Algoritmos Numéricos 2^a edição

Capítulo 1: Computação numérica

Capítulo 1: Computação numérica

Computação numérica

- Elaboração de algoritmo $\longrightarrow$ resultados numéricos.
- Cálculo Numérico: resolver problemas matemáticos usando computador.
- Solução via Cálculo Numérico.
- Operações aritméticas: adição, subtração, multiplicação e divisão.
- Operações lógicas: comparação, conjunção, disjunção e negação.
- Supercomputadores estão dedicados a realizar cálculos numéricos.

Etapas na solução de um problema

A solução de um problema pode ser obtida em quatro etapas:

1. Definição do problema.
2. Modelagem matemática.
3. Solução numérica.
4. Análise dos resultados.

Definição do problema

- Define-se o *problema real* a ser resolvido.
- Por exemplo, calcular

$$\sqrt{a},\ a > 0$$

usando as quatro operações aritméticas.

Modelagem matemática

- Formulação matemática transforma o *problema real* no *problema original*

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

- O *problema original* pode possuir mais soluções que o *problema real*

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

Solução numérica

- Escolha do método numérico para resolver o *problema original*.
- Método descrito por um algoritmo.
- Algoritmo implementado em uma linguagem de programação.
- Solução numérica dividida em três fases:
  1. elaboração do algoritmo,
  2. codificação do programa e
  3. 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.

- Não implementar um método diretamente em uma linguagem de programação.
- Descrever o método por meio de uma notação algorítmica.
- Abstrair dos detalhes da linguagem de programação.
- Concentrar apenas nos aspectos matemáticos do método.
- Facilitar sua implementação em qualquer linguagem de programação.

Codificação do programa

- Implementar o algoritmo na linguagem de programação escolhida.
- Preocupar com os detalhes de implementação da linguagem adotada.

Processamento do programa

- Editar código do programa em um arquivo.
- Executar código no computador.
- Se detectar erro de sintaxe: corrigir para que o programa possa ser executado.
- Se detectar erro de lógica: retornar à fase de elaboração para corrigir o algoritmo.

Exemplo de solução numérica

- Método de Newton para calcular raiz de $f(x) = x^2 - a = 0$,

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

- Substituindo $f(x)$ e $f'(x)$

$$x_{k+1} = x_k - \frac{x_k^2 - a}{2x_k} = x_k - \frac{x_k}{2} + \frac{a}{2x_k} \longrightarrow x_{k+1} = \left(x_k + \frac{a}{x_k}\right) \times 0{,}5.$$

- Proposto pelos matemáticos babilônicos.
- Calcular $\sqrt{9}$, 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: obter um novo *problema original*.
- Para valor inicial $x_0 = -1$ (ou qualquer $x_0 < 0$)

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

- Solução de modelos matemáticos podem produzir resultados sem sentido físico ou químico: tempo negativo, concentração complexa etc.
- Análise dos resultados discerne qual é a solução válida.

Notação algorítmica

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

Estrutura do algoritmo

- Iniciar

Algoritmo <***nome-do-algoritmo***>

- Terminar

fimalgoritmo

- Descrever finalidade

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

- Dados necessários para execução do algoritmo

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

- Valores calculados pelo algoritmo

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

Estrutura básica de um algoritmo

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

Variáveis e comentários

- Variável corresponde a uma posição de memória onde está armazenado um determinado valor.
- Variáveis são representadas por identificadores.
- Cadeias de caracteres alfanuméricos.
- Elementos de vetores e matrizes referenciados por subscritos ou índices.

$$v_i \text{ ou } v(i)$$

$$m_{ij} \text{ ou } m(i,j).$$

- Comentário é um texto inserido no algoritmo para aumentar a sua clareza.
- Texto deve ser delimitado por chaves { <texto> }.

{ cálculo da raiz }.

Expressões e comando de atribuição

- Existem três tipos de expressões
  1. aritméticas,
  2. lógicas e
  3. literais.
- Dependem dos tipos dos operadores e das variáveis envolvidas.

Expressões aritméticas

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

$$\sqrt{b^2 - 4 * a * c},$$

$$\cos(2 + x),$$

$$massa * velocidade.$$

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

$$\boxed{<variável> \leftarrow <expressão>}$$

- Por exemplo,

$$velocidade \leftarrow deslocamento/tempo.$$

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	$\text{sinal}(x) = 1$ 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

Expressões lógicas

- Operadores são lógicos e operandos são relações e/ou variáveis do tipo lógico.
- Relação é uma 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 uma relação ou de uma expressão lógica: **verdadeiro** ou **falso**.

Exemplo 1 Para $c = 1$ e $d = 3$, então $c \leq d$ é **verdadeiro**.
Para $x = 2$, $y = 3$ e $z = 10$, então $x + y = z$ é **falso**.

Operadores lógicos

- Permitem a combinação ou negação das relações lógicas.

Operador lógico	Uso
e	conjunção
ou	disjunção
não	negação

- Resultados obtidos com os operadores lógicos:
  V = **verdadeiro** e o de F = **falso**.

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

a **ou** b		
$a \backslash b$	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** $x + y + 5 = z)$ é V **e** V $\longrightarrow$ **verdadeiro**.
$(d = c$ **ou** $x + y = z)$ é F **ou** F $\longrightarrow$ **falso**.

Expressões literais

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

$$\textit{mensagem} \leftarrow \text{“matriz singular”}.$$

Comandos de entrada e saída

- Leitura em dispositivo externo

leia <*lista-de-variáveis*>

- Escrita em dispositivo externo

escreva <*lista-de-variáveis*>

Exemplo 3 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*
  $Celsius \leftarrow (Fahrenheit - 32) * 5/9$
  escreva *Fahrenheit*, *Celsius*
fimalgoritmo

Estruturas condicionais

- Alterar o fluxo natural de comandos.
- Escolher comandos quando a condição for ou não satisfeita.
- Condição representada por expressão lógica.
- Estruturas condicionais podem ser simples ou compostas.

Estrutura condicional simples

se $<$*condição*$>$ **então**
  $<$*comandos*$>$
fimse

- Lista de $<$*comandos*$>$ será executada se, e somente se, a expressão lógica $<$*condição*$>$ tiver como resultado o valor **verdadeiro**.

Exemplo 4 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 $\leftarrow \log_{10}(x)$
    escreva *x*, *LogDec*
  fimse
fimalgoritmo

Estrutura condicional composta

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

- Se resultado de *<condição>* for **verdadeiro**, então a seqüência *<comandos_1>* será executada e a seqüência *<comandos_2>* não será.
- Se o resultado de *<condição>* for **falso**, então será a lista *<comandos_2>* a única a ser executada.

Exemplo 5 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
```

Estruturas de repetição

- Faz uma seqüência de comandos ser executada repetidamente até que uma dada condição de interrupção ser satisfeita.
- Dois tipos: dependendo do número de repetições ser indefinido ou definido.

Número indefinido de repetições

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

- Comando **interrompa** faz com que o fluxo de execução seja transferido para o comando imediatamente a seguir do **fimrepita**.
- As listas *<comandos_1>* e *<comandos_2>* serão repetidas até que a expressão lógica *<condição>* resulte no valor **verdadeiro**:
  - A repetição será interrompida (*<comandos_2>* não será executada).
  - A lista *<comandos_3>*, após ao **fimrepita**, será executada.
- Caso geral de uma estrutura de repetição.

Exemplo para determinar a precisão da máquina ε

Exemplo 6 Escrever um algoritmo para determinar o maior número de ponto flutuante que, somado a 1, seja igual a 1.

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

Número definido de repetições

para $<$*controle*$>$ $\leftarrow$ $<$*valor-inicial*$>$ **até** $<$*valor-final*$>$ **passo** $<$*delta*$>$ **faça**
$<$*comandos*$>$
fimpara

- Usado quando souber com antecedência quantas vezes a estrutura deve ser repetida.
- Quando o incremento $<$*delta*$>$ tiver o valor 1, então o **passo** $<$*delta*$>$ pode ser omitido.

Exemplo para verificar que $\sum_{i=1}^{n}(2i-1) = n^2$

Exemplo 7 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 $\leftarrow$ *0*
  para *i* $\leftarrow$ *1* **até** *2* $*$ *n* $-$ *1* **passo** *2* **faça**
    Soma $\leftarrow$ *Soma* $+$ *i*
  fimpara
  escreva *Soma*, n^2
fimalgoritmo

Falha no algoritmo

abandone

- Indica 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.
- A execução será cancelada.

Algoritmo para cálculo da média aritmética e desvio padrão

Exemplo 8 Dado um vetor x com n componentes, o algoritmo calcula a média aritmética $\bar{x}$ e o desvio padrão s 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*
  $Soma \leftarrow 0$
  $Soma2 \leftarrow 0$
  para $i \leftarrow 1$ **até** n **faça**
    $Soma \leftarrow Soma + x(i)$
    $Soma2 \leftarrow Soma2 + x(i)^2$
  fim para
  $Média \leftarrow Soma/n$
  $DesvioPadrão \leftarrow \mathrm{raiz}_2((Soma2 - Soma^2/n)/(n-1))$
  escreva *Média*, *DesvioPadrão*
fim algoritmo

$\Vdash$

Algoritmo para determinar o maior elemento da linha de uma matriz

Exemplo 9 Algoritmo para determinar o maior elemento em cada linha de uma matriz A de dimensão $m \times n$.

```
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
```

Algoritmo para calcular o valor de π

Exemplo 10 Algoritmo para calcular o valor de π, com precisão dada, utilizando a série

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

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
    fim se
    Sinal ← −*Sinal*
    Denominador ← *Denominador* + *2*
  fim repita
  pi ← *4* ∗ *Soma*
fim algoritmo

⊫

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

Exemplo 11 O polinômio de quadrados mínimos que aproxima $\sqrt{x}$ para $0{,}01 \leq x \leq 1$ é, pelo processo de Horner,

$$P(x) = ((1{,}01865x - 2{,}17822)x + 2{,}06854)x + 0{,}10113.$$

```
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
```

Cálculo de raiz quadrada pelo processo babilônico

Exemplo 12 Algoritmo para calcular $\sqrt{a},\ a > 0$.

```
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
```

Notação matemática

- Definir a modelagem matemática por meio de expressões aritméticas e lógicas.
- Passar dessa notação matemática para a notação algorítmica proposta.
- Esta passagem será ilustrada por meio de alguns exemplos.

Norma-2 de um vetor x de tamanho n

Exemplo 13 Algoritmo para calcular a norma-2 ou norma Euclidiana

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

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
  fim para
  N2 ← raiz$_2$(*Soma*)
fim algoritmo

Norma-∞ de um vetor x de tamanho n

Exemplo 14 Algoritmo para achar a norma-∞ ou norma de máxima magnitude

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

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

Produto matriz-vetor

Exemplo 15 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 $i \leftarrow 1$ **até** n **faça**
    $Soma \leftarrow 0$
    para $j \leftarrow 1$ **até** m **faça**
      $Soma \leftarrow Soma + A(i,j) * v(j)$
    fim para
    $x(i) \leftarrow Soma$
  fim para
fim algoritmo

Complexidade computacional

- Definir função de complexidade para medir o custo de execução de programa.
- Esta função pode ser
  - medida do tempo para executar o algoritmo;
  - espaço de memória requerido para esta execução.
- Complexidade computacional de um algoritmo se refere à estimativa do esforço computacional despendido para resolver o problema.
- É medida pelo número necessário de operações aritméticas e lógicas.
- Por exemplo, o número de adições e multiplicações efetuadas para resolver um sistema linear de ordem n.

Complexidade de tempo

- Tipos de algoritmos:
  - Polinomiais: função de complexidade da forma

$$O(c_p n^p + c_{p-1} n^{p-1} + \ldots + c_1 n + c_0).$$

  - Exponenciais: função de complexidade tem a forma

$$O(c^n),\ c > 1.$$

- As operações aritméticas demandam diferentes tempos para serem executadas.

Função de complexidade

- Função de complexidade será definida para cada operação.
- Número de adições para fazer a decomposição LU de uma matriz de ordem n

$$O\left(\frac{1}{3}n^3 - \frac{1}{2}n^2 + \frac{1}{6}n\right)$$

ou

$$O(n^3).$$

- Número de multiplicações para resolver um sistema triangular inferior de ordem n pelas substituições sucessivas

$$O\left(\frac{1}{2}n^2 - \frac{1}{2}n\right)$$

ou

$$O(n^2).$$

Análise de complexidade

- Seja o polinômio interpolador de Lagrange de grau n

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

- Expressão 1

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

- Número de pontos m usados na interpolação é igual a $n+1$, onde n é o grau do polinômio.

Algoritmo da Expressão 1

```
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
```

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

Expressão 2

- Polinômio de Lagrange de grau n

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

- Expressão 2

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

Algoritmo da Expressão 2

```
Algoritmo Polinômio_Lagrange
{ Objetivo: Interpolar valor em tabela 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
    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))
      fimse
    fimpara
    r ← r + y(i) ∗ c/d
  fimpara
fimalgoritmo
```

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

Multiplicações $=$ Adições;

Divisões: $\sum_{i=1}^{m} 1 = m = n+1.$

Comparação das complexidades

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

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

- Número de adições é o mesmo.
- Número de multiplicações é da mesma ordem (n^2).
- Número de divisões da 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.

Implementação de algoritmos

- Fases da solução numérica:
    1. elaboração do algoritmo,
    2. codificação do programa e
    3. processamento do programa.
- Proposta de notação algorítmica.
- Elaborar um algoritmo a partir de uma formulação matemática.
- Próxima fase: codificação do programa na linguagem escolhida.
- Linguagens de programação FORTRAN, Pascal e MATLAB.

Programa para determinar a precisão de máquina

Implementar em FORTRAN o algoritmo para determinar a precisão de máquina ϵ 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 igual a 2^{-53}
  `Precisao da maquina: 1.11022302E-16`
- Se for utilizada variável real de 4 bytes, o resultado será igual a 2^{-24}
  `Precisao da maquina: 5.96046448E-08`
- Se qualquer número menor ou igual à precisão da máquina for somado a 1, dará o resultado igual a 1.

Programa para calcular média e desvio padrão

Implementar o algoritmo em Pascal.

```
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.
```

Programa para calcular π

Implementar o algoritmo em MATLAB.

```
%   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;
```

Tipos de erros

- Surgem erros de várias fontes durante as etapas de solução de um problema.
- Esses erros podem alterar profundamente os resultados obtidos.
- É importante conhecer as causas desses erros para minimizar as suas conseqüências.

Erro de truncamento

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

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

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

- Quando t aumenta, o erro de truncamento diminui.
- Estes erros são devidos aos truncamentos da série.

Erro absoluto e relativo

- Erro absoluto:

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

- 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.
- Erro relativo

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

sendo indefinido para valor real nulo.

- Vantagem sobre o erro absoluto é a independência da magnitude dos valores.

Erro na modelagem

- Na modelagem matemática de um problema real pode ser 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.
- Também os dados podem ter sido obtidos com pouca exatidão.
- Se faz necessária a realização de testes para verificar o quanto os resultados são sensíveis às alterações dos dados.
- Mudanças grandes nos resultados devido a pequenas variações nos dados são sintomas de um malcondicionamento 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, por exemplo $0{,}4_{10}$ (0,4 na base 10), 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.
- Número representado com ponto fixo: 12,34.
- Ponto flutuante: $0{,}1234{\times}10^2$.
- Forma geral de representação de um número de ponto flutuante

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

  – 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 (geralmente 2, 10 ou 16),
  – 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 têm 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 normalizado: $d_1 \neq 0$,

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

- Conversão de binário para decimal de um número menor que 1,

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

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

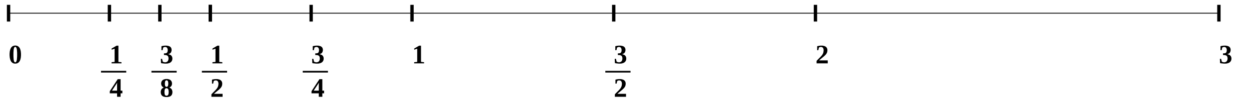
- únicos números positivos representáveis neste computador

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

- O zero é representado de uma forma especial: todos os dígitos d_i do significando e do expoente são nulos.

Números discretos

- 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.
- Representação binária

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

- No computador hipotético eles 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 pode fornecer resultados diferentes.
- Formato 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 $e = -5, \ldots, 5$: $\pm .d_1 d_2 \times 10^e$.
- 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 arredondado para dois dígitos para caber na mantissa de tamanho $p = 2$.
- O expoente é ajustado de forma a normalizar a mantissa $(d_1 \neq 0)$.

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

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

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

Subtrair 371 de 372

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

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

- 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.

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.

Multiplicar 1234 por 0,016

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

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

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

Multiplicar 875 por 3172

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

$$875 \times 3172 = .88{\scriptscriptstyle\times}10^3 \times .32{\scriptscriptstyle\times}10^4 = \begin{array}{rll} & .88 & {\scriptscriptstyle\times}10^3 \\ \times & .32 & {\scriptscriptstyle\times}10^4 \\ = & .2816 & {\scriptscriptstyle\times}10^7 \\ \to & \textit{overflow}. & \end{array}$$

- A multiplicação resultou em um valor maior do que este computador pode representar.

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

$$0{,}00183 \div 492 = .18_{\times}10^{-2} \div .49_{\times}10^{3} = \begin{array}{rll} & .18 & {}_{\times}10^{-2} \\ \div & .49 & {}_{\times}10^{3} \\ = & .3673 & {}_{\times}10^{-5} \\ \to & .37 & {}_{\times}10^{-5}. \end{array}$$

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

Dividir 0,0064 por 7312

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

$$0{,}0064 \div 7312 = .64_{\times}10^{-2} \div .73_{\times}10^{4} = \begin{array}{lll} & .64 & _{\times}10^{-2} \\ \div & .73 & _{\times}10^{4} \\ = & .8767 & _{\times}10^{-6} \\ \rightarrow & \textit{underflow}. & \end{array}$$

- O resultado da divisão foi um valor menor que este computador 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 um 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.

Capítulo 1: Computação numérica

`Fim`