Relatório Trabalhos Computacionais – P1B – TASK 01 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 01.

O ‘Task 01’ é uma tarefa de implementação de um programa capaz de realizar a solução de um sistema de equações lineares A . X = B, por meio de diversos métodos estudados na teoria.

A tarefa é dividida em 2 partes: métodos de decomposição e métodos iterativos, sendo que todos os métodos são aceitos por um mesmo programa em função de uma entrada ICOD, que determina o método de resolução na seguinte ordem:

1. Decomposição LU
2. Decomposição de Cholesky
3. Procedimento iterativo Jacobi
4. Procedimento iterativo Gauss-Seidel

Além disso, algumas observações foram feitas para a tarefa:

→ A matriz A e o(s) vetor(es) B deverão ser lidos a partir de arquivos de dados no formato (ASCII) (1)

→ Sugestão: não use rotinas prontas disponíveis na literatura/internet. Desenvolva as suas próprias rotinas para que esta atividade de programação se torne um aprendizado em métodos numéricos;

→ Tente desenvolver os códigos visando um armazenamento mínimo de dados na memória do computador (por exemplo, não deve ser criada uma nova matriz similar à matriz A para a solução do sistema e equações);

Dentre outras...

Assim, o programa foi implementado na linguagem de alto desempenho Fortran, utilizando apenas subrotinas próprias implementadas, com exceção a poucas funções nativas básicas do fortran, como matmul, maxval e sqrt.

1.2 Fluxo de execução do programa implementado

1. O usuário insere o caminho do arquivo referente à matriz A

2. O usuário escolhe o tipo de procedimento desejado.

3. Em seguida, caso se trate de uma decomposição, ela é feita ‘inplace’, substituindo a matriz A uma matriz que contenha LU ou LL^t, economizando memória.

4. Por fim, o usuário deve inserir o caminho do vetor B, e pode fazer isso iterativamente, enquanto não digitar ‘-1’, o que encerra o programa. A cada iteração, para realizar a solução com um novo vetor B, a matriz anteriormente calculada por decomposição é reutilizada, caso seja utilizado o método LU ou Cholesky.

2. Listagem do Código

2.1 Hierarquia de arquivos e modularização

A implementação final da ‘task’ 01 foi dividida em alguns arquivos distintos para melhor manutenção e visualização de código.

O arquivo principal foi batizado de ‘axb_solver.f90’ e se encontra na base do diretório do projeto, enquanto outros códigos se localizam dentro da pasta ‘routines’. Esse código central inicialmente cria um módulo com os outros arquivos, como é possível ver:

```
module mod1

    CONTAINS
        include 'routines/essentials.f90'
        include 'routines/LU_decomp.f90'
        include 'routines/Cholesky_decomp.f90'
        include 'routines/main_routines_axb.f90'

end module mod1

program AXB_Solver
    use mod1
```

O código central é responsável por ler os dados e controlar o fluxo de execução do programa, enquanto os outros possuem subrotinas auxiliares ou subrotinas responsáveis pelos quatro algoritmos a serem executados.

2.2 Código principal - ‘axb_solver.f90’.

A princípio, é preciso para o programa:

1. Definir as variáveis necessárias para a execução do código.
2. Definir número de colunas em B(no caso dessa tarefa é fixo = 1).
3. Perguntar ao usuário o caminho até o arquivo que contém a matriz A, descobrir tamanho de A, alocar espaço para a matriz A e ler a matriz A.

Todos os três itens descritos estão apresentados abaixo:

```
program AXB_Solver
   use mod1
   implicit none
   integer ::n, nb, i, ICOD, io, cb
   character*50 :: filename
   real*8, allocatable :: auxVector(:)
   real*8, allocatable :: A(:,:), X(:,:), B(:,:), Y(:,:)

   ! Definir número de colunas em cada Vetor B.
   cb = 1

   ! Ler caminho do arquivo que contém a matrix A
   write(*,*) 'Insira o caminho até o arquivo contendo a matrix A: '
   read(*,'(A)') filename

   ! Descobrir número de linhas / colunas de A. ( Supõe-se que seja quadrada).
   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, B e solução X.
  allocate(A(n,n))

  ! Lendo matriz A
  do i = 1,n
   read(11,*)A(i,:)
  enddo
  CLOSE (11)
```

Após isso, é requisitado ao usuário que tipo de método de obtenção de solução ele deseja usar, dentre os quatro disponíveis, e é lido um valor para a variável ICOD. Além disso, tendo a matriz A e o método escolhido, é possível realizar a decomposição de A. Segue respectivo fragmento do código:

```
write(*,'(A,/,/,A,/,A,/,A,/,A)') 'Insira ICOD:','ICOD = 1 : Solver com decomposição LU ',&
 'ICOD = 2 : Solver com decomposição de Cholesky',&
 'ICOD = 3 : Solver com método iterativo Jacobi',&
 'ICOD = 4 : Solver com método iterativo Gauss-Seidel'
 read(*,*) ICOD

 if(ICOD .eq. 1) then
    call LU_decomposition_inplace(A)
 else if(ICOD .eq. 2) then
    call cholenskyDecomp(A)
 endif
```

Por fim, é criado um loop que pergunta o caminho até o arquivo contendo o vetor B, lê este vetor, realiza a solução do sistema de equações lineares e imprime na tela o resultado para X, em A . X = B.

Observa-se que se o usuário digitar -1 para o caminho de ‘B’, ele encerra a execução do programa.

```
do while (filename .ne. '-1')
   write(*,*) 'Insira o caminho até o arquivo contendo B: (Caso deseja fechar o programa, digite -1)'
   read(*,'(A)') filename
   if (filename .eq. '-1') then
      stop
   endif
   ! Descobrindo número de linhas de B (nb).
   OPEN (11, status = 'old', file = filename)
   DO
      READ(11,*,iostat=io)
      IF (io/=0) GO TO 90
      nb = nb + 1
   enddo
   90 rewind(11)
   if(nb .eq. n) then ! Verifico se número de linhas de B é equivalente ao de A.
      allocate(B(nb,cb), Y(n,cb), X(n,cb)) ! Aloco espaço para matrizes B, Y( intermediária ) e X ( Solução )
      do i = 1,nb
         read(11,*) B(i,:) ! Leitura de B
      enddo
      ! Dependendo do método escolhido, realiza substituição para frente e para trás ou realiza método iterativo.
      if ((ICOD .eq. 1 ) ) then
         call forward_substitution_inplace(A, B, Y) ! L . Y = B ;
         call back_substitution(A, Y, X) ! U . X = Y

      else if ((ICOD .eq. 2 ) ) then
         call forward_substitution(A, B, Y) ! L . Y = B ;
         call back_substitution(A, Y, X) ! U . X = Y

      else if ((ICOD .eq. 3 ) ) then
         call jacobi_method(A, X, B)

      else if ((ICOD .eq. 4 ) ) then
         call Gauss_Seidel_method(A, X, B)
      endif
      ! Imprimir X
      write(*,*) 'Matrix solução X: '
      do i = 1,n
         write(*,*) X(i,:)
      enddo
```

2.3 Decomposição LU.

A decomposição LU ocorre de maneira a substituir a matriz A pelas matrizes triangulares inferior(L) e superior(U). Se for encontrado um pivô equivalente a 0, é emitido um 'Warning' avisando que a matriz é singular e não é possível realizar a decomposição.

```fortran
subroutine LU_decomposition_inplace(A)
    implicit none
    integer :: i, j, k, n, info
    real*8 :: s
    real*8, intent(inout) :: A(:,:)

    n = size(A, 1)

    do j = 1, n ! Iteração em linhas, descendo na posição do pivô.
        do i = 1, j ! (Atualizar Matrix U dentro de A)
            s = 0
            do k = 1, i - 1
                s = s + A(i,k) * A(k,j)
            enddo
            A(i,j) = A(i,j) - s
        end do
        !Se elemento da diagonal principal for 0, então a matriz é singular.
        if (A(j,j) < 1e-12) then
            info = j
            write(*,*) "WARNING: Singular Matrix, can't continue !"
        endif
        ! (Atualizar Matrix L dentro de A)
        do i = j + 1, n
            s = 0
            do k = 1, j - 1
                s = s + A(i,k) * A(k,j)
            enddo
            A(i,j) = (A(i,j) - s) / A(j,j)
        end do
    end do
    info = 0
end subroutine LU_decomposition_inplace
```

2.4 Decomposição de Cholesky.

A decomposição de Cholesky ocorre de maneira a substituir a matriz A pela matriz triangular inferior(L) e a mesma transposta(L^t), tal que L . L^t = A. Durante a execução, caso seja encontrado pivô igual a 0, a execução também é interrompida e um 'Warning' é emitido.

```fortran
subroutine cholenskyDecomp(A)
    implicit none
    integer :: i, j, k, n, info
    real*8 :: sum
    real*8, intent(inout) :: A(:,:)

    n = size(A, 1)
    info = 0

    do j = 1, n
      sum = 0.0
      do k = 1, j - 1
        sum = sum + A(j,k)**2
      end do

      if ((abs(A(j,j) - sum)  < 1e-12 ) ) then
        info = -1
        stop 'WARNING1: A não é uma matriz positiva definida'
      endif

      A(j,j) = sqrt(A(j,j) - sum)

      do i = j+1, n
        sum = 0.0
        do k = 1, j - 1
          sum = sum + A(i,k)*A(j,k)
        end do

        A(i,j) = (A(i,j) - sum) / A(j,j)
        A(j,i) = A(i,j) ! L transposta já incluida junto na matrix A
      end do
    end do

    if (info == 0) then
      write(*,*) 'Fatorização de Cholesky bem sucedida.'
    endif
  end subroutine cholenskyDecomp
```

2.5 Método de Jacobi.

O método de Jacobi inicia com uma verificação se a matriz é diagonalmente dominante. Se ela não tiver essa característica, o código não é interrompido, mas um ‘Warning’ é emitido alertando para a possibilidade de não convergência.
Além disso, o vetor solução é inicializado com 1, a tolerância é configurada inicialmente para 1e-05 e o número de iterações máximas é 1000. Esses e outros parâmetros devem ser ajustados dentro do código.

```fortran
subroutine jacobi_method(A, X, B )
   implicit none

   integer :: i,j, iter_max, iter, n,cA, p, flagDiagonalDominant
   real*8 :: tol,sumTmp,sumTmp2
   real*8,dimension(:,:), intent(in):: A, B
   real*8,dimension(:,:), intent(out) :: X
   real*8, dimension(size(X,1),size(X,2)) ::  Xold

   ! Valores iniciais
   n = size(A,1)
   cA = size(A,2)
   p = size(X,2)
   X = 1.0 ! Vetor solução inicial.
   iter_max = 1000
   tol = 1.0e-5
   flagDiagonalDominant = 1
   write(*,*)'--------------'
   ! Verifica se a matriz A é diagonalmente dominante e emite um warning caso não seja.
   do i = 1, n
       if (abs(A(i,i)) <= sum(abs(A(i,:))) - abs(A(i,i))) then
         flagDiagonalDominant = 0
       endif
   end do
   if ( flagDiagonalDominant .eq. 0) then
      write(*,*) "WARNING: Matrix isn't diagonally dominant, convergency not guaranteed. "
   endif

   ! Método iterativo de Jacobi
   do iter = 1, iter_max
       ! Copia X para Xold
       Xold = X(:,:)
       ! Atualiza o valor de cada elemento de X
       do i = 1, n
            ! Computar somatório fora da diagonal
            sumTmp = 0.0
            do j= 1,cA
               if (i .ne. j)then
                  sumTmp = sumTmp + A(i,j) * Xold(j,1)
               endif
            enddo
            ! Verificar se A é singular.
            if(A(i,i) <= 1.0e-12) then
               write(*,*) 'WARNING: Singular Matrix'
               stop
             endif

            ! Recalcular X(i,1)
           X(i,1) = (B(i,1) - sumTmp) / A(i,i) !
       end do
      ! Cálculo de norma euclidiana da diferença entre os vetores.
      sumTmp = 0
      sumTmp2 = 0
      do i = 1,n
         sumTmp  = sumTmp + (X(i,1) - Xold(i,1))**2
         sumTmp2 = sumTmp2 + X(i,1)**2
      enddo
      sumTmp = sqrt(sumTmp) ; sumTmp2 = sqrt(sumTmp2)
      sumTmp = sumTmp / sumTmp2
       if (sumTmp <= tol) then
           write(*,*) 'Convergência alcançada na iteração', iter
           exit
       endif
   end do

end subroutine jacobi_method
```

4.0 Método de Gauss-Seidel.

O método de Gauss-Seidel para solução de sistemas lineares é muito similar com o de Jacobi, de forma que a única diferença no algoritmo é que os valores atualizados da solução anteriores à um index qualquer ‘i’ são utilizados no cálculo de x(i). Além disso, a verificação para matrizes não diagonalmente dominantes não é necessária.

```
! Computar somatório fora da diagonal
sumTmp = 0.0
do j= 1,cA
   ! Para o método de Gauss - Seidel, se j < i então uso valor já atualizado para X(i),
   if (j .lt. i)then
      sumTmp = sumTmp + A(i,j) * X(j,1)
   else if(j .gt. i) then
      sumTmp = sumTmp + A(i,j) * Xold(j,1)
   endif
enddo
! Verificar se A é singular.
if(A(i,i) <= 1.0e-15) then
   write(*,*) 'WARNING: Singular Matrix'
   stop
 endif

! Recalcular X(i,1)
X(i,1) = (B(i,1) - sumTmp) / A(i,i) !
```

3. Testes com arquivos fornecidos.

Matriz 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

Vetor B 01 :

-5
0
-5
0
-5
0
-5
0
-5
0

Vetor B 02:

-10
0
-10
0
-10
0
-10
0
-10
0

Vetor B 03:

-15
0
-15
0
-15
0
-15
0
-15
0

3.1) Teste para decomposição LU:

```
abraaonote@abraaonote-IdeaPad-3-15ALC6:~/Downloads/Documentos/projetos/AlgLinComp_routines$ ./axb_solver.out
 Insira o caminho até o arquivo contendo a matrix A:
matrizes/Matriz_A.dat
Insira ICOD:

ICOD = 1 : Solver com decomposição LU
ICOD = 2 : Solver com decomposição de Cholesky
ICOD = 3 : Solver com método iterativo Jacobi
ICOD = 4 : Solver com método iterativo Gauss-Seidel
1
 Insira o caminho até o arquivo contendo B: (Caso deseja fechar o programa, digite -1)
matrizes/Vetor_B_01.dat
 Matrix solução X:
  -4.1666666666666634
  -3.3333333333333313
  -10.666666666666661
  -2.6666666666666661
  -13.499999999999996
  -2.9143354396410355E-015
  -10.666666666666666
   2.6666666666666665
  -4.1666666666666670
   3.3333333333333335
 Insira o caminho até o arquivo contendo B: (Caso deseja fechar o programa, digite -1)
matrizes/Vetor_B_02.dat
 Matrix solução X:
  -8.3333333333333268
  -6.6666666666666625
  -21.333333333333321
  -5.3333333333333321
  -26.999999999999993
  -5.8286708792820710E-015
  -21.333333333333332
   5.3333333333333330
  -8.3333333333333339
   6.6666666666666670
 Insira o caminho até o arquivo contendo B: (Caso deseja fechar o programa, digite -1)
matrizes/Vetor_B_03.dat
 Matrix solução X:
  -12.499999999999991
  -9.9999999999999911
  -31.999999999999979
  -7.9999999999999973
  -40.499999999999986
  -8.3932860661661812E-015
  -32.000000000000000
   8.0000000000000000
  -12.500000000000002
   10.000000000000002
 Insira o caminho até o arquivo contendo B: (Caso deseja fechar o programa, digite -1)
-1
```

3.2) Teste para decomposição de Cholesky:

```
C:\Users\abraa\Documents\projetos\AlgLinComp_routines>axb_solver
 Insira o caminho ate o arquivo contendo a matrix A:
matrizes/Matriz_A.dat
Insira ICOD:

ICOD = 1 : Solver com decomposi├º├úo LU
ICOD = 2 : Solver com decomposi├º├úo de Cholesky
ICOD = 3 : Solver com m├®todo iterativo Jacobi
ICOD = 4 : Solver com m├®todo iterativo Gauss-Seidel
2
 Fatoriza├º├úo de Cholesky bem sucedida.
 Insira o caminho ate o arquivo contendo B: (Caso deseja fechar o programa, digi
 te -1)
matrizes/Vetor_B_01.dat
 Matrix solucao X:
  -4.16666666666666
  -3.33333333333333
  -10.6666666666667
  -2.66666666666666
  -13.5000000000000
  2.616572154690808E-015
  -10.6666666666666
   2.66666666666666
  -4.16666666666666
   3.33333333333333
 Insira o caminho ate o arquivo contendo B: (Caso deseja fechar o programa, digi
 te -1)
matrizes/Vetor_B_02.dat
 Matrix solucao X:
  -8.33333333333333
  -6.66666666666666
  -21.3333333333333
  -5.33333333333332
  -27.0000000000000
  5.233144309381616E-015
  -21.3333333333333
   5.33333333333333
  -8.33333333333332
   6.66666666666665
 Insira o caminho ate o arquivo contendo B: (Caso deseja fechar o programa, digi
 te -1)
matrizes/Vetor_B_03.dat
 Matrix solucao X:
  -12.5000000000000
  -9.99999999999999
  -32.0000000000000
  -7.99999999999999
  -40.4999999999999
  7.735952457346736E-015
  -31.9999999999999
   7.99999999999999
  -12.5000000000000
   9.99999999999998
 Insira o caminho ate o arquivo contendo B: (Caso deseja fechar o programa, digi
 te -1)
-1
```

3.3) Teste para método de Jacobi:

```
C:\Users\abraa\Documents\projetos\AlgLinComp_routines>axb_solver
 Insira o caminho ate o arquivo contendo a matrix A:
matrizes/Matriz_A.dat
Insira ICOD:

ICOD = 1 : Solver com decomposi├º├úo LU
ICOD = 2 : Solver com decomposi├º├úo de Cholesky
ICOD = 3 : Solver com m├®todo iterativo Jacobi
ICOD = 4 : Solver com m├®todo iterativo Gauss-Seidel
3
 Insira o caminho ate o arquivo contendo B: (Caso deseja fechar o programa, digi
 te -1)
matrizes/Vetor_B_01.dat
 ---------------
 WARNING: Matrix isn't diagonally dominant, convergency not guaranteed.
 Converg├¬ncia alcan├ºada na itera├º├úo         508
 Matrix solucao X:
  -4.16402717970219
  -3.33110879996276
  -10.6595417973777
  -2.66473803129183
  -13.4908023357155
  0.000000000000000E+000
  -10.6595417973777
   2.66473803129183
  -4.16402717970219
   3.33110879996276
 Insira o caminho ate o arquivo contendo B: (Caso deseja fechar o programa, digi
 te -1)
matrizes/Vetor_B_02.dat
 ---------------
 WARNING: Matrix isn't diagonally dominant, convergency not guaranteed.
 Converg├¬ncia alcan├ºada na itera├º├úo         505
 Matrix solucao X:
  -8.32804394278431
  -6.66220882090168
  -21.3190274458924
  -5.32946086361241
  -26.9815683732504
  7.105427357601002E-016
  -21.3190274458924
   5.32946086361241
  -8.32804394278431
   6.66220882090168
 Insira o caminho ate o arquivo contendo B: (Caso deseja fechar o programa, digi
 te -1)
matrizes/Vetor_B_03.dat
 ---------------
 WARNING: Matrix isn't diagonally dominant, convergency not guaranteed.
 Converg├¬ncia alcan├ºada na itera├º├úo         504
 Matrix solucao X:
  -12.4920562625679
  -9.99330509707274
  -31.9785411688386
  -7.99419129541861
  -40.4723189274853
  7.105427357601002E-016
  -31.9785411688386
   7.99419129541861
  -12.4920562625679
   9.99330509707274
 Insira o caminho ate o arquivo contendo B: (Caso deseja fechar o programa, digi
 te -1)
-1
```

3.4) Teste para método de Gauss-Seidel:

```
C:\Users\abraa\Documents\projetos\AlgLinComp_routines>axb_solver
 Insira o caminho ate o arquivo contendo a matrix A:
matrizes/Matriz_A.dat
Insira ICOD:

ICOD = 1 : Solver com decomposi├º├úo LU
ICOD = 2 : Solver com decomposi├º├úo de Cholesky
ICOD = 3 : Solver com m├®todo iterativo Jacobi
ICOD = 4 : Solver com m├®todo iterativo Gauss-Seidel
4
 Insira o caminho ate o arquivo contendo B: (Caso deseja fechar o programa, digi
 te -1)
matrizes/Vetor_B_01.dat
 Converg├¬ncia alcan├ºada na itera├º├úo         276
 Matrix solucao X:
  -4.16533598940996
  -3.33221185177163
  -10.6631221210474
  -2.66570719137079
  -13.4954959790848
  5.329070518200751E-016
  -10.6632237207366
   2.66573469345494
  -4.16541118030750
   3.33227522191249
 Insira o caminho ate o arquivo contendo B: (Caso deseja fechar o programa, digi
 te -1)
matrizes/Vetor_B_02.dat
 Converg├¬ncia alcan├ºada na itera├º├úo         276
 Matrix solucao X:
  -8.33067484735918
  -6.66442612111963
  -21.3262518830672
  -5.33141645108125
  -26.9910016674818
 -3.552713678800501E-016
  -21.3264548634273
   5.33147139596333
  -8.33082506706533
   6.66455272479442
 Insira o caminho ate o arquivo contendo B: (Caso deseja fechar o programa, digi
 te -1)
matrizes/Vetor_B_03.dat
 Converg├¬ncia alcan├ºada na itera├º├úo         275
 Matrix solucao X:
  -12.4958960716569
  -9.99654124999568
  -31.9890683025074
  -7.99704089189062
  -40.4861091943980
 -1.421085471520200E-015
  -31.9893816450870
   7.99712571079171
  -12.4961279671477
   9.99673668920971
 Insira o caminho ate o arquivo contendo B: (Caso deseja fechar o programa, digi
```