1. Computação numérica

1.1 Etapas na solução de um problema.

1.2 Notação algorítmica.

1.3 Tipos de erros.

1.4 Aritmética de ponto flutuante.

Computação numérica

- O Cálculo Numérico.
- Solução via Cálculo Numérico.
- Operações aritméticas:
  adição, subtração, multiplicação e divisão.
- Operações lógicas:
  comparação, conjunção, disjunção e negação.
- Solução de um problema

  1. definição do problema,
  2. modelagem matemática,
  3. solução numérica e
  4. análise dos resultados.

Etapas na solução de um problema

Definição do problema

- Define-se o *problema real* a ser resolvido.
- Calcular $\sqrt{a},\ a > 0$, usando as quatro operações aritméticas.

Modelagem matemática

- Formulação matemática transforma *problema real* em *problema original*

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

- *Problema original* possui mais soluções que o *problema real*

  $+\sqrt{a}$ e $-\sqrt{a}$.

Solução numérica

- Escolha do método numérico para resolver o *problema original*.
- Método descrito por um algoritmo.
- Algoritmo implementado por uma linguagem.
- Solução numérica dividida em três fases

  1. elaboração do algoritmo,
  2. codificação do programa e
  3. processamento do programa.

Elaboração do algoritmo

- Não implementar método em uma linguagem.
- Descrever método em notação algorítmica.
- Abstrair dos detalhes da linguagem de programação utilizada.
- Concentrar nos aspectos matemáticos.
- Facilitar a implementação em uma linguagem qualquer.

Codificação do programa

- Implementar algoritmo na linguagem escolhida.
- Preocupar com detalhes de implementação.

Processamento do programa

- Editar código do programa em arquivo.
- Executar código no computador.
- Detectar erro de sintaxe e de lógica.

Exemplo de solução numérica

- Método de Newton para calcular raiz de

$$f(x) = x^2 - a = 0,$$

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

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

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

- Processo babilônico produz os resultados para o cálculo de $\sqrt{9}$ usando $x_0 = 1$

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

Análise dos resultados

- Adequação da solução numérica ao *problema real*.
- Se solução não for satisfatória então obter um novo *problema original*.
- Para valor inicial $x_0 = -2$ o processo convergirá para -3

```
i       x_i      x_i-3
0    -2.0000
1    -3.2500   -6.2500
2    -3.0096   -6.0096
3    -3.0000   -6.0000
```

- Solução do modelo matemático pode produzir soluções sem sentido físico:
  tempo negativo, concentração complexa, etc.
- Análise dos resultados discerne qual é a solução válida.

Notação algorítmica

- Descrição do algoritmo por uma notação algorítmica melhora seu entendimento.
- São enfatizados apenas os aspectos do raciocínio lógico.
- Não considera detalhes de implementação da linguagem.
- Mohammed ibu-Musa al-Khowarizmi ($\approx$ 800 D. C.).

Estrutura do algoritmo

- Iniciar

  `Algoritmo` <nome-do-algoritmo>.

- Terminar

  `fim algoritmo`.

- Descrever finalidade

  { `Objetivo:` <objetivo-do-algoritmo> }.

- Dados para execução do algoritmo

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

- Valores calculados pelo algoritmo

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

Exemplo de algoritmo

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

Variáveis e comentários

- Variável corresponde a posição de memória.
- Variáveis representadas por identificadores.
- Cadeias de caracteres alfanuméricos.
- Vetores e matrizes referenciados por subscritos ou índices: v_i ou v(i) e m_{ij} ou m(i,j).
- Comentário é um texto inserido no algoritmo para aumentar sua clareza.
- Texto delimitado por chaves { <texto> }

  { cálculo da raiz }.

Expressões e comando de atribuição

- Expressões: aritméticas, lógicas e literais.

Expressões aritméticas

- Operadores aritméticos e operandos são constantes e/ou variáveis aritméticas.
- Notação semelhante à fórmula

  $\sqrt{b^2 - 4 * a * c}$, $\cos(2 + x)$ ou

  $massa * velocidade$.

- Símbolo $\leftarrow$ usado para atribuir resultado de expressão à variável

  <variável> $\leftarrow$ *<expressão>*.

  $velocidade \leftarrow deslocamento/tempo$.

Expressões lógicas

- Operadores lógicos e operandos são relações e/ou variáveis do tipo lógico.
- Relação: comparação realizada entre valores do mesmo tipo.
- Comparação indicada por um operador relacional

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

.

- Resultado de relação ou expressão lógica: `verdadeiro` ou `falso`.

Exemplo 1

Para $c = 1$ e $d = 3$: $c \leq d$ é `verdadeiro`

Para $x = 2$, $y = 3$ e $z = 10$: $x + y = z$ é `falso`.

Operadores lógicos

- Permitem combinação ou negação das relações lógicas

operador lógico	uso
e	conjunção
ou	disjunção
não	negação

- Resultados com operadores lógicos,
  V = verdadeiro e F = falso

a e b		
a\b	V	F
V	V	F
F	F	F

a ou b		
a\b	V	F
V	V	V
F	V	F

não a		
a	V	F
	F	V

.

Exemplo 2

Para c = 1, d = 3, x = 2, y = 3 e z = 10:

(d > c **e** x + y + 5 = z) ⟶ V **e** V ⟶ verdadeiro.

(d = c **ou** x + y = z) ⟶ F **ou** F ⟶ falso.

Expressões literais

- Expressão literal: formada por operadores e operandos literais.
- Expressão literal mais simples:
  cadeia de caracteres delimitada por aspas

  mensagem $\leftarrow$ "matriz singular".

Comandos de entrada e saída

- Leitura em dispositivo externo

`leia` *<lista-de-variáveis>* .

- Escrita em dispositivo externo

`escreva` *<lista-de-variáveis>* .

Exemplo 3

```
⋮
parâmetros de entrada grau, coeficientes
parâmetros de saída ordenada
⋮
leia grau, coeficientes
⋮
escreva ordenada
⋮
```

Estruturas condicionais

- Alterar o fluxo natural de comandos.
- Escolher comandos quando condição for ou não satisfeita.
- Condição representada por expressão lógica.

Estrutura condicional simples

```
se <condição> então
  <comandos>
fim se
```

.

- Lista de <*comandos*> executada se, e somente se, <*condição*> tiver resultado `verdadeiro`.

Exemplo 4

```
⋮
se peso ≠ 0 então
  delta ← raiz2(peso + b)
  c ← cos(peso)
fim se
⋮
```

Estrutura condicional composta

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

.

- Se resultado de <*condição*> for `verdadeiro` então <*comandos_1*> será executada e <*comandos_2*> não será executada.
- Se resultado de <*condição*> for `falso` então <*comandos_2*> será a única executada.

Exemplo 5

```
⋮
se temperatura > 0 então
  abstemp ← temperatura
  varia ← linear ∗ abstemp
senão
  abstemp ← −temperatura
fim se
⋮
```

Estruturas de repetição

- Seqüência de comandos executada repetidamente até que a condição de interrupção seja satisfeita.

Número indefinido de repetições

```
repita
  <comandos_1>
  se <condição> então
    interrompa
  fim se
  <comandos_2>
fim repita
<comandos_3>
```

.

- `interrompa` transfere fluxo de execução para comando seguinte ao `fim repita` (<*comandos_3*>).
- <*comandos_1*> e <*comandos_2*> serão repetidos até que <*condição*> tenha resultado `verdadeiro`.

Exemplo de `repita`–`fim repita`

Exemplo 6

```
⋮
i ← 0
repita
  i ← i + 1
  se i > 5 então
    interrompa
  fim se
  raiz ← raiz_2(i)
  escreva i, raiz
fim repita
j ← 10
⋮
```

- Calculadas as raízes quadradas de $i = 1, 2, \ldots, 5$.

Número definido de repetições

```
para <cont> ← <v-ini> até <v-fin> passo <del> faça
  <comandos>
fim para
```

- Inicialmente: <*cont*>←<*v-ini*>.
- Se <*cont*> for maior que <*v-fin*> então não executa <*comandos*>.
- Se não for maior, então <*comandos*> serão executados.
- Variável <*cont*> será incrementada de <*del*>.
- Verificar se <*cont*> é maior que <*v-fin*>.
- Se não for maior então <*comandos*> serão executados e assim sucessivamente.
- Repetições se processam até que <*cont*> seja maior que <*v-fin*>.
- Quando <*del*> for 1 então `passo` <*del*> pode ser omitido.

Exemplo de `para−faça`

Exemplo 7

```
⋮
para x ← 1 até 9 passo 2 faça
  y ← 2^x
  escreva x, y
fim para
⋮
```

- Geram tabela x, 2^x, com x = 1, 3, 5, 7 e 9.

Falha no algoritmo

- Indicar que haverá falha evidente na execução do algoritmo

  `abandone`.

- Por exemplo, uma divisão por zero, uma singularidade da matriz ou uso inapropriado de parâmetros.

- Execução será cancelada.

Exemplo de algoritmo

```
Algoritmo Raiz2
{ Objetivo: Calcular raiz quadrada pelo processo babilônico }
parâmetros de entrada a, toler
  { valor para se 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
  fim se
  { cálculo do valor inicial x_0 = z }
  c(1) ← 1,01865; c(2) ← −2,17822
  c(3) ← 2,06854; c(4) ← 0,10112
  p ← 1; b ← a
  se a > 1 então
    repita
      b ← b * 0,01; p ← p * 10
      se b ≤ 1 então  interrompa  fim se
    fim repita
  fim se
  se a < 0,01 então
    repita
      b ← b * 100; p ← p * 0,1
      se b ≥ 0,01 então  interrompa  fim se
    fim repita
  fim se
  z ← c(1)
  para i ← 2 até 4 faça
    z ← z * b + c(i)
  fim para
  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  fim se
    z ← x
  fim repita
  { teste de convergência }
  se delta ≤ toler então
    raiz ← x
  senão
    escreva "processo não convergiu com 50 iterações"
  fim se
fim algoritmo
```

Complexidade computacional

- Função de complexidade F para medir o custo de execução de um programa.
- $F(n)$ pode ser medida do tempo para executar o algoritmo que resolve um problema de tamanho n.
- $F(n)$ pode ser espaço de memória requerido para a execução.
- Complexidade computacional de algoritmo se refere à estimativa do esforço computacional despendido para resolver o problema.
- Complexidade é medida pelo número necessário de operações aritméticas e lógicas:
  número de adições e multiplicações efetuadas para resolver um sistema linear de ordem n.

Notação matemática

Norma-2 vetorial

- A norma-2 ou norma Euclidiana de um vetor x de tamanho n é definida pela expressão

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

- Notação matemática $\longrightarrow$ notação algorítmica.

```
Algoritmo Norma2
{ Objetivo: Calcular norma-2 de 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
  fim para
  n2 ← raiz_2(soma)
fim algoritmo
```

Polinômios de Lagrange

- Polinômio interpolador de Lagrange de grau n

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

- Expandindo resulta a Expressão 1

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

Algoritmo para Expressão 1

```
Algoritmo Lagrange_Expressão_1
{ Objetivo: Interpolar usando 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)))
      fim se
    fim para
    r ← r + p
  fim para
fim algoritmo
```

Complexidade computacional

Operações	Complexidade
Adições	$2n^2 + 3n + 1$
Multiplicações	$n^2 + n$
Divisões	$n^2 + n$

- n: grau do polinômio de Lagrange.

Polinômios de Lagrange

- Polinômio interpolador de Lagrange de grau n

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

- Expandindo resulta a Expressão 2

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

Algoritmo para Expressão 2

```
Algoritmo Lagrange_Expressão_2
{ Objetivo: Interpolar usando 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
    c ← 1; d ← 1
    para j ← 1 até m faça
      se i ≠ j então
        c ← c * (z − x(j)); d ← d * (x(i) − x(j))
      fim se
    fim para
    r ← r + y(i) * c/d
  fim para
fim algoritmo
```

Complexidade computacional

Operações	Complexidade
Adições	$2n^2 + 3n + 1$
Multiplicações	$2n^2 + 3n + 1$
Divisões	$n + 1$

- n: grau do polinômio de Lagrange.

Tipos de erros

- Surgem várias fontes de erros que podem alterar profundamente os resultados.
- Conhecer as causas desses erros para minimizar as suas conseqüências.

Erro de truncamento

- Devido à aproximação de uma fórmula por outra.
- Para avaliar uma função matemática no computador somente as quatro operações aritméticas podem ser requeridas.
- Aproximar $f(x) = \text{sen}(x)$ por uma série

$$\text{sen}(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!},$$

$$\text{sen}(x) = x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \ldots, 0 \leq x \leq \frac{\pi}{4}.$$

$\sum_{n=0}^{t} (-1)^n \frac{x^{2n+1}}{(2n+1)!} - \text{sen}(x)$			
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

- Erro absoluto definido como

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

- Tamanho do erro absoluto é mais grave quando o valor verdadeiro for pequeno: 1711,321 $\pm$ 0,030 é exato com cinco dígitos significativos enquanto que 0,001 $\pm$ 0,030 tem pouco significado.

- Erro relativo definido como

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

  sendo indefinido para valor real nulo.

- Vantagem sobre erro absoluto: independência da magnitude dos valores.

Erro na modelagem

- Na modelagem de um problema real pode se fazer necessário o uso de dados obtidos por medidas experimentais.
- Pode ocorrer uma modelagem incorreta na qual a expressão matemática não reflete perfeitamente o fenômeno físico.
- Os dados terem sido obtidos com pouca exatidão.
- Se faz necessário a realização de testes para verificar o quanto os resultados são sensíveis às alterações dos dados fornecidos.
- Mudanças grandes nos resultados devido à pequenas variações nos dados são sintomas de um mal-condicionamento do modelo proposto.
- Uma nova modelagem do fenômeno é a tentativa de cura do problema.

Erro grosseiro

- A possibilidade de um computador cometer um erro é muito pequena.
- 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.
- Isto 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 não pode ser representado exatamente em um computador.
- Ele tem que ser convertido para a base 2 e armazenado em um número finito de *bits*.
- Erro de arredondamento é causado por esta imperfeição na representação de um número.
- Para analisar as causas e conseqüências desse tipo de erro precisa-se conhecer aritmética de ponto flutuante.

Aritmética de ponto flutuante

- Causas do erro de arredondamento.
- Representação com ponto fixo: 12,34.
- Com ponto flutuante: $0{,}1234 \times 10^2$.
- Forma geral de representação de um número

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

  d_i's são os dígitos da parte fracionária, tais que $0 \leq d_i \leq B-1$, $d_1 \neq 0$, B é o valor da base, p é o número de dígitos e e é um expoente inteiro.
- Um número de ponto flutuante tem três partes: o sinal, a parte fracionária chamada de significando ou mantissa e o expoente.
- As três partes tem um comprimento total fixo que depende do computador e do tipo de número: precisão simples, dupla ou estendida.

Computador hipotético

- Computador hipotético com dois dígitos ($p = 2$), base $B = 2$ e expoente na faixa $-1 \leq e \leq 2$.
- Número é normalizado: $d_1 \neq 0$,

  $\pm .10_2 \times 2^e$ ou $\pm .11_2 \times 2^e$, $-1 \leq e \leq 2$.

- Conversão de binário para decimal

  $.10_2 = 1 \times 2^{-1} + 0 \times 2^{-2} = 1/2$ e

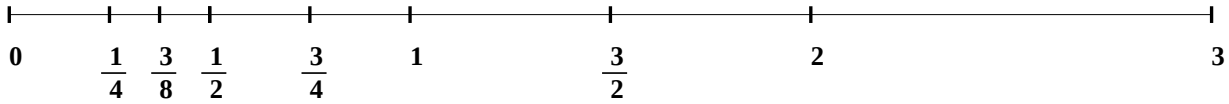
  $.11_2 = 1 \times 2^{-1} + 1 \times 2^{-2} = 3/4$,

- Únicos números positivos representáveis

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

Números discretos

- O zero é representado de uma forma especial: todos os dígitos d_i do significando e do expoente são nulos.
- Os números de ponto flutuante são discretos e não contínuos como um *número real* definido na Matemática



- 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.

Números discretos cont.

- Representação binária

  $0{,}6_{10} = 0{,}100110011001\ldots_2$ e

  $0{,}7_{10} = 0{,}1011001100110\ldots_2$.

- Os dois números serão representados igualmente como $.10_2 \times 2^0$.
- Tanto $0{,}6_{10}$ quanto $0{,}7_{10}$ serão vistos como $0{,}5_{10}$ pelo computador.
- Esta é uma grande causa de erro de arredondamento nos processos numéricos.

Formato IEEE de ponto flutuante

- A forma de representação de um número de ponto flutuante depende do fabricante do computador.
- Um mesmo programa implementado em computadores que utilizam formatos diferentes podem fornecer resultados diferentes.
- Formato proposto pelo IEEE
  (*Institute of Electrical and Electronics Engineers*)

propriedade	precisão		
	simples	dupla	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

- *overflow* e *underflow*.

Precisão das operações numéricas

- Computador hipotético com dois dígitos ($p = 2$), base $B = 10$, e expoente na faixa $-5 \leq e \leq 5$: $\pm.d_1 d_2 \times 10^e$.
- Quando dois números são somados ou subtraí-dos, os dígitos do número de expoente menor devem ser deslocados de modo a alinhar as casas decimais.
- O resultado é arredondado para dois dígitos para caber na mantissa de tamanho $p = 2$.
- O expoente é ajustado de forma a normalizar a mantissa ($d_1 \neq 0$).

Exemplo: somar 4,32 e 0,064

- Os números são armazenados no formato especificado.
- As casas decimais são alinhadas.
- A operação de adição é efetuada.
- O resultado é arrendondado para dois dígitos

$$4{,}32 + 0{,}064 = .43 \times 10^{1} + .64 \times 10^{-1} =$$

$$\begin{array}{lll} & .43 & \times 10^{1} \\ + & .0064 & \times 10^{1} \\ = & .4364 & \times 10^{1} \\ \rightarrow & .44 & \times 10^{1}. \end{array}$$

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

Exemplo: subtrair 371 de 372

- Os números são armazenados no formato especificado.
- Resulta em um mesmo valor no caso.
- A operação de subtração é efetuada.
- O resultado é convertido para zero

$$372 - 371 = .37 \times 10^3 - .37 \times 10^3 =$$

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

- 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: somar 691 e 2,71

- Os números são armazenados no formato especificado.
- As casas decimais são alinhadas.
- A operação de adição é efetuada.
- O resultado é arrendondado para dois dígitos

$$691 + 2{,}71 = .69 \times 10^3 + .27 \times 10^1 =$$

$$\begin{array}{rll} & .69 & \times 10^3 \\ + & .0027 & \times 10^3 \\ = & .6927 & \times 10^3 \\ \rightarrow & .69 & \times 10^3. \end{array}$$

- 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: multiplicar 1234 por 0,016

- Os números são armazenados no formato definido.
- A operação de multiplicação é efetuada utilizando $2p = 4$ dígitos na mantissa.
- O resultado é arrendondado para dois dígitos e normalizado

$$1234 \times 0{,}016 = .12 \times 10^4 \times .16 \times 10^{-1} =$$

$$\begin{array}{rll} & .12 & \times 10^4 \\ \times & .16 & \times 10^{-1} \\ = & .0192 & \times 10^3 \\ \rightarrow & .19 & \times 10^2. \end{array}$$

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

Exemplo: multiplicar 875 por 3172

- Os números são armazenados no formato indicado.
- A operação de multiplicação é efetuada utilizando $2p = 4$ dígitos.
- O resultado é arrendondado, normalizado e como o expoente $e = 7 > 5$ então ocorre um *overflow*

$$875 \times 3172 = .88 \times 10^3 \times .32 \times 10^4 =$$

$$\begin{array}{rll} & .88 & \times 10^3 \\ \times & .32 & \times 10^4 \\ = & .2816 & \times 10^7 \\ \rightarrow & overflow. & \end{array}$$

- A multiplicação resultou em um valor maior que esse computador hipotético pode representar.

Exemplo: dividir 0,00183 por 492

- Os números são armazenados no formato especificado.
- A operação de divisão é efetuada utilizando $2p = 4$ dígitos na mantissa.
- O resultado é arrendondado para dois dígitos e normalizado

$$0{,}00183 \div 492 = .18 \times 10^{-2} \div .49 \times 10^{3} =$$

$$\begin{array}{lll} & .18 & \times 10^{-2} \\ \div & .49 & \times 10^{3} \\ = & .3673 & \times 10^{-5} \\ \rightarrow & .37 & \times 10^{-5}. \end{array}$$

- O erro relativo desse resultado foi de aproximadamente 0,52%.

Exemplo: dividir 0,0064 por 7312

- Os números são armazenados no formato definido.
- A divisão é efetuada utilizando $2p = 4$ dígitos na mantissa.
- O resultado é arrendondado, normalizado e sendo o expoente $e = -6 < -5$ então ocorre um *underflow*

$$0{,}0064 \div 7312 = .64 \times 10^{-2} \div .73 \times 10^{4} =$$

$$\begin{array}{lll} & .64 & \times 10^{-2} \\ \div & .73 & \times 10^{4} \\ = & .8767 & \times 10^{-6} \\ \rightarrow & \mathit{underflow}. & \end{array}$$

- O resultado da divisão foi um valor menor que esse computador hipotético pode armazenar.
- Sem considerar o zero que tem uma representação especial.

Conversão de base

- Erro devido à conversão de base.
- Um número é fornecido ao computador na base 10 e armazenado na base 2.
- Números inteiros têm representação binária exata $44_{10} = 101100_2$.
- Número com decimais pode resultar em número binário com infinitos dígitos ($0{,}4_{10} = 0{,}01100110\ldots_2$).
- Os dígitos têm que ser arrendondados para armazenamento em formato de ponto flutuante.