \tag{2.86}$$

A resposta exata é 120,1.

E 2.5.6. a) $\delta_{\text{abs}} = 3{,}46 \times 10^{-7}$, $\delta_{\text{rel}} = 1{,}10 \times 10^{-7}$; b) $\delta_{\text{abs}} = 1{,}43 \times 10^{-4}$, $\delta_{\text{rel}} = 1{,}00 \times 10^{-3}$.

E 2.8.1. 2%, deve-se melhorar a medida na variável x, pois, por mais que o erro relativo seja maior para esta variável, a propagação de erros através desta variáveis é muito menos importante do que para a outra variável.

E 2.8.2. 3,2% pela aproximação ou 3,4% pelo segundo método, isto é, $(0{,}96758 \leq I \leq 1{,}0342)$.

E 2.9.1. Quando μ é pequeno, $e^{1/\mu}$ é um número grande. A primeira expressão produz um "overflow" (número maior que o máximo representável) quando μ é pequeno. A segunda expressão, no entanto, reproduz o limite 1 quando $\mu \to 0+$.

E 2.9.2. a) $\frac{1}{2} + \frac{x^2}{4!} + O(x^4)$; b) $x/2 + O(x^2)$; c) $5 \cdot 10^{-4}x + O(x^2)$; d) $\frac{\sqrt{2}}{4}y + O(y^2) = \frac{\sqrt{2}}{4}x + O(x^2)$

E 2.9.3. A expressão da direita se comporta melhor devido à retirada do cancelamento catastrófico em x em torno de 0.

E 2.9.4. Possíveis soluções são:

$$\sqrt{e^{2x}+1}-e^x = \sqrt{e^{2x}+1}-e^x \cdot \frac{\sqrt{e^{2x}+1}+e^x}{\sqrt{e^{2x}+1}+e^x} \tag{2.212}$$

$$= \frac{e^{2x}+1-e^{2x}}{\sqrt{e^{2x}+1}+e^x} = \frac{1}{\sqrt{e^{2x}+1}+e^x} \tag{2.213}$$

e, de forma análoga:

$$\sqrt{e^{2x}+x^2}-e^x = \frac{x^2}{\sqrt{e^{2x}+x^2}+e^x}. \tag{2.214}$$

E 2.9.5. $4{,}12451228 \times 10^{-16}$ J; 0,002%; $0{,}26654956 \times 10^{-14}$ J; 0,002%; $4{,}98497440 \times 10^{-13}$ J; 0,057%; $1{,}74927914 \times 10^{-12}$ J; 0,522%.

Referências Bibliográficas

[1] Cecill and free sofware. http://www.cecill.info. Acessado em 30 de julho de 2015.

[2] M. Baudin. Introduction to scilab. http://forge.scilab.org/index.php/p/docintrotoscilab/. Acessado em 30 de julho de 2015.

[3] R.L. Burden and J.D. Faires. *Análise Numérica*. Cengage Learning, 8 edition, 2013.

[4] J. P. Demailly. *Analyse Numérique et Équations Differentielles*. EDP Sciences, Grenoble, nouvelle Édition edition, 2006.

[5] W Gautschi. Numerical analysis: An introduction birkhauser. *Barton, Mass, USA*, 1997.

[6] Walter Gautschi and Gabriele Inglese. Lower bounds for the condition number of vandermonde matrices. *Numerische Mathematik*, 52(3):241–250, 1987/1988.

[7] L.F. Guidi. Notas da disciplina cálculo numérico. http://www.mat.ufrgs.br/~guidi/grad/MAT01169/calculo_numerico.pdf. Acessado em julho de 2016.

[8] E. Isaacson and H.B. Keller. *Analysis of numerical methods*. Dover, Ontário, 1994.

[9] Arieh Iserles. *A first course in the numerical analysis of differential equations*. Cambridge university press, 2009.

[10] W.H. Press. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007.

[11] R. Rannacher. Einführung in die numerische mathematik (numerik 0). http://numerik.uni-hd.de/~lehre/notes/num0/numerik0.pdf. Acessado em 10.08.2014.

[12] Todos os Colaboradores. Cálculo numérico - um livro colaborativo - versão com scilab. disponível em https://www.ufrgs.br/reamat/CalculoNumerico/livro-sci/main.html, Novembro 2016.

Índice Remissivo