

Minicurso de C/C++ para Matemática

Pedro H A Konzen

19 de outubro de 2023

Conteúdo

1	Licença	2
2	Sobre a Linguagem	2
2.1	Instalação e Execução	2
2.1.1	IDE	2
2.2	Olá, mundo!	3
3	Elementos da Linguagem	3
3.1	Tipos de Dados Básicos	3
3.2	Operações Aritméticas Elementares	5
3.3	Funções e Constantes Elementares	7
3.4	Operadores de Comparação Elementares	8
3.5	Operadores Lógicos Elementares	9
3.6	Arranjos	10
4	Elementos da Programação Estruturada	11
4.1	Métodos/Funções	11
4.2	Ramificação	14
4.3	Repetição	16
4.3.1	<code>while</code>	16
4.4	<code>do ... while</code>	18
4.4.1	<code>for</code>	19
	Referências Bibliográficas	20

1 Licença

Este trabalho está licenciado sob a Licença Atribuição-CompartilhaIgual 4.0 Internacional Creative Commons. Para visualizar uma cópia desta licença, visite http://creativecommons.org/licenses/by-sa/4.0/deed.pt_BR ou mande uma carta para Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

2 Sobre a Linguagem

C e C++ são **linguagens de programação compiladas de propósito geral**. A primeira é **estruturada e procedural**, tendo sido criada em 1972 por Dennis Ritchie¹. A segunda foi inicialmente desenvolvida por Bjarne Stroustrup² como uma extensão da primeira. Em sua mais recente especificação, a **linguagem C++ se caracteriza por ser multi-paradigma (imperativa, orientada a objetos e genérica)**.

2.1 Instalação e Execução

Códigos C/C++ precisam ser compilados antes de serem executados. De forma simplificada, o **compilador** é um programa que interpreta e converte o código em um programa executável em computador. Há vários compiladores gratuitos disponíveis na web. Ao longo deste minicurso, usaremos a coleção de compiladores **GNU GCC** instalados em sistema operacional **Linux**.

2.1.1 IDE

Usar um **ambiente integrado de desenvolvimento (IDE, em inglês, *integrated development environment*)** é a melhor forma de capturar o melhor das linguagens C/C++. Algumas alternativas são:

- [Eclipse](#)
- [GNU Emacs](#)
- [VS Code](#)

¹Dennis Ritchie, 1941-2011, cientista da computação estadunidense. Fonte: [Wikipédia](#).

²Bjarne Stroustrup, 1950, cientista da computação dinamarquês. Fonte: [Wikipédia](#).

2.2 Olá, mundo!

Vamos **implementar** nosso primeiro programa C/C++. **Em geral, são três passos: 1. escrever; 2. compilar; 3. executar.**

1. **Escrever o código.**

Em seu IDE preferido, digite o código:

Código 1: ola.cc

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Olá, mundo!\n");
6     return 0;
7 }
```

2. **Compilar.**

Para compilá-lo, digite no terminal de seu sistema operacional

```
1 $ gcc ola.cc -o ola.x
```

3. **Executar.**

Terminada a compilação, o arquivo executável `ola.x` é criado. Para executá-lo, digite

```
1 $ ./ola.x
```

3 Elementos da Linguagem

3.1 Tipos de Dados Básicos

Na linguagem C/C++, **dados são alocados em variáveis com tipos declarados³.**

Exemplo 3.1. Consideramos o seguinte código.

³Consulte [Wikipedia: C data type](#) para uma lista dos tipos de dados disponíveis na linguagem

Código 2: dados.cc

```
1  /* dados.cc
2     Exemplo de alocação de variáveis.
3  */
4  #include <stdio.h>
5
6  int main()
7  {
8     // var inteira
9     int i = 1;
10    // var pto flutuante
11    double x;
12
13    x = 2.5;
14    char s[6] = "i + x";
15    double y = i + x;
16    printf("%s = %f\n", s, y);
17    return 0;
18 }
```

Na linha 9, é alocada uma **variável do tipo inteira** com identificador `i` e **valor** 1. Na linha 11, é alocada uma **variável do tipo ponto flutuante** (64 *bits*) com identificador `x`.

Na linha 14, é alocada uma **variável do tipo *string***⁴. Na linha 15, alocamos uma nova variável `y`.

Observação 3.1. (**Comentários e Continuação de Linha.**) Códigos C++ admitem **comentários** e **continuação de linha** como no seguinte exemplo acima. Comentários em linha podem ser feitos com `//` e de múltiplas linhas com `/* ... */`. Linhas de instruções muito compridas podem ser quebradas em múltiplas linhas com a instrução de continuação de linha `\`.

Observação 3.2. (**Notação científica.**) Podemos usar **notação científica** em C++. Por exemplo 5.2×10^{-2} é digitado da seguinte forma `5.2e-2`.

Código 3: notacaoCientifica.cpp

```
1  #include <stdio.h>
2
```

⁴Um arranjo de `char` (caracteres).

```
3 int main()
4 {
5
6     int i = -51;
7     double x = 5.2e-2;
8
9     // inteiro
10    printf("inteiro: %d\n", i);
11    // fixada
12    printf("fixada: %f\n", x);
13    // notação científica
14    printf("científica: %e\n", x);
15    return 0;
16 }
```

Exercício 3.1.1. Antes de implementar, diga qual o valor de x após as seguintes instruções.

```
1 int x = 1;
2 int y = x;
3 y = 0;
```

Justifique sua resposta e verifique-a.

Exercício 3.1.2. Implemente um código em que a(o) usuá(ri)a entra com valores para as variáveis x e y . Então, os valores das variáveis são permutados entre si. Dica: a entrada de dados por usuá(ri)a pode ser feita com o método C/C++ `scanf` da biblioteca `stdio.h`. Por exemplo,

```
1 double x;
2 scanf("%lf", &x);
```

faz a leitura de um `double` (long float) e o armazena na variável x .

3.2 Operações Aritméticas Elementares

Os operadores aritméticos elementares são⁵:

`*`, `/`, `%` : multiplicação, divisão, módulo

⁵Em ordem de precedência.

+, - : adição, subtração

Exemplo 3.2. Qual é o valor impresso pelo seguinte código?

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("%f\n", 2+17%9/2*2-1 );
6     return 0;
7 }
```

Observamos que as operações $*$, $/$ e $%$ têm precedência maior que as operações $+$ e $-$. Operações de mesma precedência seguem a ordem da esquerda para direita, conforme escritas na linha de comando. **Usa-se parênteses para alterar a precedência entre as operações**, por exemplo

```
1 printf("%f\n", (2+17)%9/2*2-1 );
```

imprime o resultado -1 . Sim, pois a **divisão inteira** está sendo usada. Para computar a divisão em ponto flutuante, um dos operandos deve ser **double**. Para tanto, podemos fazer um **casting double** `((2+17)\%9)/2*2-1` ou, simplesmente, `(2+17)\%9/2.*2-1`.

Observação 3.3. (**Precedência das Operações.**) Consulte mais informações sobre a precedência de operadores em [Wikipedia:Operators in C and C++](#).

Exercício 3.2.1. Escreva um programa para computar o vértice da parábola

$$ax^2 + bx + c = 0, \quad (1)$$

para $a = 2$, $b = -2$ e $c = 4$.

O operador $%$ módulo computa o resto da divisão inteira, por exemplo, $5\%2$ é igual a 1 .

Exercício 3.2.2. Use C/C++ para computar os inteiros não negativos q e r tais que

$$25 = q \cdot 3 + r, \quad (2)$$

sendo r o menor possível.

3.3 Funções e Constantes Elementares

A biblioteca C/C++ `math.h` disponibiliza várias funções e constantes elementares.

Exemplo 3.3. O seguinte código, imprime os valores de π , $\sqrt{2}$ e $\ln e$.

Código 4: `mat.cc`

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     printf("pi = %.9e\n", M_PI);
7     printf("2^(1/2) = %.5f\n", sqrt(2.));
8     printf("log(e) = %f\n", log(M_E));
9     return 0;
10 }
```

Observação 3.4. (Compilação e Linkagem.) A compilação de um código C/C++ envolve a linkagem de bibliotecas. A `stdio.h` é linkada de forma automática na compilação. Já, `math.h` precisa ser explicitamente linkada com

```
1 $ gcc foo.cc -lm
```

Observação 3.5. (Logaritmo Natural.) Notamos que `log` é a função logaritmo natural, i.e. $\ln(x) = \log_e(x)$. A implementação C/C++ para o logaritmo de base 10 é `log10(x)`.

Exercício 3.3.1. Compute

- a) $\sin\left(\frac{\pi}{4}\right)$
- b) $\log_3(\pi)$
- c) $e^{\log_2(\pi)}$
- d) $\sqrt[3]{-27}$

Exercício 3.3.2. Compute as raízes do seguinte polinômio quadrático

$$p(x) = 2x^2 - 2x - 4 \quad (3)$$

usando a fórmula de Bhaskara⁶.

3.4 Operadores de Comparação Elementares

Os operadores de comparação elementares são

== : igual a

!= : diferente de

> : maior que

< : menor que

>= : maior ou igual que

<= : menor ou igual que

Estes operadores retornam os **valores lógicos** **true** (verdadeiro, 1) ou **false** (falso, 0).

Por exemplo, temos

Código 5: opComp.cc

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int x = 2;
6     bool res = x + x == 5;
7     printf("2 + 2 == 5? %d", res);
8 }
```

Exercício 3.4.1. Considere a circunferência de equação

$$c: (x - 1)^2 + (y + 1)^2 = 1. \quad (4)$$

Escreva um código em que a(o) usuá(ri)a entra com as coordenadas de um ponto $P = (x, y)$ e o código verifica se P pertence ao disco determinado por c .

Exercício 3.4.2. Antes de implementar, diga qual é o valor lógico da instrução `sqrt(3) == 3`. Justifique sua resposta e verifique!

⁶Bhaskara Akaria, 1114 - 1185, matemático e astrônomo indiano. Fonte: [Wikipédia](#).

3.5 Operadores Lógicos Elementares

Os operadores lógicos elementares são:

&& : e lógico

|| : ou lógico

! : não lógico

Exemplo 3.4. (Tabela Booleana do &&.) A tabela booleana⁷ do e lógico é

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

O seguinte código, monta essa tabela booleana, verifique!

```

1 #include <stdio.h>
2
3 int main()
4 {
5     bool T = true;
6     bool F = false;
7     printf("A    | B    | A && B\n");
8     printf("%d    | %d    | %d\n", T, T, T&&T);
9     printf("%d    | %d    | %d\n", T, F, T&&F);
10    printf("%d    | %d    | %d\n", F, T, F&&T);
11    printf("%d    | %d    | %d\n", F, F, F&&F);
12 }
```

Exercício 3.5.1. Construa as tabelas booleanas do operador || e do !.

Exercício 3.5.2. Escreva um código para verificar as seguintes comparações

- $1.4 \leq \sqrt{2} < 1.5$.
- $|x| < 1$, $x = \sin(\pi/3)$.
- $|x| > \frac{1}{2}$, $x = \cos(\pi * 2)$.

⁷George Boole, 1815 - 1864, matemático britânico. Fonte: [Wikipédia](#).

Exercício 3.5.3. Considere um retângulo $r : ABDC$ de vértices $A = (1, 1)$ e $D = (2, 3)$. Crie um código em que a(o) usuá(a)ria(o) informa as coordenadas de um ponto $P = (x, y)$ e o código verifica cada um dos seguintes itens:

1. $P \in r$.
2. $P \in \partial r$.
3. $P \notin \bar{r}$.

Exercício 3.5.4. Implemente uma instrução para computar o operador `xor` (ou exclusivo). Dadas duas afirmações A e B, A `xor` B é `true` no caso de uma, e somente uma, das afirmações ser `true`, caso contrário é `false`.

3.6 Arranjos

Um **arranjo**⁸ é uma sequência de dados do mesmo tipo. Os elementos dos arranjos são indexados⁹ e mutáveis (podemos ser alterados por nova atribuição).

Exemplo 3.5. No código abaixo, alocamos o ponto $P = (2, 3)$ e o vetor $v = (2.5, \pi, -1.)$ como arranjos.

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     // P = (2, 3)
7     int P[2] = {2, 3};
8     printf("P = (%d, %d)\n", P[0], P[1]);
9
10    double v[3];
11    v[0] = 2.5;
12    v[1] = M_PI;
13    v[2] = -1.;
14    printf("v = (%lf, %lf, %lf)\n", v[0], v[1], v[2]);
15
16    return 0;
17 }
```

⁸Em inglês, *array*

⁹O índice é um inteiro não negativo, sendo o primeiro elemento indexado por 0 (zero).

Exercício 3.6.1. Escreva um código em que a(o) usuária(o) entra com um ponto $P = (x, y)$ e o programa informe se P pertence ao disco determinado pela circunferência de equação $(x - 1)^2 + y^2 = 4$. Use de um arranjo para alocar o ponto P .

Exercício 3.6.2. Considere os vetores

$$\mathbf{v} = (-1., 2., 1.) \quad (5)$$

$$\mathbf{w} = (1., -3., 2.). \quad (6)$$

Faça um código que aloca os vetores como arranjos e imprime o vetor soma $\mathbf{v} + \mathbf{w}$.

Exercício 3.6.3. Considere a matriz

$$A = \begin{vmatrix} 1. & -2. \\ 3. & 3. \end{vmatrix}. \quad (7)$$

Faça um código que aloca a matriz como um arranjo bidimensional (um arranjo de arranjos) e compute seu determinante.

4 Elementos da Programação Estruturada

C/C++ são linguagens **procedurais**¹⁰ e contém instruções para a **programação estruturada**. Neste paradigma de programação, as computações são organizadas em sequências de blocos computacionais e, um bloco inicia sua computação somente após o bloco anterior tiver terminado. Contam com estruturas de **ramificação** (seleção de blocos), **repetição** de blocos e definição de **funções/métodos** (sub-blocos computacionais).

4.1 Métodos/Funções

Um **método** (ou **função**) é um subprograma (ou subbloco computacional) que pode ser chamado/executado em qualquer parte do programa principal. Todo código C/C++ inicia-se no método `main()`, consulte o Código 1. A sintaxe de definição de um método é

¹⁰C++ também é orientada-a-objetos.

```

1 typeOut foo(typeIn0 x0, typeIn1 x1, ..., typeInN x2)
2 {
3     typeOut out;
4     statment0;
5     statment1;
6     ...;
7     statmentN;
8     return out;
9 }

```

Aqui, `typeOut` denota o tipo da saída, `foo` denota o identificador/nome do método, `typeIn0 x1`, `typeIn1 x2`, ..., `typeInN x3` são os tipos e identificadores dos parâmetros de entrada¹¹. O escopo do método é delimitado entre chaves e pode conter qualquer instrução (*statment*) C/C++. O método é encerrado¹² quando terminado seu escopo ou ao encontrar a instrução `return`. Esta instrução, também, permite o retorno de um dado do mesmo tipo da saída do método.

Exemplo 4.1. Vamos considerar a função

$$f(x) = 2x - 3. \quad (8)$$

a) No código abaixo, o método `f` computa a função e imprime seu valor¹³.

Código 6: `method.cc`

```

1 #include <stdio.h>
2
3 void f(double x)
4 {
5     double y = 2.*x - 3.;
6     printf("f(%lf) = %lf\n", x, y);
7 }
8
9 int main()
10 {
11     f(0.);
12     double x = -1.;

```

¹¹Parâmetros de entrada são opcionais

¹²No encerramento do método o código retorna ao programa principal.

¹³`void` é a instrução para “no type”.

```
13     f(x);
14     double y = 2.;
15     f(y);
16     return 0;
17 }
```

b) Nesta versão do código, o método `f` retorna o valor computado da função f e é o método principal `main` que imprime o resultado.

```
1  #include <stdio.h>
2
3  double f(double x)
4  {
5      return 2.*x - 3.;
6  }
7
8  int main()
9  {
10     double y = f(0.);
11     printf("f(%lf) = %lf\n", 0., y);
12     printf("f(%lf) = %lf\n", -1., f(-1.));
13     double z = 2.;
14     printf("f(%lf) = %lf\n", z, f(z));
15     return 0;
16 }
```

Exercício 4.1.1. Implemente uma função para computar as raízes de um polinômio de grau 1 $p(x) = ax + b$. Assuma que $a \neq 0$.

Exercício 4.1.2. Implemente uma função para computar as raízes reais de um polinômio de grau 2 $p(x) = ax^2 + bx + c$. Assuma que p tenha raízes reais.

Exercício 4.1.3. Considerando vetores em \mathbb{R}^3

$$x = (x_1, x_2, x_3), \quad (9)$$

$$y = (y_1, y_2, y_3), \quad (10)$$

implemente um código que contenha:

a) função para computação do vetor soma $\mathbf{x} + \mathbf{y}$.

b) função para computação do produto escalar $\mathbf{x} \cdot \mathbf{y}$.

Exercício 4.1.4. Implemente uma função que computa o determinante de matrizes reais 2×2 .

Exercício 4.1.5. Implemente uma função que computa a multiplicação matrix-vetor Ax , com A 2×2 e x um vetor coluna de dois elementos.

Exercício 4.1.6. (Recursividade) Implemente uma função recursiva para computar o fatorial de um número natural n , i.e. $n!$.

4.2 Ramificação

Uma estrutura de ramificação é uma instrução para a tomada de decisões durante a execução de um programa. Nas linguagens C/C++ usa-se a sintaxe

```
1 if (condition0) {  
2     block0;  
3 } else if (condition1) {  
4     block1;  
5 } else {  
6     block2;  
7 }
```

A instrução `if` permite a execução do bloco computacional `block0` somente no caso de `condition0` seja `true` (verdadeira). A instrução `else if` somente é verificada quando `condition0 == false`. Neste caso, o `block1` é executado somente se `condition1 == true`. Senão, `block2` é executado.

Exemplo 4.2. Os seguintes códigos computam os zeros da função

$$f(x) = ax + b, \quad (11)$$

para parâmetros informados por usuário(o).

a) Caso restrito a raiz real única.

```
1 #include <stdio.h>  
2  
3 int main()  
4 {  
5     double a,b;  
6     printf("a = ");
```

```
7   scanf("%lf", &a);
8   printf("b = ");
9   scanf("%lf", &b);
10
11  if (a != 0.) {
12      double x = -b/a;
13      printf("x = %lf\n", x);
14  }
15
16  return 0;
17 }
```

b) Caso de raiz real única ou múltiplas.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      double a,b;
6      printf("a = ");
7      scanf("%lf", &a);
8      printf("b = ");
9      scanf("%lf", &b);
10
11     if (a != 0.) {
12         double x = -b/a;
13         printf("x = %lf\n", x);
14     } else if ((a == 0.) && (b == 0.)) {
15         printf("Todo x real é zero da função.\n");
16     }
17
18     return 0;
19 }
```

c) Caso de raiz real única, ou múltiplas ou nenhuma.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      double a,b;
```

```
6   printf("a = ");
7   scanf("%lf", &a);
8   printf("b = ");
9   scanf("%lf", &b);
10
11  if (a != 0.) {
12      double x = -b/a;
13      printf("x = %lf\n", x);
14  }
15
16  return 0;
17 }
```

Exercício 4.2.1. Implemente um código que contenha uma função que recebe dois números n e m e imprime o maior deles.

Exercício 4.2.2. Implemente um código que contenha uma função que recebe os coeficientes de um polinômio

$$p(x) = ax^2 + bx + c \quad (12)$$

e classifique-o como um polinômio de grau 0, 1 ou 2.

Exercício 4.2.3. Implemente um código que contenha uma função para a computação das raízes de um polinômio de segundo grau.

4.3 Repetição

Estruturas de repetição são instruções que permitem a execução repetida de um bloco computacional. São três instruções disponíveis `while`, `do ... while` e `for`.

4.3.1 while

A sintaxe da instrução `while` é

```
1 while (condition) {
2     block
3 }
```

Isto é, enquanto (`while`) a expressão `condition == true`, o bloco computacional `block` é repetidamente executado. Ao final de cada execução, a condição

é novamente verificada. Quando `condition == false`, `block` não é executado e o código segue para a primeira instrução após o escopo do `while`.

Exemplo 4.3. O seguinte código computa a soma dos 10 primeiros termos da progressão geométrica

$$a_i = 2^{-i}, \quad (13)$$

para $i = 0, 1, 2, \dots$

Código 7: while.cc

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     int i = 0;
7     double s = 0.;
8     while (i < 10) {
9         s = s + pow(0.5, double(i));
10        i += 1;
11    }
12    printf("s = %lf\n", s);
13    return 0;
14 }
```

Observação 4.1. As instruções de controle `break`, `continue` são bastante úteis em várias situações. A primeira, encerra as repetições e, a segunda, pula para uma nova repetição.

Exercício 4.3.1. Use `while` para imprimir os dez primeiros números ímpares.

Exercício 4.3.2. Crie uma função para a computação da soma de dois vetores $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, com dado $n \geq 0$.

Exercício 4.3.3. Use a instrução `while` para escreva uma função que retorne o n -ésimo termo da função de Fibonacci¹⁴, $n \geq 1$.

¹⁴Leonardo Fibonacci, 1170 - 1250, matemático italiano. Fonte: [Wikipédia](#).

4.4 do ... while

Diferentemente da instrução `while`, a `do ... while` verifica a condição de repetição ao final do escopo do seu bloco computacional.

Exemplo 4.4. O seguinte código computa a soma dos 10 primeiros termos da progressão geométrica

$$a_i = 2^{-i}, \quad (14)$$

para $i = 0, 1, 2, \dots$

Código 8: doWhile.cc

```

1  #include <stdio.h>
2  #include <math.h>
3
4  int main()
5  {
6      int i = 0;
7      double s;
8      do {
9          s += pow(0.5, double(i));
10         i += 1;
11     } while (i < 10);
12     printf("s = %lf\n", s);
13     return 0;
14 }
```

Exercício 4.4.1. Uma aplicação do Método Babilônico¹⁵ para a aproximação da solução da equação $x^2 - 2 = 0$, consiste na iteração

$$x_0 = 1, \quad (15)$$

$$x_{i+1} = \frac{x_i}{2} + \frac{1}{x_i}, \quad i = 0, 1, 2, \dots \quad (16)$$

Faça um código com `while` para computar aproximação x_i , tal que $|x_i - x_{i-1}| < 10^{-5}$.

¹⁵Matemática Babilônica, matemática desenvolvida na Mesopotâmia, desde os Sumérios até a queda da Babilônia em 539 a.C.. Fonte: [Wikipédia](#).

4.4.1 for

A estrutura **for** tem a sintaxe

```
1 for (init; condition; iter) {  
2     block;  
3 }
```

onde, *init* é a instrução de inicialização, *condition* é o critério de parada, *iter* é a instrução do iterador.

Exemplo 4.5. O seguinte código computa a soma dos 10 primeiros termos da progressão geométrica

$$a_i = 2^{-i}, \quad (17)$$

para $i = 0, 1, 2, \dots$

```
1 #include <stdio.h>  
2 #include <math.h>  
3  
4 int main()  
5 {  
6     double s = 0;  
7     for (int i=0; i<10; ++i) {  
8         s += pow(2., double(-i));  
9     }  
10    printf("s = %lf\n", s);  
11    return 0;  
12 }
```

Exercício 4.4.2. Use a instrução **for** para escreva uma função que retorne o n -ésimo termo da função de Fibonacci¹⁶, $n \geq 1$.

Exercício 4.4.3. Implemente uma função para computar o produto escalar de dois vetores de n elementos. Use a instrução de repetição **for** e assuma que os vetores estão alocados como um arranjo **double**.

Exercício 4.4.4. Implemente uma função para computar a multiplicação de uma matriz A $n \times n$ por um vetor coluna x de n elementos. Use a instrução **for** e assuma que o vetor e a matriz estejam alocadas como arranjos **double**.

¹⁶Leonardo Fibonacci, 1170 - 1250, matemático italiano. Fonte: [Wikipédia](#).

Exercício 4.4.5. Implemente uma função para computar a multiplicação de uma matriz A $n \times m$ por uma matriz B de $m \times n$. Use a instrução `for` e assumo que as matrizes estão alocadas como arranjos `double`.

Referências