Capítulo 1

Computação numérica

Uma etapa intermediária importante durante a solução de um problema envolve a elaboração de um algoritmo, o qual deverá ser posteriormente implementado em uma linguagem de programação para a obtenção dos resultados numéricos em um computador.

O Cálculo Numérico é uma metodologia para resolver problemas matemáticos por intermédio de um computador, sendo amplamente utilizado por engenheiros e cientistas. Uma solução via Cálculo Numérico é sempre numérica, enquanto os métodos analíticos usualmente fornecem um resultado em termos de funções matemáticas. Muito embora uma solução numérica seja uma aproximação do resultado exato, ela pode ser obtida em grau crescente de exatidão. Uma solução numérica é calculada mesmo quando o problema não tem solução analítica, fato comum nas equações diferenciais. A integral indefinida

$$\int e^{-x^2} dx,$$

de grande utilidade em Estatística, possui primitiva que não pode ser representada, explicitamente, por funções elementares. A área sob a curva descrita por e^{-x^2} de a até b pode ser determinada por meio de algoritmos numéricos que são aplicáveis a qualquer outro integrando, não sendo, portanto, necessário fazer substituições especiais ou mesmo a integração por partes a fim de obter o resultado.

Para computar um resultado numérico, são necessárias as operações aritméticas (adição, subtração, multiplicação e divisão) e lógicas (comparação, conjunção, disjunção e negação). Considerando que estas são as únicas operações matemáticas que os computadores são capazes de realizar, então os computadores e o Cálculo Numérico formam uma combinação perfeita. Cumpre relembrar que o primeiro computador de grande porte totalmente eletrônico, o ENIAC (Electronic Numerical Integrator And Calculator), foi projetado para fazer cálculos balísticos e, atualmente, os maiores supercomputadores no mundo inteiro estão dedicados a realizar cálculos numéricos.

1.1 Etapas na solução de um problema

Dado um problema qualquer, como resolvê-lo no computador utilizando as técnicas do Cálculo Numérico? Será mostrado, a partir de um exemplo simples, que a solução de um problema pode ser obtida em quatro etapas: definição do problema, modelagem matemática, solução numérica e análise dos resultados.

1.1.1 Definição do problema

Nesta etapa, define-se qual é o *problema real* a ser resolvido. Seja, por exemplo, calcular $\sqrt{a}$, $a > 0$ usando apenas as quatro operações aritméticas.

1.1.2 Modelagem matemática

O *problema real* é transformado no *problema original* por meio de uma formulação matemática. No exemplo,

$$x = \sqrt{a} \longrightarrow x^2 = a \longrightarrow f(x) = x^2 - a = 0.$$

O *problema real*, calcular $\sqrt{a}$, $a > 0$, foi transformado no *problema original*, que é determinar a raiz de uma equação algébrica de grau 2.

Geralmente, o *problema original* possui mais soluções que o *problema real*. No exemplo, $+\sqrt{a}$ e $-\sqrt{a}$ são as duas raízes da equação algébrica.

1.1.3 Solução numérica

Nesta etapa, é feita a escolha do método numérico mais apropriado para resolver o *problema original* oriundo da modelagem matemática. Feita a escolha do método, este é descrito por intermédio de um algoritmo, o qual é posteriormente implementado em um computador para obtenção dos resultados numéricos.

Esta etapa pode ser subdividida em três fases: elaboração do algoritmo, codificação do programa e processamento do programa.

Elaboração do algoritmo

Um algoritmo é a descrição de um conjunto de comandos que, quando ativados, resultam em uma sucessão finita de acontecimentos. Em vez de implementar um método diretamente em uma linguagem de programação, é preferível descrevê-lo por meio de uma notação algorítmica. Com isso, é possível abstrair dos detalhes da linguagem de programação do computador e concentrar apenas nos aspectos matemáticos do método.

Além do mais, a descrição do método em uma notação algorítmica facilita a sua implementação em qualquer linguagem de programação. Na Seção 1.2, é apresentada a notação algorítmica adotada para descrever os métodos numéricos incluídos neste texto.

Codificação do programa

Nesta fase, o algoritmo é implementado na linguagem de programação escolhida. Como os aspectos matemáticos do método já foram pensados na fase de elaboração do algoritmo, a questão agora é só se preocupar com os detalhes de implementação da linguagem adotada. Os apêndices mostram como passar da notação algorítmica descrita na Seção 1.2 para as linguagens de programação FORTRAN (Apêndice A), Pascal (Apêndice B) e MATLAB (Apêndice C).

Processamento do programa

Finalmente, o código do programa obtido da implementação do algoritmo em uma linguagem de programação deve ser editado em um arquivo para que possa ser executado pelo computador. Se for detectado algum erro de sintaxe oriundo da fase de codificação, ele tem que ser corrigido para que o programa possa ser executado. Se na fase de processamento ocorrer algum erro de lógica, ou seja, a execução do programa produz resultados inesperados, então deve-se retornar à fase de elaboração para corrigir o algoritmo.

Exemplo 1.1 Para exemplificar a etapa de solução numérica no exemplo de cálculo de $\sqrt{a}$, será utilizado o método de Newton, a ser descrito na Seção 6.5.1, para calcular uma raiz de $f(x) = x^2 - a = 0$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

Substituindo $f(x)$ e $f'(x)$ na expressão acima, tem-se que

$$x_{k+1} = x_k - \frac{x_k^2 - a}{2x_k} = x_k - \frac{x_k}{2} + \frac{a}{2x_k},$$

ou seja,

$$x_{k+1} = \left(x_k + \frac{a}{x_k}\right) \times 0{,}5.$$

Este é um processo iterativo para calcular $\sqrt{a}$, a partir de um valor inicial x_0, usando apenas as operações aritméticas. Ele foi proposto pelos matemáticos babilônicos, mas, às vezes, também é atribuído a Heron de Alexandria ($\approx$ 100 d. C.) ou ao grego Arquitas (428-365 a. C.). Para o cálculo de $\sqrt{9}$, usando $x_0 = 1$, o processo babilônico produz os resultados

```
i     x_i      x_i-3
0   1.0000
1   5.0000    2.0000
2   3.4000    0.4000
3   3.0235    0.0235
4   3.0001    0.0001
5   3.0000    0.0000
```

A coluna `x_i` mostra as sucessivas aproximações de $\sqrt{9}$ a cada iteração `i` e a coluna `x_i-3` apresenta a diferença entre o valor aproximado `x_i` e o valor exato 3. ■

1.1.4 Análise dos resultados

A adequação da solução numérica ao *problema real* é verificada nesta última etapa. Se a solução não se mostrar satisfatória, deve-se obter um novo *problema original* por intermédio de uma nova formulação matemática e determinar uma nova solução numérica.

Exemplo 1.2 Para o exemplo de cálculo de raiz quadrada, se for atribuído o valor inicial $x_0 = -1$ (ou qualquer $x_0 < 0$), então o processo convergirá para -3, que, embora seja uma raiz de $f(x) = x^2 - 9 = 0$, não é $\sqrt{9}$.

```
i     x_i     x_i-3
0   -1.0000
1   -5.0000   -8.0000
2   -3.4000   -6.4000
3   -3.0235   -6.0235
4   -3.0001   -6.0001
5   -3.0000   -6.0000
```

■

Alguns modelos matemáticos podem produzir soluções que não têm sentido físico ou químico, como, por exemplo, tempo negativo, concentração complexa etc. O objetivo da análise dos resultados é justamente discernir qual a solução válida dentre as várias fornecidas pelo modelo matemático.

1.2 Notação algorítmica

A descrição de um algoritmo[1], por intermédio de uma notação algorítmica, melhora o seu entendimento, pois apenas os aspectos do raciocínio lógico são enfatizados, sem ser necessário levar em consideração os detalhes de implementação de uma linguagem de programação. Os algoritmos deste texto são descritos em uma notação baseada naquela proposta por Farrer e outros [18]. Apesar da descrição nos apêndices de como implementar os algoritmos deste texto em algumas linguagens, a literatura deve ser consultada para obter mais material de referência das linguagens de programação FORTRAN [16], Pascal [17], MATLAB [32] ou mesmo outra que se deseja utilizar.

1.2.1 Estrutura do algoritmo

Um algoritmo deve iniciar com

Algoritmo *<nome-do-algoritmo>*

e terminar com

fimalgoritmo

[1]Esta palavra deriva do nome do matemático árabe Mohammed ibu-Musa al-Khowarizmi ($\approx$ 800 d.C.).

Também,

{ **Objetivo:** *<objetivo-do-algoritmo>* }

deve ser utilizado para descrever a finalidade do algoritmo. Os dados necessários para a execução de um algoritmo são requisitados por meio do comando

parâmetros de entrada *<lista-de-variáveis>*

onde *<lista-de-variáveis>* são os nomes das variáveis, separadas por vírgulas, contendo os valores fornecidos. Não se faz necessário descrever exatamente como os valores dessas variáveis serão fornecidos ao algoritmo. Compete ao programador decidir durante a codificação do programa se os dados serão fornecidos interativamente pelo teclado, lidos de um arquivo, passados como argumentos de um subprograma ou até mesmo definidos como constantes dentro do próprio programa.

Por outro lado, os valores de interesse calculados pelo algoritmo são disponibilizados pelo comando

parâmetros de saída *<lista-de-variáveis>*

podendo a *<lista-de-variáveis>* ser ampliada ou reduzida pelo programador.

Exemplo 1.3 Este exemplo ilustra a estrutura básica de um algoritmo que deve ser implementado como um subprograma. Ele recebe os parâmetros de entrada necessários à sua execução e retorna os parâmetros de saída calculados.

```
Algoritmo Exemplo
{ Objetivo: Mostrar a estrutura de um algoritmo }
parâmetros de entrada a, b, c
parâmetros de saída x, y
  ⋮
fimalgoritmo
```

■

1.2.2 Variáveis e comentários

Uma variável corresponde a uma posição de memória do computador onde está armazenado um determinado valor. As variáveis são representadas por identificadores que são cadeias de caracteres alfanuméricos, podendo os elementos de vetores e matrizes ser referenciados por subscritos ou índices. Por exemplo, v_i ou $v(i)$ e m_{ij} ou $m(i,j)$.

Um comentário é um texto inserido em qualquer parte do algoritmo para aumentar a sua clareza. Esse texto deve ser delimitado por chaves { <texto> }, como, por exemplo, { cálculo da raiz }.

1.2.3 Expressões e comando de atribuição

Existem três tipos de expressões: aritméticas, lógicas e literais, dependendo dos tipos dos operadores e das variáveis envolvidas.

Expressões aritméticas

Expressão aritmética é aquela cujos operadores são aritméticos e cujos operandos são constantes e/ou variáveis aritméticas. A notação é semelhante àquela utilizada para representar uma fórmula como $\sqrt{b^2 - 4 * a * c}$, $\cos(2 + x)$ ou $massa * velocidade$.

O símbolo $\leftarrow$ é usado para atribuir o resultado de uma expressão a uma variável, ou seja,

<variável> $\leftarrow$ <expressão>

Por exemplo, $velocidade \leftarrow deslocamento/tempo$. A Tabela 1.1 apresenta algumas funções matemáticas que aparecem nos algoritmos descritos neste texto.

Tabela 1.1 Funções matemáticas.

Função	Descrição	Função	Descrição
	Trigonométricas		
sen	seno	cos	co-seno
tan	tangente	sec	secante
	Exponenciais		
exp	exponencial	$\log_{10}$	logaritmo decimal
$\log_e$	logaritmo natural	raiz_2	raiz quadrada
	Numéricas		
abs	valor absoluto	quociente	divisão inteira
arredonda	arredonda em direção ao inteiro mais próximo	sinal	sinal(x) = 1 se $x>0$, = 0 se $x=0$ e = -1 se $x<0$
max	maior valor	resto	resto de divisão
min	menor valor	trunca	arredonda em direção a 0

Exemplo 1.4 A Tabela 1.2 mostra exemplos de uso das funções matemáticas numéricas. ■

Tabela 1.2 Resultados de funções matemáticas numéricas.

Função	x e y	Valor	x e y	Valor
abs(x)	5	5	-3	3
arredonda(x)	0,4	0	0,5	1
quociente(x,y)	5 e 3	1	3 e 5	0
resto(x,y)	5 e 3	2	3 e 5	3
sinal(x)	-2	-1	7	1
trunca(x)	1,1	1	1,9	1

Expressões lógicas

Expressão lógica é aquela cujos operadores são lógicos e cujos operandos são relações e/ou variáveis do tipo lógico. Uma relação é uma comparação realizada entre valores do mesmo tipo. A natureza da comparação é indicada por um operador relacional definido conforme a Tabela 1.3. O resultado de uma relação ou de uma expressão lógica é **verdadeiro** ou **falso**.

Tabela 1.3 Operadores relacionais.

Operador relacional	Descrição
$>$	maior que
$\geq$	maior ou igual a
$<$	menor que
$\leq$	menor ou igual a
$=$	igual a
$\neq$	diferente de

Exemplo 1.5 Sendo $c = 1$ e $d = 3$, então $c \leq d$ é **verdadeiro**, enquanto se $x = 2$, $y = 3$ e $z = 10$, então $x + y = z$ é **falso**. ■

Os operadores lógicos mostrados na Tabela 1.4 permitem a combinação ou negação das relações lógicas.

Tabela 1.4 Operadores lógicos.

Operador lógico	Uso
e	conjunção
ou	disjunção
não	negação

A Tabela 1.5 mostra os resultados obtidos com os operadores lógicos, sendo o significado de V **verdadeiro** e o de F **falso**.

Tabela 1.5 Resultados com operadores lógicos.

a **e** b		
$a \backslash b$	V	F
V	V	F
F	F	F

a **ou** b		
$a \backslash b$	V	V
V	V	V
F	V	F

não a		
a	V	F
	F	V

Exemplo 1.6 Para os valores definidos no Exemplo 1.5, o resultado de ($d > c$ **e** $x + y + 5 = z$) é V **e** V, que implica **verdadeiro**. Por outro lado, ($d = c$ **ou** $x + y = z$) é F **ou** F, implicando **falso**. ■

Expressões literais

Uma expressão literal é formada por operadores literais e operandos, os quais são constantes e/ou variáveis do tipo literal.

O caso mais simples de uma expressão literal é uma constante literal, a qual é constituída por uma cadeia de caracteres delimitada por aspas, por exemplo, *mensagem* ← "matriz singular".

1.2.4 Comandos de entrada e saída

O comando

```
leia <lista-de-variáveis>
```

é usado para indicar que a *<lista-de-variáveis>* está disponível para leitura em algum dispositivo externo. Por sua vez, o comando

```
escreva <lista-de-variáveis>
```

deve ser utilizado para indicar onde certos valores de interesse estão disponíveis no programa e podem ser escritos em algum dispositivo externo. Compete ao programador decidir pela ampliação da *<lista-de-variáveis>* ou mesmo a omissão do comando **escreva**.

Exemplo 1.7 Elaborar um algoritmo para ler uma temperatura em grau Fahrenheit e converter para grau Celsius.

```
Algoritmo Converte_grau
{ Objetivo: Converter grau Fahrenheit para Celsius }
  leia Fahrenheit
  Celsius ← (Fahrenheit − 32) ∗ 5/9
  escreva Fahrenheit, Celsius
fimalgoritmo
```

■

1.2.5 Estruturas condicionais

O uso de uma estrutura condicional torna possível a escolha dos comandos a serem executados quando certa condição for ou não satisfeita, possibilitando, desta maneira, alterar o fluxo natural de comandos. Esta condição é representada por uma expressão lógica. As estruturas condicionais podem ser simples ou compostas.

Estrutura condicional simples

Esta estrutura apresenta a forma

```
se <condição> então
  <comandos>
fimse
```


Neste caso, a lista de *<comandos>* será executada se, e somente se, a expressão lógica *<condição>* tiver como resultado o valor **verdadeiro**.

Exemplo 1.8 Fazer um algoritmo para calcular o logaritmo decimal de um número positivo.

```
Algoritmo Logaritmo_decimal
{ Objetivo: Calcular logaritmo decimal }
  leia x
  se x > 0 então
    LogDec ← log10(x)
    escreva x, LogDec
  fimse
fimalgoritmo
```

Os comandos *LogDec* ← $\log_{10}(x)$ e **escreva** x, *LogDec* só serão executados se a variável x contiver um valor maior que zero. ■

Estrutura condicional composta

Quando houver duas alternativas possíveis, deve ser usada uma estrutura da forma

```
se <condição> então
  <comandos_1>
senão
  <comandos_2>
fimse
```

Se a expressão lógica *<condição>* tiver como resultado o valor **verdadeiro**, então a seqüência *<comandos_1>* será executada e a seqüência *<comandos_2>* não será executada.

Por outro lado, se o resultado de *<condição>* for **falso**, então será a lista *<comandos_2>* a única a ser executada.

Exemplo 1.9 Elaborar um algoritmo para avaliar a função modular $f(x) = |2x|$.

```
Algoritmo Função_modular
{ Objetivo: Avaliar uma função modular }
  leia x
  se x ≥ 0 então
    fx ← 2 ∗ x
  senão
    fx ← −2 ∗ x
  fimse
  escreva x, fx
fimalgoritmo
```

Se a variável x contiver um valor positivo ou nulo, então o comando $fx \leftarrow 2 * x$ será executado, seguindo-se o comando **escreva** x, fx. No entanto, se x contiver um valor negativo, os comandos $fx \leftarrow -2 * x$ e **escreva** x, fx serão os únicos a serem executados. ■

1.2.6 Estruturas de repetição

Uma estrutura de repetição faz com que uma seqüência de comandos seja executada repetidamente até que uma dada condição de interrupção seja satisfeita.

Existem, basicamente, dois tipos dessas estruturas, dependendo de ser o número de repetições indefinido ou definido.

Número indefinido de repetições

Este tipo de estrutura de repetição apresenta a forma

```
repita
   <comandos_1>
   se <condição> então
      interrompa
   fimse
   <comandos_2>
fimrepita
<comandos_3>
```

O comando **interrompa** faz com que o fluxo de execução seja transferido para o comando imediatamente a seguir do **fimrepita**. Assim, as listas *<comandos_1>* e *<comandos_2>* serão repetidas até que a expressão lógica *<condição>* resulte no valor **verdadeiro**. Quando isso ocorrer, a repetição será interrompida (*<comandos_2>* não será executada) e a lista *<comandos_3>*, após ao **fimrepita**, será executada.

Exemplo 1.10 Escrever um algoritmo para determinar o maior número de ponto flutuante (ver Seção 1.7) que, somado a 1, seja igual a 1.

```
Algoritmo Epsilon
{ Objetivo: Determinar a precisão da máquina }
  Epsilon ← 1
  repita
     Epsilon ← Epsilon/2
     se Epsilon + 1 = 1 então
        interrompa
     fimse
  fimrepita
  escreva Epsilon
fimalgoritmo
```


Esta seqüência faz com que seja calculada a chamada precisão da máquina ε. Quando a variável *Epsilon* assumir um valor que, adicionado a 1, seja igual a 1, então a estrutura **repita–fimrepita** é abandonada e o comando **escreva** *Epsilon* será executado. ■

A forma **repita–fimrepita** é o caso geral de uma estrutura de repetição. Se a lista *<comandos_1>* não existir, ter-se-á uma estrutura de repetição com interrupção no início (estrutura **enquanto**). Similarmente, se não houver a lista *<comandos_2>*, então será uma estrutura com interrupção no final (estrutura **repita–até**).

Número definido de repetições

Quando se souber com antecedência quantas vezes a estrutura deve ser repetida, pode ser usado um comando de forma mais simples

```
para <controle> ← <valor-inicial> até <valor-final> passo <delta> faça
   <comandos>
fimpara
```

Nesta estrutura, inicialmente, é atribuído à variável *<controle>* o valor de *<valor-inicial>* e verificado se ele é maior do que o *<valor-final>*. Se for maior, a estrutura **para-faça** não será executada. Se for menor ou igual, então os *<comandos>* serão executados e a variável *<controle>* será incrementada com o valor de *<delta>*. Novamente, é verificado se a variável *<controle>* é maior do que o *<valor-final>*; se não for maior, então os *<comandos>* serão executados e assim sucessivamente.

As repetições se processam até que a variável *<controle>* seja maior do que o *<valor-final>*. Quando o incremento *<delta>* tiver o valor 1, então o **passo** *<delta>* pode ser omitido da estrutura **para–faça**.

Exemplo 1.11 Escrever um algoritmo para mostrar que a soma dos n primeiros números ímpares é igual ao quadrado de n.

```
Algoritmo Primeiros_ímpares
{ Objetivo: Verificar propriedade dos números ímpares }
  leia n
  Soma ← 0
  para i ← 1 até 2 * n − 1 passo 2 faça
     Soma ← Soma + i
  fimpara
  escreva Soma, n²
fimalgoritmo
```

■

1.2.7 Falha no algoritmo

O comando

```
abandone
```

é usado para indicar que haverá uma falha evidente na execução do algoritmo, por exemplo, uma divisão por zero, uma singularidade da matriz ou mesmo o uso inapropriado de parâmetros. Neste caso, a execução será cancelada. O programador deve implementar o **abandone** na linguagem de programação a ser usada, caso o comando não esteja disponível.

1.2.8 Exemplos de algoritmos

Exemplo 1.12 Dado um vetor x com n componentes, a Figura 1.1 mostra um algoritmo para calcular a média aritmética $\bar{x}$ e o desvio padrão s^2 de seus elementos, sabendo que

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{e} \quad s^2 = \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right).$$

■

```
Algoritmo Média_desvio
{ Objetivo: Calcular média aritmética e desvio padrão }
parâmetros de entrada n, x
  { tamanho e elementos do vetor }
parâmetros de saída Média, DesvioPadrão
  Soma ← 0
  Soma2 ← 0
  para i ← 1 até n faça
    Soma ← Soma + x(i)
    Soma2 ← Soma2 + x(i)^2
  fimpara
  Média ← Soma/n
  DesvioPadrão ← raiz_2((Soma2 − Soma^2/n)/(n − 1))
  escreva Média, DesvioPadrão
fimalgoritmo
```

Figura 1.1 Algoritmo para cálculo da média aritmética e desvio padrão.

(Ver significado da função raiz_2 na Tabela 1.1, na página 6.)

Exemplo 1.13 A Figura 1.2 apresenta um algoritmo para determinar o maior elemento em cada linha de uma matriz A de dimensão $m \times n$. ■

Exemplo 1.14 Um algoritmo para calcular o valor de π, com precisão dada, utilizando a série

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots\right)$$

é exibido na Figura 1.3. Cumpre ressaltar que esta não é a maneira mais eficiente de calcular π, mas serve para ilustrar o uso da estrutura de repetição **repita–fimrepita**. ■


```
Algoritmo Matriz_maior
{ Objetivo: Determinar maior elemento em cada linha da matriz }
parâmetros de entrada m, n, A
  { número de linhas, número de colunas e elementos da matriz }
parâmetros de saída Maior
  { vetor contendo o maior elemento de cada linha }
  para i ← 1 até m faça
    Maior(i) ← A(i, 1)
    para j ← 2 até n faça
      se A(i, j) > Maior(i) então
        Maior(i) ← A(i, j)
      fimse
    fimpara
    escreva i, Maior(i)
  fimpara
fimalgoritmo
```

Figura 1.2 Algoritmo para determinar o maior elemento da linha de uma matriz.

```
Algoritmo Calcular_pi
{ Objetivo: Calcular o valor de π }
parâmetros de entrada Precisão
  { precisão no cálculo de π }
parâmetros de saída pi
  Soma ← 1
  Sinal ← −1
  Denominador ← 3
  repita
    Soma ← Soma + Sinal/Denominador
    se 1/Denominador < Precisão então
      interrompa
    fimse
    Sinal ← −Sinal
    Denominador ← Denominador + 2
  fimrepita
  pi ← 4 ∗ Soma
fimalgoritmo
```

Figura 1.3 Algoritmo para calcular o valor de π.

Exemplo 1.15 Conforme mostrado no Exemplo 4.5, o polinômio de quadrados mínimos que aproxima $\sqrt{x}$ para $0{,}01 \le x \le 1$ é $P(x) = 1{,}01865x^3 - 2{,}17822x^2 + 2{,}06854x + 0{,}10113$.

Pelo processo de Horner (ver Seção 6.1.1), o polinômio pode ser escrito na forma $P(x) = ((1{,}01865x - 2{,}17822)x + 2{,}06854)x + 0{,}10113$, de forma a economizar operações aritméticas. Na Figura 1.4, é apresentado um algoritmo para calcular uma aproximação de $\sqrt{x}$ usando o polinômio acima. ■

```
Algoritmo Aproximar_raiz
{ Objetivo: Calcular valor aproximado da raiz quadrada }
parâmetros de entrada x
   { valor que se deseja uma aproximação da raiz quadrada }
parâmetros de saída Aprox
   { aproximação de quadrados mínimos da raiz quadrada }
   se x < 0,01 ou x > 1 então
      escreva "argumento fora dos limites"
      abandone
   fimse
   c(1) ← 1,01865; c(2) ← −2,17822; c(3) ← 2,06854; c(4) ← 0,10113
   Aprox ← c(1)
   para i ← 2 até 4 faça
      Aprox ← Aprox * x + c(i)
   fimpara
fimalgoritmo
```

Figura 1.4 Algoritmo para avaliar uma aproximação de quadrados mínimos de $\sqrt{x}$.

Exemplo 1.16 Um algoritmo para calcular $\sqrt{a}$, $a > 0$, baseado no processo babilônico, descrito na Seção 1.1.3, é mostrado na Figura 1.5. ■

1.3 Notação matemática

Definida a modelagem matemática por meio de expressões aritméticas e lógicas, o passo seguinte é passar dessa notação matemática para a notação algorítmica proposta. Esta passagem será ilustrada por meio de alguns exemplos.

Exemplo 1.17 A Figura 1.6 mostra um algoritmo para calcular a norma-2 ou norma Euclidiana de um vetor x de tamanho n, definida por

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}.$$

■

Exemplo 1.18 O algoritmo da Figura 1.7 determina a norma-∞ ou norma de máxima magnitude de um vetor x de tamanho n

$$\|x\|_\infty = \max_{1 \le i \le n} |x_i|.$$

■


```
Algoritmo Raiz2
{ Objetivo: Calcular raiz quadrada pelo processo babilônico }
parâmetros de entrada a, Toler
   { valor para calcular a raiz e tolerância }
parâmetros de saída Raiz   { raiz quadrada de a }
   { teste se a é não positivo }
   se a ≤ 0 então escreva "argumento inválido", abandone, fimse
   { cálculo do valor inicial x0 = z }
   c(1) ← 1,01865; c(2) ← −2,17822; c(3) ← 2,06854; c(4) ← 0,10113
   p ← 1; b ← a
   se a > 1 então
      repita
         b ← b * 0,01; p ← p * 10
         se b ≤ 1 então interrompa, fimse
      fimrepita
   fimse
   se a < 0,01 então
      repita
         b ← b * 100; p ← p * 0,1
         se b ≥ 0,01 então interrompa, fimse
      fimrepita
   fimse
   z ← c(1)
   para i ← 2 até 4 faça
      z ← z * b + c(i)
   fimpara
   z ← z * p; i ← 0; escreva i, z
   { cálculo da raiz }
   repita
      x ← (z + a/z) * 0,5; Delta ← abs(x − z); i ← i + 1
      escreva i, x, Delta
      se Delta ≤ Toler ou i = 50 então interrompa, fimse; z ← x
   fimrepita
   { teste de convergência }
   se Delta ≤ Toler então
      Raiz ← x
   senão
      escreva "processo não convergiu com 50 iterações"
   fimse
fimalgoritmo
```

Figura 1.5 Cálculo de raiz quadrada pelo processo babilônico.
(Ver significado da função abs na Tabela 1.1, na página 6.)

```
Algoritmo Norma2
{ Objetivo: Calcular a norma-2 de um vetor }
parâmetros de entrada n, x
  { tamanho do vetor e o vetor }
parâmetros de saída N2
  { norma-2 do vetor }
  Soma ← 0
  para i ← 1 até n faça
    Soma ← Soma + (abs(x(i)))^2
  fimpara
  N2 ← raiz2(Soma)
fimalgoritmo
```

Figura 1.6 Norma-2 de um vetor x de tamanho n.

(Ver significado das funções abs e raiz$_2$ na Tabela 1.1, na página 6.)

```
Algoritmo NormaInf
{ Objetivo: Calcular a norma-∞ de um vetor }
parâmetros de entrada n, x
  { tamanho do vetor e o vetor }
parâmetros de saída Ninf
  { norma-∞ do vetor }
  Ninf ← abs(x(1))
  para i ← 2 até n faça
    se abs(x(i)) > Ninf então
      Ninf ← abs(x(i))
    fimse
  fimpara
fimalgoritmo
```

Figura 1.7 Norma-∞ de um vetor x de tamanho n.

(Ver significado da função abs na Tabela 1.1, na página 6.)

Exemplo 1.19 A Figura 1.8 apresenta um algoritmo para calcular o vetor x $(n \times 1)$ resultante do produto de uma matriz A $(n \times m)$ por um vetor v $(m \times 1)$

$$x_i = \sum_{j=1}^{m} a_{ij} v_j, \; i = 1, 2, \ldots, n.$$

■


```
Algoritmo Matvet
{ Objetivo: Calcular o produto de uma matriz por um vetor }
parâmetros de entrada n, m, A, v
  { número de linhas, número de colunas, }
  { elementos da matriz e elementos do vetor }
parâmetros de saída x
  { vetor resultante do produto matriz-vetor }
  para i ← 1 até n faça
    Soma ← 0
    para j ← 1 até m faça
      Soma ← Soma + A(i,j) * v(j)
    fimpara
    x(i) ← Soma
  fimpara
fimalgoritmo
```

Figura 1.8 Produto matriz-vetor.

1.4 Complexidade computacional

É usual definir uma função de complexidade para medir o custo de execução de um programa. Esta função pode ser tanto uma medida do tempo para executar o algoritmo que resolve um problema de tamanho n quanto o espaço de memória requerido para esta execução.

A complexidade computacional de um algoritmo se refere à estimativa do esforço computacional despendido para resolver o problema e é medido pelo número necessário de operações aritméticas e lógicas como, por exemplo, o número de adições e multiplicações efetuadas para resolver um sistema linear de ordem n.

Os problemas possuem complexidade de tempo que pode ser enquadrada em dois grupos [40]. O primeiro é composto pelos algoritmos polinomiais, sendo a função de complexidade da forma $O(c_p n^p + c_{p-1} n^{p-1} + \ldots + c_1 n + c_0)$. O outro grupo é formado pelos algoritmos exponenciais, onde a função de complexidade tem a forma $O(c^n), c > 1$.

Os algoritmos estudados neste texto são polinomiais. Como as operações aritméticas demandam diferentes tempos para serem executadas pelo computador, a função de complexidade será definida, separadamente, para adição, multiplicação e divisão, sendo uma subtração contada como uma adição.

Por exemplo, o número de adições necessárias para fazer a decomposição LU de uma matriz de ordem n é $O(\frac{1}{3}n^3 - \frac{1}{2}n^2 + \frac{1}{6}n)$, ou, simplesmente, $O(n^3)$. O número de multiplicações utilizadas para resolver um sistema triangular inferior de ordem n usando as substituições sucessivas é $O(\frac{1}{2}n^2 - \frac{1}{2}n)$ ou $O(n^2)$.

Exemplo 1.20 Seja o polinômio interpolador de Lagrange de grau n, definido por (3.5), na página 129

$$L_n(x) = \sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}.$$

Expandindo, resulta a Expressão 1, cujo algoritmo é mostrado na Figura 1.9

$$\begin{aligned} L_n(x) &= y_0 \times \frac{x - x_1}{x_0 - x_1} \times \frac{x - x_2}{x_0 - x_2} \times \ldots \times \frac{x - x_n}{x_0 - x_n} \\ &+ y_1 \times \frac{x - x_0}{x_1 - x_0} \times \frac{x - x_2}{x_1 - x_2} \times \ldots \times \frac{x - x_n}{x_1 - x_n} \\ \ldots &+ y_n \times \frac{x - x_0}{x_n - x_0} \times \frac{x - x_1}{x_n - x_1} \times \ldots \times \frac{x - x_{n-1}}{x_n - x_{n-1}}. \end{aligned}$$

```
Algoritmo Lagrange_Expressão_1
{ Objetivo: Interpolar usando polinômio de Lagrange }
parâmetros de entrada m, x, y, z
   { número de pontos, abscissas }
   { ordenadas e valor a interpolar }
parâmetros de saída r  { valor interpolado }
   r ← 0
   para i ← 1 até m faça
      p ← y(i)
      para j ← 1 até m faça
         se i ≠ j então
            p ← p * ((z - x(j))/(x(i) - x(j)))
         fimse
      fimpara
      r ← r + p
   fimpara
fimalgoritmo
```

Figura 1.9 Exemplo de algoritmo do polinômio de Lagrange.

Considerando que o número de pontos m usados na interpolação é igual a $n+1$, onde n é o grau do polinômio, então a complexidade computacional do algoritmo da Figura 1.9 é

Adições: $\sum_{i=1}^{m} 2(m-1) + 1 = 2m^2 - 2m + m = 2(n+1)^2 - (n+1) = 2n^2 + 3n + 1$;

Multiplicações: $\sum_{i=1}^{m} (m-1) = m^2 - m = (n+1)^2 - (n+1) = n^2 + n$;

Divisões: $\sum_{i=1}^{m} (m-1) = m^2 - m = (n+1)^2 - (n+1) = n^2 + n$.

Estes resultados estão resumidos na Tabela 1.6.

Tabela 1.6 Complexidade da interpolação de Lagrange via Expressão 1.

(n: grau do polinômio interpolador.)

Operações	Complexidade
adições	$2n^2 + 3n + 1$
multiplicações	$n^2 + n$
divisões	$n^2 + n$

O polinômio de Lagrange também pode ser expandido de modo a resultar a Expressão 2

$$\begin{aligned} L_n(x) &= y_0 \times \frac{(x - x_1) \times (x - x_2) \times \ldots \times (x - x_n)}{(x_0 - x_1) \times (x_0 - x_2) \times \ldots \times (x_0 - x_n)} \\ &+ y_1 \times \frac{(x - x_0) \times (x - x_2) \times \ldots \times (x - x_n)}{(x_1 - x_0) \times (x_1 - x_2) \times \ldots \times (x_1 - x_n)} \\ \ldots &+ y_n \times \frac{(x - x_0) \times (x - x_1) \times \ldots \times (x - x_{n-1})}{(x_n - x_0) \times (x_n - x_1) \times \ldots \times (x_n - x_{n-1})}. \end{aligned}$$

O algoritmo desta expressão é mostrado na Figura 3.2, na página 132, e a sua complexidade computacional é compilada na Tabela 3.2, na página 132. Comparando os resultados das Tabelas 1.6 e 3.2, nota-se que o número de adições é o mesmo e o de multiplicações é da mesma ordem (n^2). No entanto, o número de divisões utilizado pela Expressão 2 é de uma ordem de grandeza a menos. ■

O polinômio de Lagrange serve para exemplificar que uma mesma notação matemática pode resultar em algoritmos de diferentes complexidades. Isto deve ser lembrado ao se elaborar um algoritmo.

1.5 Implementação de algoritmos

Uma das quatro etapas na solução de um problema é a solução numérica, a qual pode ser subdividida em três fases: elaboração do algoritmo, codificação do programa e processamento do programa. Nas seções anteriores foi proposta uma notação algorítmica e mostrado como elaborar um algoritmo a partir de uma formulação matemática. A próxima fase é a codificação do programa na linguagem escolhida. Nesta seção serão mostrados três exemplos de implementações dos algoritmos nas linguagens de programação FORTRAN, Pascal e MATLAB descritas nos Apêndices A, B e C, respectivamente.

Exemplo 1.21 Implementar em FORTRAN o algoritmo do Exemplo 1.10 usando variável de ponto flutuante de 8 *bytes*.

```
      program PreMaq
c       Programa para determinar a precisao da maquina
c       para variavel real de 8 bytes
      real*8 Epsilon
      Epsilon =  1.0d0
   10 continue
        Epsilon = Epsilon / 2.0d0
      if( Epsilon+1.0d0.ne.1.0d0 ) go to 10
      write(*,16) Epsilon
      stop
   16 format('Precisao da maquina:',1pd15.8)
      end
```

A execução do programa fornece o resultado

```
Precisao da maquina: 1.11022302E-16
```

que é igual a 2^{-53}. Se for utilizada variável real de 4 bytes, o resultado será

```
Precisao da maquina: 5.96046448E-08
```

sendo igual a 2^{-24}. Estes resultados mostram que se qualquer número menor ou igual à precisão da máquina for somado a 1, o resultado será 1. ■

Exemplo 1.22 Implementar em Pascal o algoritmo da Figura 1.1.

```
program Media_desvio;
type vetor = array[1..100] of real;
var n: integer;
    Media, DesvioPadrao: real;
    x: vetor;
{     Calculo da media aritmetica e desvio padrao }
procedure MediaDesvioPadrao(n:integer;x:vetor;var Media,DesvioPadrao:real);
var i: integer;
    Soma, Soma2: real;
begin
   Soma := 0;
   Soma2 := 0;
   for i := 1 to n do begin
      Soma := Soma + x[i];
      Soma2 := Soma2 + sqr(x[i]);
   end;
   Media := Soma / n;
   DesvioPadrao := sqrt((Soma2-sqr(Soma)/n)/(n-1));
end; { procedure MediaDesvioPadrao }
begin
var i: integer;
    writeln('Numero de elementos: ');
    readln(n);
    writeln('Elementos: ');
    for i := 1 to n do
       read(x[i]);
    MediaDesvioPadrao(n, x, Media, DesvioPadrao);
    writeln('Media =',Media:10:5,'   Desvio padrao =',DesvioPadrao:10:5);
end.
```

■

Exemplo 1.23 Implementar em MATLAB o algoritmo da Figura 1.3.

```
%  Calculo de pi com uma dada precisao
function Pi = Calcular_pi(Precisao)
Soma = 1;
Sinal = -1;
Denominador = 3;
while 1
   Soma = Soma + Sinal / Denominador;
   if 1 / Denominador < Precisao
      break
   end
   Sinal = -Sinal;
   Denominador = Denominador + 2;
end
Pi = 4 * Soma;
```

■

1.6 Tipos de erros

Durante as etapas de solução de um problema, surgem erros de várias fontes que podem alterar profundamente os resultados obtidos. É de importância fundamental conhecer as causas desses erros para minimizar as suas conseqüências.

Erro de truncamento

O erro de truncamento é devido à aproximação de uma fórmula por outra. É sabido que, para avaliar uma função matemática no computador, somente as operações aritméticas e lógicas podem ser requeridas, por serem as operações que ele é capaz de efetuar. Por exemplo, para avaliar $f(x) = \text{sen}(x)$ esta tem que ser aproximada por uma série, tal como

$$\text{sen}(x) = \sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \ldots,\ 0 \le x \le \frac{\pi}{4}.$$

À medida que n aumenta, mais o valor da série se aproxima do valor real. A Tabela 1.7 mostra a diferença entre o valor obtido pela série de $\text{sen}(x)$ e um valor mais exato, para n até 2, 3 e 4. Quando n aumenta, o erro de truncamento diminui, ficando claro que estes erros são devidos aos truncamentos da série.

Tabela 1.7 Efeito do erro de truncamento no cálculo de $\text{sen}(x)$.

	$\sum_{n=0}^{t}(-1)^n \frac{x^{2n+1}}{(2n+1)!} - \text{sen}(x)$		
x	$t=2$	$t=3$	$t=4$
0	0	0	0
$\pi/16$	$2{,}4\times10^{-6}$	$2{,}2\times10^{-9}$	$1{,}2\times10^{-12}$
$\pi/8$	$7{,}8\times10^{-5}$	$2{,}9\times10^{-7}$	$6{,}1\times10^{-10}$
$\pi/6$	$3{,}3\times10^{-4}$	$2{,}1\times10^{-6}$	$8{,}1\times10^{-9}$
$\pi/4$	$2{,}5\times10^{-3}$	$3{,}6\times10^{-5}$	$3{,}1\times10^{-7}$

Erro absoluto e relativo

O erro cometido na computação de um resultado pode ser medido de duas maneiras. A primeira é o erro absoluto definido como

$$\text{erro absoluto} = \text{valor real} - \text{valor aproximado}.$$

O tamanho do erro absoluto é mais grave quando o valor verdadeiro for pequeno. Por exemplo, $1711{,}321 \pm 0{,}030$ é exato com cinco dígitos significativos, enquanto $0{,}001 \pm 0{,}030$ tem pouco significado. A outra maneira é o erro relativo definido como

$$\text{erro relativo} = \frac{\text{valor real} - \text{valor aproximado}}{\text{valor real}},$$

sendo indefinido para valor real nulo. A sua vantagem sobre o erro absoluto é a independência da magnitude dos valores.

Erro na modelagem

Na etapa da modelagem matemática de um problema real, muitas vezes se faz necessário o uso de dados obtidos por medidas experimentais. Pode ocorrer tanto uma modelagem incorreta na qual a expressão matemática não reflete perfeitamente o fenômeno físico quanto os dados terem sido obtidos com pouca exatidão. Nesses casos, é necessária a realização de testes para verificar o quanto os resultados são sensíveis às alterações dos dados fornecidos.

Mudanças grandes nos resultados devido a pequenas variações nos dados são sintomas de um malcondicionamento do modelo proposto, sendo uma nova modelagem do fenômeno a tentativa de cura do problema.

Erro grosseiro

A possibilidade de um computador cometer um erro é muito pequena; no entanto, podem ser cometidos erros na elaboração do algoritmo, na sua implementação e mesmo na digitação de dados. Executar o programa, cujo resultado seja conhecido, ajuda a remover erros, mas demonstra, apenas, que o programa está correto para aquela massa de dados! A solução seria elaborar uma *prova de correção de programa* que é uma tarefa não trivial.

Erro de arredondamento

Um número decimal qualquer, por exemplo $0{,}4_{10}$ (0,4 na base 10), não pode ser representado exatamente em um computador porque ele tem que ser convertido para a base 2 e armazenado em um número finito de *bits*. O erro causado por esta imperfeição na representação de um número é chamado de erro de arredondamento. As causas e conseqüências desse tipo de erro serão abordadas introdutoriamente na Seção 1.7.

Para uma análise mais detalhada sobre os efeitos do erro de arredondamento deve ser consultado um texto mais específico, como, por exemplo, um livro clássico de James Hardy Wilkinson [39].

1.7 Aritmética de ponto flutuante

Para uma melhor compreensão das causas do erro de arredondamento, se faz necessário conhecer como os números são armazenados em um computador. Um número pode ser representado com ponto fixo, por exemplo, 12,34 ou com ponto flutuante[2] $0{,}1234{\times}10^2$. Neste texto, será abordada apenas a aritmética de ponto flutuante. A forma geral de representação de um número de ponto flutuante é similar à notação científica

$$\pm .d_1 d_2 d_3 \ldots d_p \times B^e,$$

onde os d_i's são os dígitos da parte fracionária, tais que $0 \le d_i \le B-1$, $d_1 \ne 0$, B é o valor da base (geralmente 2, 10 ou 16), p é o número de dígitos e e é um expoente inteiro. Deste modo, um número de ponto flutuante tem três partes: o sinal, a parte fracionária chamada de significando ou mantissa e o expoente. Estas três partes têm um comprimento total fixo que depende do computador e do tipo de número: precisão simples, dupla ou estendida, conforme será visto mais adiante. Com o intuito de mostrar a representação de um número de ponto flutuante, seja um computador hipotético com dois dígitos ($p = 2$), base $B = 2$ e expoente na faixa $-1 \le e \le 2$. Como o número é normalizado, isto é, $d_1 \ne 0$, eles serão da forma

$$\pm .10_2 \times 2^e \text{ ou } \pm .11_2 \times 2^e,\ e = -1, \ldots, 2.$$

Considerando a conversão de binário para decimal de um número menor que 1,

$$.10_2 = 1 \times 2^{-1} + 0 \times 2^{-2} = 1/2 \text{ e}$$
$$.11_2 = 1 \times 2^{-1} + 1 \times 2^{-2} = 3/4,$$

então os únicos números positivos representáveis neste computador são

$.10_2 \times 2^{-1}$	$= 1/2 \times 2^{-1}$	$= 1/4$	$.11_2 \times 2^{-1}$	$= 3/4 \times 2^{-1}$	$= 3/8$
$.10_2 \times 2^{0}$	$= 1/2 \times 1$	$= 1/2$	$.11_2 \times 2^{0}$	$= 3/4 \times 1$	$= 3/4$
$.10_2 \times 2^{1}$	$= 1/2 \times 2$	$= 1$	$.11_2 \times 2^{1}$	$= 3/4 \times 2$	$= 3/2$
$.10_2 \times 2^{2}$	$= 1/2 \times 4$	$= 2$	$.11_2 \times 2^{2}$	$= 3/4 \times 4$	$= 3$

O zero é representado de uma forma especial: todos os dígitos d_i do significando e do expoente são nulos. O mais importante a ser observado acerca dos números de ponto flutuante é que eles são discretos e não contínuos como um *número real* definido na Matemática, conforme pode ser visto na Figura 1.10.

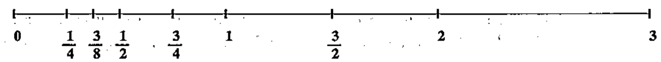

Figura 1.10 Valores discretos dos números de ponto flutuante.

[2] Os nomes corretos desses formatos em português deveriam ser vírgula fixa e vírgula flutuante, pois em nossa língua é a vírgula que separa a parte inteira de um número da sua parte decimal. No entanto, como a terminologia ponto fixo e ponto flutuante já está consagrada na literatura, optamos por adotá-la.

O conceito de sempre existir um número real entre dois números reais quaisquer não é válido para os números de ponto flutuante. A falha deste conceito tem conseqüência desastrosa, conforme será mostrado a seguir. Considere a representação binária

$$0{,}6_{10} = 0{,}100110011001\ldots_2 \text{ e } 0{,}7_{10} = 0{,}1011001100110\ldots_2.$$

Se estes dois números forem armazenados naquele computador hipotético, eles serão representados igualmente como $.10_2 \times 2^0$. Isto significa que tanto $0{,}6_{10}$ quanto $0{,}7_{10}$ serão vistos como $0{,}5_{10}$ por aquele computador. Esta é uma grande causa de erro de arredondamento nos processos numéricos.

A forma de representação de um número de ponto flutuante depende do fabricante do computador, portanto um mesmo programa implementado em computadores que utilizam formatos diferentes pode fornecer resultados diferentes.

O formato utilizado pela maioria dos microcomputadores e estações de trabalho é aquele proposto pelo IEEE (Institute of Electrical and Electronics Engineers), o qual é mostrado na Tabela 1.8 [25].

Tabela 1.8 Formato IEEE de ponto flutuante.

Propriedade	Precisão Simples	Precisão Dupla	Precisão Estendida
comprimento total	32	64	80
bits na mantissa	23	52	64
bits no expoente	8	11	15
base	2	2	2
expoente máximo	127	1023	16383
expoente mínimo	−126	−1022	−16382
maior número	$\approx 3{,}40 \times 10^{38}$	$\approx 1{,}80 \times 10^{308}$	$\approx 1{,}19 \times 10^{4932}$
menor número	$\approx 1{,}18 \times 10^{-38}$	$\approx 2{,}23 \times 10^{-308}$	$\approx 3{,}36 \times 10^{-4932}$
dígitos decimais	7	16	19

Se uma operação aritmética resultar em um número que seja maior em módulo que o maior número representável, ocorrerá um *overflow*. Se, por outro lado, resultar em um número que em módulo seja menor que o menor número representável diferente de zero, ocorrerá um *underflow*. O modo de tratar *overflow* e *underflow* dependerá do compilador utilizado para gerar o programa.

Será mostrada, a seguir, a precisão das operações numéricas envolvendo números de ponto flutuante. Para tal, será utilizado um outro computador hipotético com dois dígitos ($p = 2$), base $B = 10$ para facilitar o entendimento e expoente $e = -5, \ldots, 5$: $\pm .d_1 d_2 \times 10^e$.

Quando dois números são somados ou subtraídos, os dígitos do número de expoente menor devem ser deslocados de modo a alinhar as casas decimais. O resultado é, então, arredondado para dois dígitos para caber na mantissa de tamanho $p = 2$. Isto feito, o expoente é ajustado de forma a normalizar a mantissa ($d_1 \neq 0$).

Exemplo 1.24 Somar 4,32 e 0,064.
Os números são armazenados no formato especificado, as casas decimais são alinhadas e a operação de adição é efetuada. O resultado é arrendondado para dois dígitos

$$\begin{array}{rlrl} 4{,}32 + 0{,}064 = .43{\times}10^1 + .64{\times}10^{-1} = & & .43 & \times 10^1 \\ & + & .0064 & \times 10^1 \\ & = & .4364 & \times 10^1 \\ & \rightarrow & .44 & \times 10^1. \end{array}$$

O resultado da adição foi 4,4 em vez de 4,384. ■

Exemplo 1.25 Subtrair 371 de 372.
Os números são armazenados no formato especificado, resultando em um mesmo valor no caso e a operação de subtração é efetuada. O resultado é convertido para zero

$$\begin{array}{rlrl} 372 - 371 = .37{\times}10^3 - .37{\times}10^3 = & & .37 & \times 10^3 \\ & - & .37 & \times 10^3 \\ & = & .00 & \times 10^3 \\ & \rightarrow & .00 & \times 10^0. \end{array}$$

A subtração deu 0 em vez de 1. A perda de precisão quando dois números aproximadamente iguais são subtraídos é a maior fonte de erro nas operações de ponto flutuante. ■

Exemplo 1.26 Somar 691 e 2,71.
Os números são armazenados no formato especificado, as casas decimais são alinhadas e a operação de adição é efetuada. O resultado é arrendondado para dois dígitos

$$\begin{array}{rlrl} 691 + 2{,}71 = .69{\times}10^3 + .27{\times}10^1 = & & .69 & \times 10^3 \\ & + & .0027 & \times 10^3 \\ & = & .6927 & \times 10^3 \\ & \rightarrow & .69 & \times 10^3. \end{array}$$

A adição resultou em 690 em vez de 693,71. O deslocamento das casas decimais de 2,71 causou uma perda total dos seus dígitos durante a operação. ■

Exemplo 1.27 Multiplicar 1234 por 0,016.
Os números são armazenados no formato definido e a operação de multiplicação é efetuada utilizando-se $2p = 4$ dígitos na mantissa. O resultado é arrendondado para dois dígitos e normalizado

$$\begin{array}{rlrl} 1234 \times 0{,}016 = .12{\times}10^4 \times .16{\times}10^{-1} = & & .12 & \times 10^4 \\ & \times & .16 & \times 10^{-1} \\ & = & .0192 & \times 10^3 \\ & \rightarrow & .19 & \times 10^2. \end{array}$$

O resultado da multiplicação foi 19 em vez de 19,744. ■