Matrizes em Fortran 90

Flavio Aristone
Juvenal Muniz

1 de maio de 2010

Resumo

Discutiremos neste documento a implementação de operações básicas sobre matrizes na linguagem de programação Fortran 90: adição, multiplicação, transposta e inversa. Calcularemos o determinante de uma matriz pela sua decomposição LU.

Sumário

1 Introdução

Uma matriz é um arranjo retangular de números. Por exemplo,

$$\mathbf{A} = \begin{pmatrix} 8 & 7 & 2 \\ 3 & 5 & 10 \\ 4 & 11 & 3 \\ 1 & 15 & 7 \end{pmatrix}$$

As operações básicas que podem ser aplicadas a matrizes são: *adição*, *multiplicação*, *transposição* e *inversão*. Nossa preocupação maior neste documento é a implementação destas operações na linguagem de programação Fortran 90. Para uma discussão teórica sobre matrizes, sugerimos o livro [3].

2 Teoria

Antes de implementarmos as operações matriciais, vamos fornecer um breve resumo de alguns conceitos da linguagem Fortran 90 que iremos utilizar mais adiante neste documento.

2.1 Arrays, vetores e matrizes

Para implementar matrizes na linguagem Fortran 90 iremos utilizar uma **array**[1]. Uma array é uma estrutura de dados que consiste de uma **coleção** de valores (variáveis ou constantes) de um mesmo tipo que são referidos por um mesmo nome. Um valor individual dentro da array é chamado um **elemento da array** e é identificado pelo nome da array juntamente com um **índice** indicando a localização do elemento dentro da array. Por exemplo, a seguinte coleção de 5 números inteiros

$$\{5, 7, 9, 1, 8\}$$

poderá ser representada em Fortran 90 por

```
integer, dimension(5) :: numeros = (/ 5, 7, 9, 1, 8 /)
```

O trecho de código acima declara uma array chamada "numeros" contendo 5 números inteiros. No código, a array "numeros" é inicializada já na sua própria declaração pelo **construtor** de arrays (/ ... /). Outra forma de declarar e inicializar esta array seria

```
integer, dimension(5) :: numeros

numeros(1) = 5
numeros(2) = 7
numeros(3) = 9
numeros(4) = 1
numeros(5) = 8
```

O número entre os parênteses é o índice que localiza a posição dentro da array; o índice 1 localiza a primeira posição na array, o 2 a segunda e assim em diante. Portanto, o código acima atribui o valor 5 à posição 1 da array, o valor 7 à posição 2, etc. O sinal de igual "=" é a forma de atribuir valores a variáveis em Fortran 90.

As arrays em Fortran 90 podem ter mais de um índice, podendo ser organizadas em múltiplas dimensões. Essas arrays são convenientes para representar dados organizados, por exemplo, em forma matricial (linhas e colunas). Utilizaremos arrays bidimensionais (dois índices) para representar matrizes. Chamaremos de **matrizes** as arrays bidimensionais e de **vetores** as arrays unidimensionais. Por exemplo, se quiséssemos representar a matriz identidade de ordem 3

$$\mathbf{Id}_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

poderíamos utilizar o seguinte código em Fortran 90

```
integer :: i, j
integer, dimension(3,3) :: identidade

do i = 1, 3
  do j = 1, 3
    if (i /= j) then
      identidade(i,j) = 0
    else
      identidade(i,j) = 1
    end if
  end do
end do
```

[1]Por falta de uma melhor tradução da palavra, iremos utilizar o termo em inglês neste documento.

Neste trecho de código utilizamos as variáveis **i** e **j** para representar os valores das linhas e colunas, respectivamente. O laço externo percorre as linhas da matriz, equanto que o laço interno percorre as colunas. Para cada elemento da matriz é atribuído o valor 0, caso a posição deste elemento esteja fora da diagonal ($i \neq j$), ou o valor 1, caso o elemento esteja na diagonal ($i = j$).

2.1.1 Alocação dinâmica de arrays

Em todos os trechos de código acima, as declarações das arrays são do tipo denominado **alocação estática da memória**, isso porque o tamanho de cada array é escolhido no momento da declaração e não poderá mudar. Desta maneira, o tamanho da array deverá ser grande o suficiente para que todos os valores possam ser armazenados sem problemas.

A alocação estática de memória pode apresentar sérias limitações. Por exemplo, suponha que você declare uma array de 1000 posições. Essa array irá ocupar parte da memória do computador para armazenar 1000 números, mesmo que você só utilize 2 posições dela, ou seja, você estaria desperdiçando 998 posições de memória.

Uma solução para este problema é a chamada **alocação dinâmica da memória**, exemplificada no seguinte código

```
integer, allocatable, dimension(:,:) :: identidade
```

O código acima declara a matriz identidade com alocação dinâmica da memória. Observe que os sinais de dois pontos, :, entre os parênteses, são obrigatórios, assim como a palavra "allocatable". Observe também que a vírgula após o primeiro sinal de dois pontos foi necessária, pois estamos declarando uma array bidimensional (uma matriz). Com este tipo de declaração, nenhuma memória é reservada. Para utilizarmos a matriz identidade, temos que alocar (reservar) memória antes. Veja o seguinte código

```
integer, allocatable, dimension(:,:) :: identidade

allocate(identidade(3,3))
```

Para alocarmos memória, utilizamos a função **allocate** com o nome da matriz e o tamanho que quisermos[2]. Feito isto, podemos atribuir valores aos elementos da matriz normalmente. Após utilizarmos a matriz, poderemos liberar a memória para ela reservada, utilizando a função **deallocate**, como a seguir

```
deallocate(identidade)
```

2.2 Funções e subrotinas

Utilizamos certos procedimentos em Fortran 90, tais como **funções** e **subrotinas**, para organizar melhor nossos programas. Essa organização facilita a correção de erros, estruturação do programa em tarefas e subtarefas, etc. Iremos apenas mostrar como utilizar funções e subrotinas em Fortran 90. Para mais informações sobre este assunto, sugerimos o livro [1].

Basicamente, subrotinas são procedimentos que são executados por uma instrução chamada **CALL**. Subrotinas podem retornar múltiplos resultados através da sua lista de argumentos de chamada. Funções são procedimentos que são executados pelos próprios

[2]Isto é, caso haja memória suficiente para alocar a matriz.

nomes em expressões e só podem retornar um único resultado a ser usado na expressão. Os exemplos a seguir dão a forma geral de uma subrotina e de uma função, respectivamente

```
subroutine minha_subrotina ( lista de argumentos )
...
  (declarações de variáveis)
...
  (código a ser executado)
...
return
end subroutine


function minha_funcao ( lista de argumentos )
...
  (declarações de variáveis)
...
  (código a ser executado)
...
minha_funcao = expressão
return
end function
```

Tanto para subrotinas quanto para funções, é possível estabelecer uma lista de argumentos que serão utilizados pela subrotina ou função. Vale destacar que na função, é necessário que haja uma atribuição de valor ao nome da função, indicando o valor que será retornado pela mesma.

3 Implementação

O código a seguir é a implementação em Fortran 90 das operações básicas sobre matrizes e o cálculo do determinante de uma matriz pela sua decomposição em matrizes triangulares inferior e superior, conhecida como decomposição LU. O algoritmo utilizado é conhecido como **algoritmo de Doolittle**. Utilizamos um algoritmo simples para o cálculo da inversa de uma matriz. Esse algoritmo é baseado no processo de eliminação gaussiana e tem um caráter mais didático, pois seu desempenho é baixo.

Tentamos, exageradamente, comentar o código para facilitar a compreensão do material. Utilizamos várias técnicas e construções sintáticas para mostrar várias formas de implementação. O código é extenso devido ao seu caráter didático, mas você poderá modificá-lo à vontade, buscando melhores formas de expor os algoritmos ou, até mesmo, utilizar algoritmos mais eficientes e precisos!

É aconselhável que você digite (ao invés de copiar-e-colar) todo o código e tente executá-lo, pois assim, estará treinando suas habilidades para detectar e corrigir erros, caso ocorram. Vale ressaltar que este é um código com caráter didático e não deve ser utilizado em outros ambientes que não os de sala de aula.

Algumas instruções em Fortran 90 possuem opções até agora não utilizadas, como por exemplo, a opção **advance='no'** na instrução **write**. Não detalharemos o significado dessas opções, mas todas elas são facilmente encontradas em [1].

```
! A variável seed será compartilhada com a subrotina que cria os elementos das matrizes A e B
! aleatoriamente. Isto é necessário para que os valores das matrizes não sejam iguais.
!
! Módulos devem sempre vir definidos antes do seu uso no programa.
module global
implicit none
integer :: seed
end module


! Início do programa
program matrizes
use global
!
! Propósito:
! Criar procedimentos para somar, multiplicar, transpor, inverter e decompor matrizes e
! calcular o determinante de uma matriz.
!
! Data: 1.5.2010
! Programador: Juvenal Muniz
!
! Revisões
! 1.5.2010 : Código original
implicit none

! Declarações das variáveis e das subrotinas utilizadas
! Descrição das variáveis e/ou matrizes
! n Dimensão da matriz
! i Contador
! condicao Indica se houve erro ao abrir o arquivo
! opcao Armazena a opcao selecionada pelo usuário
! determinante Valor do determinante da matriz
! a, b, c Matrizes a, b e c
! arquivo Constante utilizada para acessar o arquivo
! resposta Armazenará a resposta à pergunta (S ou N)
! nome_arquivo Nome do arquivo contendo os elementos das matrizes

! Descrição das subrotinas
! matriz_aleatoria Gera uma matriz quadrada aleatória de ordem n
! matriz_adicao Efetua a adição de duas matrizes
! matriz_mul Efetua a multiplicação de duas matrizes
! matriz_transposta Obtém a transposta de uma matriz
! matriz_inversa Obtém a inversa de uma matriz
! matriz_lu Decompõe a matriz em matrizes triangulares inferior e superior
! matriz_imprime Exibe os elementos da matriz

integer :: n, i, j, condicao, opcao
real :: determinante
real, allocatable, dimension(:,:) :: a, b, c
integer, parameter :: arquivo = 17
character(1) :: resposta
character(20):: nome_arquivo

! Valor inicial para a função RAN()
seed = 32764561

! Leitura da dimensão das matrizes quadradas
write(*,'(A)',advance='no') 'Qual e a dimensao das matrizes? '
read(*,'(I)') n

! Verifica se a dimensão é maior ou igual a 2. Caso contrário, interrompe o programa.
if (n >= 2) then
    allocate(a(n,n), b(n,n), c(n,n))
else
  write(*,*) 'As matrizes devem possuir dimensao maior ou igual a 2!'
  stop
end if

! Pede confirmação para gerar as matrizes aleatoriamente
write(*,'(A)',advance='no') 'Deseja que as matrizes a e b sejam geradas? (S ou N) '
read(*,'(A1)') resposta

! Verifica a resposta dada pelo usuário
if ((resposta == 'S') .or. (resposta == 's')) then
  ! Usuário prefere que as matrizes sejam geradas aleatoriamente
  call matriz_aleatoria(a, n)
```

```
  call matriz_aleatoria(b, n)
else
  ! Usuário prefere fornecer as matrizes em um arquivo
  write(*,'(/,x,A)') 'ATENCAO! Termine o arquivo com uma linha em branco.'
  write(*,'(x,A)',advance='no') 'Nome do arquivo contendo as matrizes (max. 20 ↩
      caracteres): '
  read(*,'(A20)') nome_arquivo

  ! Abrindo o nome_arquivo para leitura.
  ! status='old' força que nome_arquivo já exista
  ! iostat=condicao armazena informações sobre erros ao abrir o arquivo
  open(unit=arquivo, file=nome_arquivo, status='old', action='read', iostat=condicao)

  ! O nome_arquivo abriu sem problemas?
  if (condicao == 0) then
    ! Arquivo aberto
    ! Agora vamos ler os valores das matrizes a e b
    read(arquivo,*) ((a(i,j), j = 1, n), i = 1, n)
    read(arquivo,*) ((b(i,j), j = 1, n), i = 1, n)
    write(*,'(/,x,A)') 'Matrizes lidas com sucesso!'
  else
    ! Erro ao abrir o arquivo. Terminar o programa.
    write(*,'(/,x,A)') '** Erro ao abrir o arquivo! Verifique o arquivo de entrada.'
    close(arquivo)
    stop
  end if

  ! Fechar o arquivo, pois já temos os valores armazenados nas variáveis
  close(arquivo)
end if

! Menu para seleção das operações matriciais implementadas
! Este laço se repetirá até que o usuário escolha a opção 0.
do
  write(*,'(3/,x,A)') '(1) Somar as matrizes A e B'
  write(*,*) '(2) Multiplicar as matrizes A e B'
  write(*,*) '(3) Obter a transposta de A'
  write(*,*) '(4) Obter a inversa de A'
  write(*,*) '(5) Calcular o determinante de A'
  write(*,*) '(6) Exibir as matrizes A e B'
  write(*,*) '(0) Sair'
  write(*,'(/,A)',advance='no') 'Digite a opcao desejada (0-6): '
  read(*,*) opcao

  if (opcao == 1) then
    ! C = A + B
    call matriz_soma(a, b, c, n)

    write(*,'(/,x,A)') 'A soma da matriz A com B e igual a '
    call matriz_imprime(c, n)
  else if (opcao == 2) then
    ! C = A * B
    call matriz_mul(a, b, c, n)

    write(*,'(/,x,A)') 'O produto da matriz A com B e igual a '
    call matriz_imprime(c, n)
  else if (opcao == 3) then
    ! Matriz transposta de A
    write(*,'(/,x,A)') 'A matriz transposta de A e igual a '
    write(*,'(/,x,A)') 'MATRIX A'
    call matriz_imprime(a, n)
    call matriz_transposta(a, n)

    write(*,'(/,x,A)') 'MATRIX A (transposta)'
    call matriz_imprime(a, n)
  else if (opcao == 4) then
    ! Matriz inversa de A
    write(*,'(/,x,A)') 'A matriz inversa de A e igual a '
    write(*,'(/,x,A)') 'MATRIX A'
    call matriz_imprime(a, n)

    call matriz_inversa (a, b, n)

    write(*,'(/,x,A)') 'MATRIX B (inversa)'
    call matriz_imprime(b, n)
    call matriz_inversa_verifica(a, b, n)
```

```
  else if (opcao == 5) then
    ! Decomposição de A em LU
    call matriz_lu(a, n)

    ! Cálculo do determinante para uma matriz LU
    determinante = 1.0
    do i = 1, n
      determinante = determinante * a(i,i)
    end do

    write(*,'(/,x,A)') 'MATRIX A'
    call matriz_imprime(a, n)
    write(*,'(/,x,A,F)') 'O determinante da matriz A vale: ', determinante
  else if (opcao == 6) then
    ! Exibe os elementos das matrizes A e B
    write(*,'(/,x,A)') 'MATRIX A'
    call matriz_imprime(a, n)

    write(*,'(/,x,A)') 'MATRIX B'
    call matriz_imprime(b, n)
  else if (opcao == 0) then
    ! Interrompe o programa
    write(*,'(/,x,A,/)') 'Programa terminado. Ate mais!'
    exit
  end if
end do

! Libera a memória ocupada pelas matrizes A, B, C
deallocate(a, b, c)
end program


! Esta subrotina exibe os valores dos elementos de uma matriz
!
! Parâmetros:
! a Matriz cujos elementos serão exibidos
! n Dimensão da matriz (apenas matrizes quadradas)
subroutine matriz_imprime(a, n)
implicit none
integer :: n, i, j
real :: a(n,n)

do i = 1, n
  do j = 1, n
    write (*,'(f,2x)',advance='no') a(i,j)
  end do
  print *
end do
end subroutine


! Esta subrotina gera uma matriz com valores aleatórios para seus elementos
!
! Parâmetros:
! a Matriz que armazenará os elementos gerados
! n Dimensão da matriz (apenas matrizes quadradas)
subroutine matriz_aleatoria(a, n)
use global
implicit none
integer :: n, i, j
real :: a(n,n)

do i = 1, n
  do j = 1, n
    ! Calcula valores aleatórios de -5 à +5
    ! A função ran() retorna valores entre 0.0 e 1.0
    a(i,j) = (-1.0)**(i+j) * 5.0 * ran(seed)
  end do
end do
end subroutine


! Esta subrotina faz a adição de duas matrizes quadradas (a + b)
!
! Parâmetros:
! a e b Matrizes de entrada
```

```
! c Matriz que armazenará a soma de a com b
! n Dimensão da matriz (apenas matrizes quadradas)
subroutine matriz_soma(a, b, c, n)
implicit none
integer :: n, i, j
real :: a(n,n), b(n,n), c(n,n)

do i = 1, n
  do j = 1, n
    c(i,j) = a(i,j) + b(i,j)
  end do
end do
end subroutine


! Esta subrotina faz a multiplicação de duas matrizes quadradas (a * b)
!
! Parâmetros:
! a e b Matrizes de entrada
! c Matriz que armazenará o produto de a por b
! n Dimensão da matriz (apenas matrizes quadradas)
subroutine matriz_mul(a, b, c, n)
implicit none
integer :: n, i, j, k
real :: a(n,n), b(n,n), c(n,n)

c = 0.0

do i = 1, n
  do j = 1, n
    do k = 1, n
      c(i,j) = c(i,j) + a(i,k) * b(k,j)
    end do
  end do
end do
end subroutine


! Esta subrotina cria a transposta de uma matriz
!
! Parâmetros:
! a Matriz de entrada
! n Dimensão da matriz (apenas matrizes quadradas)
!
! AVISO: A matriz a será modificada no processo
subroutine matriz_transposta(a, n)
implicit none
integer :: n, i, j
real :: a(n,n), temp

do i = 1, n
  do j = 1, n
    if (i < j) then
      temp = a(i,j)
      a(i,j) = a(j,i)
      a(j,i) = temp
    end if
  end do
end do
end subroutine


! Esta subrotina calcula a inversa de uma matriz por eliminação gaussiana
!
! Parâmetros:
! matriz Matriz de entrada
! inversa Matriz que armazenará a inversa
! n Dimensão da matriz (apenas matrizes quadradas)
!
! AVISO: A matriz a será modificada no processo
! Referência :
! http://math.uww.edu/~mcfarlat/inverse.htm
! http://www.tutor.ms.unimelb.edu.au/matrix/matrix_inverse.html
subroutine matriz_inversa(matriz, inversa, n)
implicit none
integer :: n
```

```
real, dimension(n,n) :: matriz
real, dimension(n,n) :: inversa

logical :: invertivel = .true.
integer :: i, j, k, l
real :: m
real, dimension(n,2*n) :: matriz_aumentada

! Matriz aumentada com uma matriz identidade
do i = 1, n
  do j = 1, 2 * n
    if (j <= n ) then
      matriz_aumentada(i,j) = matriz(i,j)
    else if ((i+n) == j) then
      matriz_aumentada(i,j) = 1
    else
      matriz_aumentada(i,j) = 0
    endif
  end do
end do

! Reduzir a matriz aumentada a uma matriz triangular superior pela eliminação gaussiana
do k = 1, n - 1
  ! Verifica se algum elemento da diagonal é zero
  if (abs(matriz_aumentada(k,k)) <= 1.0E-6) then
    invertivel = .false.
    do i = k + 1, n
      ! Verifica se os elementos são maiores que zero
      if (abs(matriz_aumentada(i,k)) > 1.0E-6) then
        do j = 1, 2 * n
          matriz_aumentada(k,j) = matriz_aumentada(k,j) + matriz_aumentada(i,j)
        end do
        invertivel = .true.
        exit
      endif

      ! Se algum elemento da diagonal for zero, não podemos calcular a inversa
      if (invertivel == .false.) then
        write(*,'(/,x,A,/)') '** A matriz nao e inverstivel! **'
        inversa = 0
        stop
      endif
    end do
  endif

  ! Eliminação gaussiana
  do j = k + 1, n
    m = matriz_aumentada(j,k) / matriz_aumentada(k,k)

    do i = k, 2 * n
      matriz_aumentada(j,i) = matriz_aumentada(j,i) - m * matriz_aumentada(k,i)
    end do
  end do
end do

! Teste para invertibilidade
do i = 1, n
  ! Elementos da diagonal não podem ser zero
  if (abs(matriz_aumentada(i,i)) <= 1.0E-6) then
    write(*,'(/,x,A,/)') '** A matriz nao e inverstivel! **'
    inversa = 0
    stop
  endif
end do

! Elementos da diagonal iguais a 1
do i = 1 , n
  m = matriz_aumentada(i,i)
  do j = i , 2 * n
      matriz_aumentada(i,j) = matriz_aumentada(i,j) / m
  end do
end do

! Reduzir o lado esquerdo da matriz aumentada a matriz identidade
do k = n - 1, 1, -1
  do i = 1, k
```

```
  m = matriz_aumentada(i,k+1)
    do j = k, 2 * n
      matriz_aumentada(i,j) = matriz_aumentada(i,j) - matriz_aumentada(k+1,j) * m
    end do
  end do
end do

! Armazene o resultado
do i =1, n
  do j = 1, n
    inversa(i,j) = matriz_aumentada(i, j + n)
  end do
end do

end subroutine


! Esta subrotina verifica se a matriz inversa é válida (a * b = identidade)
!
! Parâmetros:
! a Matriz original
! b Matriz inversa de a
! n Dimensão da matriz (apenas matrizes quadradas)
subroutine matriz_inversa_verifica(a, b, n)
implicit none
integer :: n, i, j
real, dimension(n,n) :: a, b, identidade
logical :: ok

ok = .true.

call matriz_mul(a, b, identidade, n)

do i = 1, n
  do j = 1, n
    ! Elementos fora da diagonal principal não podem ser diferentes de zero
    if ((i /= j) .and. (abs(identidade(i,j)) >= 1.0E-6)) then
      ok = .false.
    ! Elementos na diagonal principal não podem ser diferentes de um
    else if ((i == j) .and. (abs(identidade(i,i) - 1.0) >= 1.0E-6)) then
      ok = .false.
    end if
  end do
end do

if (ok == .false.) then
  write(*,'(/,x,A,/)') 'A verficacao da matriz inversa falhou'
else
  write(*,'(/,x,A,/)') 'A verficacao da matriz inversa foi concluida com sucesso!'
end if

end subroutine


! Esta subrotina decompõe uma matriz em matrizes triangulares inferior e superior (LU)
! pelo algoritmo de Doolittle (diagonal principal da matriz L igual a 1). Para uma
! descrição teórica sobre o algoritmo de Doolittle, veja [2].
!
! Parâmetros:
! a Matriz que armazenará as matrizes LU
! n Dimensão da matriz (apenas matrizes quadradas)
!
! AVISO: A matriz a será modificada no processo
subroutine matriz_lu(a, n)
implicit none
integer :: n, i, j, k
real :: a(n,n), s, ss

! Os elementos da diagonal não podem ser zero
if (abs(a(1,1)) <= 1.0E-6) then
  write(*,'(/,x,A,/)') '** Impossivel fatorar a matriz **'
  stop
end if

do i = 2, n
  a(i,1) = a(i,1) / a(1,1)
```

```
end do

do i = 2, n - 1

  s = 0.0
  do k = 1, i - 1
    s = s - a(i,k) * a(k,i)
  end do

  a(i,i) = ( a(i,i) + s )

  ! Os elementos da diagonal não podem ser zero
  if (abs(a(i,i)) <= 1.0E-6) then
    write(*,'(/,x,A,/)') 'impossivel fatorar a matriz'
    stop
  end if

  do j = i + 1, n
    ss = 0.0
    s = 0.0

    do k = 1, i - 1
      ss = ss - a(i,k) * a(k,j)
      s = s - a(j,k) * a(k,i)
    end do

    a(i,j) = a(i,j) + ss
    a(j,i) = (a(j,i) + s) / a(i,i)
  end do
end do

s = 0.0

do k = 1, n - 1
  s = s - a(n,k) * a(k,n)
end do

a(n,n) = a(n,n) + s
end subroutine
```

Referências

[1] Stephen J. Chapman, *Fortran 95/2003 for scientists and engineers*, McGrall-Hill Companies, New York, NY, 2007.

[2] R.L. Burden e J.D. Faires, *Numerical analysis*, Brooks Cole, California, CA, 2000.

[3] J.L. Boldrini et al., *Álgebra linear*, Editora Harbra, Ltda., São Paulo, SP, 1980.