Relatório Trabalhos Computacionais – P1B – TASK 02
Algebra Linear Computacional

Aluno: Abraão Carvalho Gomes **DRE:** 121.066.101

Professor: Luís Volnei S. Sagrilo

1. Introdução

1.1 Descrição do Projeto da Tarefa 02.

O ‘Task 02’ requer a implementação de um programa computacional para o cálculo dos autovalores e autovetores de uma matriz A utilizando dois métodos: o Método da Potência e o Método de Jacobi. Além disso, o programa permite o cálculo do determinante da matriz, quando requisitado pelo usuário. Nesse sentido, ambas informações sobre o método utilizado e a necessidade de impressão de determinante são perguntadas ao usuário, armazenando-as em variáveis de controle ‘ICOD’ e ‘detCOD’.

O programa foi desenvolvido na linguagem de programação de alto desempenho Fortran e permite a leitura da matriz A a partir de um arquivo ASCII, como um arquivo de texto (.txt) ou um arquivo de dados (.dat). A leitura dos dados da matriz é feita a partir da inserção do caminho no sistema pelo usuário, facilitando a utilização do programa com diferentes conjuntos de dados.

1.2 Fluxo de execução do programa implementado

1. É perguntado ao usuário se ele deseja utilizar o método da Potência(ICOD = 1) ou o método de Jacobi(ICOD = 2).
2. É perguntado ao usuário se ele deseja calcular o determinante(detCOD = 1 para sim e detCOD = 0 para não).
3. O método selecionado é executado. Caso seja o da potência, o autovalor e seu autovetor correspondente é impresso, mas se for o de Jacobi uma lista com os autovalores é impressa, seguida de uma matriz com todos os autovetores em suas respectivas posições.

4. Em seguida, caso o usuário deseje o determinante de A, há duas opções: Se tiver sido realizado o método da potência, é emitido um warning avisando que o método não suporta cálculo de determinantes. Caso o método de Jacobi tenha sido executado, o cálculo da determinante é realizado normalmente, multiplicando todos os autovalores de A e imprimindo o resultado no terminal.

2. Listagem de Código

2.1 Hierarquia de arquivos e modularização

A implementação da ‘task 02’ foi mais simples e, portanto, precisou apenas de um programa principal e duas subrotinas, uma para cada método de resolução de autovalores e autovetores.

Nesse sentido, o arquivo principal foi nomeado como ‘autoV_solver.f90’ e contém um módulo utilizado pelo programa principal. Tal módulo contém as subrotinas do arquivo ‘routines/main_routines_autoV.f90’.

```
module mod1

    CONTAINS
        include 'routines/main_routines_autoV.f90'

end module mod1

program eigen_calculator
    use mod1
```

2.2 Programa Principal

O programa central é responsável por:

1. Perguntar ao usuário o caminho até o arquivo que contém matriz A, verificar número de linhas/colunas em A, alocar espaço para a matriz A e ler os dados em A.
2. Obter o método de resolução, armazenando em ‘ICOD’
3. Obter se determinante é desejável, armazenamento em ‘detCOD’
4. Executar subrotinas importadas conforme necessário.
5. Imprimir na tela os resultados desejados.

Em seguida, segue imagens com os códigos das instruções descritas:

(1)

```
program eigen_calculator
   use mod1
   implicit none
   integer ::n, i, ICOD,detCOD,io
   character*50 :: filename
   real*8 :: detA
   real*8, allocatable :: A(:,:), X(:,:), E(:) ! X : matriz com autovetores ; E : vetor com autovalores

   write(*,*) 'Insira o caminho até o arquivo contendo a matrix A: '
   read(*,'(A)') filename
   ! Descobrindo número de linhas de A (n).
   n = 0
   OPEN (11,status = 'old', file = filename)
   DO
   READ(11,*,iostat=io)
   IF (io/=0) EXIT
   n = n + 1
   END DO
   rewind(11)
   ! Alocando espaço para A
  allocate(A(n,n))
  ! Lendo matriz A
  do i = 1,n
   read(11,*)A(i,:)
  enddo
  CLOSE (11)
```

(2)

```
write(*,'(A,/,A,/,A)') 'Insira ICOD:','ICOD = 1 : Metodo da potencia ',&
 'ICOD = 2 : Metodo de Jacobi'
 read(*,*) ICOD
```

(3)

```
 write(*,'(A,/,A)') 'Deseja calculo de determinante?', 'Responda com 0 ou 1. Sim = 1 ; Não = 0 '
 read(*,*) detCOD
```

Segue na próxima página

(4~5)

```
if(ICOD .eq. 1) then
   ! Alocar espaco para 1 autovetor e 1 autovalor nas matrizes criadas X e E.
   allocate(X(n,1) , E(1))
   ! Chamar subrotina com metodo da potencia
   call power_method(A,E(1),X(:,1))
   write(*,'(/,A, f8.4)') 'Autovalor encontrado pelo metodo da potencia: ', E(1)
   write(*,*) 'Autovetor encontrado pelo metodo da potencia: ','[' ,X(:,1), ']'

else if(ICOD .eq. 2) then
   ! Alocar espaco para autovetores e autovalores esperados do metodo de jacobi.
   allocate(X(n,n) , E(n))
   ! Chamar subrotina com metodo de Jacobi.
   call jacobi_method(A,E,X)
   write(*,'(A,/)') 'Segue autovalores de A e, em seguida, matriz com autovetores associados em ordem.'
   write(*,*),'[ ',E(:) , '  ]'
   write(*,'(/)')
   do i = 1,n
      do j = 1,n
     write(*,'(f8.4,A)', advance = 'no') X(i,j) ,'     '
      enddo
      write(*,'(/)')
   enddo

endif
if(detCOD .eq. 1) then
   if(ICOD .eq. 1) then
      write(*,*) "WARNING: The power method doesn't allow determinant calculation! Try Jacobi Method instead"
   else if(ICOD .eq. 2) then
      detA = 1
      do i =1,n
         detA = detA * E(i)
      enddo
      write(*,*) 'O determinante de A é ', detA
   endif
endif
end program
```

2.3 Método da Potência

O método é executado com parâmetros de número máximo de iterações, tolerância e inicialização de autovetores fixos, esses que podem ser ajustados em código.

Caso o maior valor encontrado no autoVetor atual seja considerado 0 (menor que 1.0e-12), é emitido um ‘Warning’ indicando que a matriz é singular e não é possível prosseguir com o cálculo de autovalor.

```
subroutine power_method(A, eigenvalue, eigenvector)
  implicit none
  integer :: i, n, iter_max, iter_count
  real*8 :: tol, lambda_old, lambda_new, norm_factor

  real*8, intent(in):: A(:,:)
  real*8, intent(out) :: eigenvalue
  real*8, intent(out) :: eigenvector(:)

  n = size(A, 1)
  iter_max = 10000 ! Número máximo de iterações
  tol = 1.0E-5 ! Tolerância para convergência

  eigenvector = 1.0 / sqrt(real(n))! Inicialização do autovetor

  iter_count = 0
  do while (iter_count < iter_max)
    lambda_old = eigenvalue
    eigenvector = matmul(A, eigenvector) ! Multiplica a matriz pelo autovetor
    eigenvalue = maxval(abs(eigenvector)) ! Fator de normalização
    if ( eigenvalue .lt. 1.0e-12) then
      write(*,*) "WARNING: Singular Matrix, can't find eigenvalue! "
      exit
    endif
    eigenvector = eigenvector / eigenvalue ! Normalização do autovetor

    if (abs(eigenvalue - lambda_old) < tol) exit ! Critério de convergência
    iter_count = iter_count + 1
  end do
  print*,'Convergência alcançada na iteração', iter_count

end subroutine power_method
```

Nota-se que a função abs() do fortran foi utilizada, tal que, nesse caso, recebe um vetor como entrada, retornando a norma euclidiana do vetor.

2.3 Método de Jacobi

O código para o método de jacobi segue os passos originais dados na teoria, e difere apenas em um deles. Seja os passos:

1. Inicialize autovetores e parâmetros.

Enquanto não alcançar uma das condições de parada(número máximo de iterações alcançado ou valor máximo fora da diagonal menor que a tolerância), faça:

2. Encontrar maior valor absoluto fora da diagonal A(p,q), armazenando p e q. Se for menor que a tolerância, pare.
3. Definir ângulo ‘theta’ e seus respectivos valores para seno e cosseno.

   É possível mas não necessário montar uma matriz P, que inicializa como identidade, mas é modificada assim:
   P(p,p) = cos(theta) ; P(p,q) = -sen(theta) ; P(q,q) = cos(theta) ; P(q,p) = sen(theta)

4. Realizar operações necessárias para efetivar as seguintes multiplicações matriciais:

   A(k+1) = P^t . A(k) . P

   X(k+1) = X(k) . P

   Em minha implementação, fiz essas multiplicações de forma mais direta, sem montar P.

Segue cada uma das etapas na implementação feita:

(1)

```
subroutine jacobi_method(A, eigenvalues, eigenvectors)
  implicit none
  integer :: i, j, p, q, n, iter_max, iter_count, maxOffDiag_i, maxOffDiag_j
  real*8,intent(inout) :: A(:,:)
  real*8, intent(out) :: eigenvalues(:), eigenvectors(:,:)
  real*8 :: tol, c, s, t, sum_offdiag, max_offdiag, theta

  n = size(A, 1)
  iter_max = 10000 ! Nú?mero máximo de iterações
  tol = 1.0E-5 ! Tolerância para convergência
  eigenvectors = 0.0
  do i = 1, n
    eigenvectors(i,i) = 1.0
  end do
```

(2)

```
iter_count = 0
do while (iter_count < iter_max) ! limitando numero maximo de iteracoes ( condicao de parada 1 )
  ! Primeiro, deseja-se achar o maior elemento da matriz A e seu indice.
  max_offdiag = 0.0
  do i = 1, n-1
    do j = i+1, n
      if (abs(A(i,j)) > max_offdiag) then
        max_offdiag = abs(A(i,j))
        p = i
        q = j
      end if
    end do
  end do

 ! Se o maior elemento da matrix for menor que a tolerancia, considera-se A diagonalizada.
  if (max_offdiag < tol) exit ! Crité?rio de convergê?ncia
```

(3)

```
! Metodo para diagonalizar A.

 ! c é o cosseno de 'fi' e s é o seno de 'fi'. fi é arcotg(2.0 * A(p,q) / (A(p,p) - A(q,q))
 if (abs(A(p,p) - A(q,q)) < tol) then
   c = sqrt(2.0) / 2.0
   s = sqrt(2.0) / 2.0
 else
   theta = 0.5 * atan(2.0 * A(p,q) / (A(p,p) - A(q,q)))
   c = cos(theta)
   s = sin(theta)
 end if
```

(4)

```
   ! Sendo P a matriz que incializa como identidade e tem:
   !  P(p,p) = cos(theta) ; P(p,q) = -sen(theta) ; P(q,q) = cos(theta) ; P(q,p) = sen(theta)

   ! Equivalente a multiplicar P transposta por A = A'
   do i = 1, n
     t = A(i,p)
     A(i,p) = c * t + s * A(i,q)
     A(i,q) = -s * t + c * A(i,q)
   end do

   ! Equivalente a multiplicar A' por P = A'' = P^t . A . P
   do j = 1, n
     t = A(p,j)
     A(p,j) = c * t + s * A(q,j)
     A(q,j) = -s * t + c * A(q,j)
   end do

   ! Equivalente a multiplicar X por P
   do i = 1, n
     t = eigenvectors(i,p)
     eigenvectors(i,p) = c * t + s * eigenvectors(i,q)
     eigenvectors(i,q) = -s * t + c * eigenvectors(i,q)
   end do

   iter_count = iter_count + 1
 end do
 print*,'Convergência alcançada na iteração', iter_count

 ! Obtenho autovalores da diagonal de A, que agora foi diagonalizada.
 do i = 1, n
  eigenvalues(i) = A(i,i)
 enddo
```

3. Testes - Exemplo resolvido com código

Foi utilizada a mesma matriz fornecida para a ‘task 01’, sendo assim:

A =

```
 30     0   -15    15     0     0     0     0     0     0
  0    40   -15    10     0     0     0     0     0     0
-15   -15    30     0   -15    15     0     0     0     0
 15    10     0    40   -15    10     0     0     0     0
  0     0   -15   -15    30     0   -15    15     0     0
  0     0    15    10     0    40   -15    10     0     0
  0     0     0     0   -15   -15    30     0   -15    15
  0     0     0     0    15    10     0    40   -15    10
  0     0     0     0     0     0   -15   -15    30     0
  0     0     0     0     0     0    15    10     0    40
```

3.1) Utilizando método da potência

```
C:\Users\abraa\Documents\projetos\AlgLinComp_routines>autoV_solver
 Insira o caminho at├® o arquivo contendo a matrix A:
matrizes/Matriz_A.dat
Insira ICOD:
ICOD = 1 : Metodo da potencia
ICOD = 2 : Metodo de Jacobi
1
Deseja calculo de determinante?
Responda com 0 ou 1. Sim = 1 ; N├úo = 0
1
 Converg├¬ncia alcan├ºada na itera├º├úo         667

Autovalor encontrado pelo metodo da potencia:  64.3280
 Autovetor encontrado pelo metodo da potencia: [  0.400033550372241
  0.551260462907735      -0.851212224902395       6.424965538159752E-002
 -3.261589624199749E-003  -1.00000000000000       0.851738356684504
  5.737692358536305E-002 -0.397235822783994       0.548728532728934      ]
 WARNING: The power method doesn't allow determinant calculation! Try Jacobi Met
 hod instead
```

3.2) Utilizando método de Jacobi

```
C:\Users\abraa\Documents\projetos\AlgLinComp_routines>autoV_solver
 Insira o caminho at├® o arquivo contendo a matrix A:
matrizes/Matriz_A.dat
Insira ICOD:
ICOD = 1 : Metodo da potencia
ICOD = 2 : Metodo de Jacobi
2
Deseja calculo de determinante?
Responda com 0 ou 1. Sim = 1 ; N├úo = 0
1
 Converg├¬ncia alcan├ºada na itera├º├úo         158
Segue autovalores de A e, em seguida, matriz com autovetores associados em ordem.

 [   0.4445554416001 46         35.7931553356035        61.6786625035411
   64.0272585193892        8.05636717300952        64.3279923583547
   2.75921590197876        61.2842769351248        15.4272048040973
   36.2013089735884        ]


  0.1794      -0.4475      -0.2375       0.2171      -0.4050      -0.2168      -0.3369       0.2415      -0.3172      -0.4249

  0.1505       0.5270      -0.3422       0.1965      -0.2095      -0.2991      -0.2326       0.2343      -0.0946       0.5409

  0.4837       0.0977       0.4808       0.0408      -0.1537       0.4631      -0.5089       0.0097       0.1514       0.0596

  0.1303      -0.0751      -0.0208       0.5333       0.4388      -0.0331       0.1029       0.5134       0.4596      -0.1161

  0.6232      -0.1172      -0.4356      -0.5062       0.3896      -0.0000       0.0000       0.0000      -0.0000       0.0000

 -0.0000       0.0000       0.0000      -0.0000      -0.0000       0.5439       0.3547       0.4962      -0.5590       0.1406

  0.4837       0.0977       0.4808       0.0408      -0.1537      -0.4631       0.5089      -0.0097      -0.1514      -0.0596

 -0.1303       0.0751       0.0208      -0.5333      -0.4388      -0.0331       0.1029       0.5134       0.4596      -0.1161

  0.1794      -0.4475      -0.2375       0.2171      -0.4050       0.2168       0.3369      -0.2415       0.3172       0.4249

 -0.1505      -0.5270       0.3422      -0.1965       0.2095      -0.2991      -0.2326       0.2343      -0.0946       0.5409

 O determinante de A ├®    3075468539462.40
```