```

*******************************************
./part6.f90

PROGRAM part6

  USE part6_mod

  IMPLICIT NONE

  INTEGER, PARAMETER :: m=4,n=4
  REAL, DIMENSION(m,n) :: mat1
  REAL, DIMENSION(n,m) :: mat2


  CHARACTER(LEN=*), PARAMETER :: fname='mat.txt'
  CHARACTER(LEN=*), PARAMETER :: fname2='mat.dat'

  !** We assume a file "mat.txt" has been created and exists
  !** inside the current working directory.

  mat1=getmatf(m,n,fname)   !*** Read in matrix from txt file
  mat2=transpose(mat1)      !*** Take the transpose and place in mat2
  CALL outmatf(mat2,fname)  !*** Write out transpose to same file

  mat1=0  !*** Destroy mat1

  CALL outmatf_u(mat2,fname2) !*** Output mat2 to binary file
  mat1=getmatf_u(m,n,fname2)  !*** Read from binary file into mat1

  CALL outmat(mat1) !*** output new mat1 to the screen



END PROGRAM part6

*******************************************


*******************************************
./part6_mod.f90

MODULE part6_mod

  IMPLICIT NONE

  CONTAINS

  ! ***********************************************************************

  FUNCTION infnorm(vec)
  !** Calculates the infinity norm of vector vec.
    !*** Dummy declarations
    REAL, DIMENSION(:), INTENT(IN)  :: vec
    !*** Local declarations
    REAL :: infnorm

    infnorm=MAXVAL(ABS(vec))

  END FUNCTION infnorm

 ! ***********************************************************************

  FUNCTION twonorm(vec)
  !** Calculates the Euclidean norm of vector vec.
    !*** Dummy declarations
    REAL, DIMENSION(:), INTENT(IN)  :: vec
```

```
    !*** Local declarations
    REAL :: twonorm

    twonorm=SQRT(SUM(vec**2))

  END FUNCTION twonorm

 ! ***********************************************************************

  FUNCTION cont(y,x,tol,max_iters)
  !*** Returns a logical type. If .TRUE. then the tolerance has been met or
  !*** the maximum number of iterations has been exceeded.

    !*** Dummy variables
    REAL, DIMENSION(:), INTENT(IN) :: y,x
    REAL, INTENT(IN) :: tol
    INTEGER, INTENT(IN) :: max_iters
    !*** Local declarations
    LOGICAL,DIMENSION(2) :: cont
    INTEGER, SAVE :: iter=0

    cont(1)=(infnorm(ABS(y-x)) < tol)
    cont(2)=(iter>max_iters)
    iter=iter+1

  END FUNCTION cont

  ! ***********************************************************************

  FUNCTION power1(mat,x,tol,eigv,flag,max_iters)
  !**** Func. to calc. the dominant eigenvalue and corresponding normalised
  !**** eigenvector of the (n) by (n) matrix [mat] the initial guess is the
  !**** input vector (x) and the method is considered to have converged if
  !**** absolute error is less than the given tolerance (tol). The
  !**** eigenvalue is returned in (eigv) and the eigenvector is
  !**** returned in x

    !**** Dummy arguments
    REAL, DIMENSION(:,:), INTENT(IN)  :: mat
    REAL, DIMENSION(:), INTENT(INOUT)  :: x
    REAL, INTENT(IN) :: tol
    REAL, INTENT(OUT) :: eigv
    LOGICAL,DIMENSION(:), INTENT(OUT) :: flag
    INTEGER :: max_iters


    !**** Local Declarations
    REAL, DIMENSION(SIZE(x))  :: y
    INTEGER :: power1
    INTEGER, DIMENSION(1) :: loc

    power1=0
    flag=.FALSE.
    !**** Normalise initial estimate
    x=x/infnorm(x)

    DO                               !** Iterate while "flag" is .TRUE.
      y=mulmatvec(mat,x)             !** Calculate new eigenvector estimate "y"
      loc=MAXLOC(ABS(y))             !** Location of largest magnitude in ABS()
      eigv=y(loc(1))/x(loc(1))       !** Estimate eigenvalue
      y=y/infnorm(y)                 !** Normalise new eigenvector estimate
      flag=cont(y,x,tol,max_iters) !** Decide if another iteration is needed
      x=y                            !** Update eigenvector "x" for next iteration
      power1=power1+1
      IF (flag(1) .OR. flag(2))  EXIT !** EXIT loop if any flag is true
    ENDDO

```

```
   END FUNCTION power1

! ********************************************************************

FUNCTION getmat(m,n)
!**** Function to input a matrix from the keyboard. The number of rows
!**** (m) and the number of columns (n) are input arguments to the
!**** function

  INTEGER, INTENT(IN) :: m,n       !**** Dummy declaration
  REAL, DIMENSION(m,n) :: getmat   !**** Local Declaration

  INTEGER :: i

  DO i=1,m
    PRINT '("Enter matrix row :",i2)',i !*** Prompt for row number
    READ*,getmat(i,:)                   !*** Read in row
  ENDDO

END FUNCTION getmat

! ********************************************************************

SUBROUTINE outmat(mat)
!**** Subroutine to output a matrix to the screen.

  REAL, DIMENSION(:,:), INTENT(IN)  :: mat  !*** Dummy declaration

  INTEGER :: i

  DO i=1,SIZE(mat,1)
    PRINT*,mat(i,:)
  ENDDO

END SUBROUTINE outmat

! ********************************************************************

FUNCTION mulmat(mat1,mat2)
!**** Function to input two matrices [mat1] & [mat2] and check if
!**** [mat1]*[mat2] is a valid matrix multiplication. If it is valid the
!**** matrix product [mat1]*[mat2] is returned.

  REAL, DIMENSION(:,:), INTENT(IN)  :: mat1
  REAL, DIMENSION(:,:), INTENT(IN)  :: mat2

  INTEGER :: m,n,k,i,j,p

  REAL, DIMENSION(SIZE(mat1,1),SIZE(mat2,2))  :: mulmat

  m=SIZE(mat1,1) ; n=SIZE(mat1,2) ; k=SIZE(mat2,2)

  !**** Perform the matrix multiplication
  !**** using three DO loops

  IF (SIZE(mat2,1) == n) THEN
    DO i=1,m               !*** For each row of getmat (mat3)
      DO j=1,k             !*** For each column of getmat (mat3)
        mulmat(i,j)=0      !*** initialise to zero
        DO p=1,n
          mulmat(i,j)=mulmat(i,j)+mat1(i,p)*mat2(p,j)
        ENDDO
      ENDDO
    ENDDO
  ELSE
    PRINT*,"Size mismatch in mulmat"
  ENDIF
```

```

END FUNCTION mulmat

! ********************************************************************

FUNCTION mulmatvec(mat,vec)
!**** Function to input a matrix [mat] and a vector [v] check if matrix
!**** vector multiplication is valid  w.r.t. their sizes. If it is then
!**** this function returns the matrix vector product.

  REAL, DIMENSION(:,:), INTENT(IN)  :: mat
  REAL, DIMENSION(:), INTENT(IN)  :: vec

  INTEGER :: m,n,k,i,j
  REAL, DIMENSION(SIZE(mat,1))  :: mulmatvec

  m=SIZE(mat,1) ; n=SIZE(mat,2) ; k=SIZE(mat,1)

  !**** Perform the matrix multiplication using three DO loops
  IF (n==k) THEN
    DO i=1,m
      mulmatvec(i)=0
      DO j=1,k
          mulmatvec(i)=mulmatvec(i)+mat(i,j)*vec(j)
      ENDDO
    ENDDO
  ELSE
    PRINT*,"Size mismatch in mulmatvec!"
  ENDIF

END FUNCTION mulmatvec

! ********************************************************************

FUNCTION transmat(mat)

  !**** Dummy arguments
  REAL, DIMENSION(:,:), INTENT(IN)  :: mat
  INTEGER :: m,n,i,j
  REAL, DIMENSION(SIZE(mat,2),SIZE(mat,1))  :: transmat !*** Return

  !**** Findout the no. of rows and cols in the matrix
  m=SIZE(mat,1) ; n=SIZE(mat,2)

  !**** Perform the transpose using two DO loops
  DO i=1,n            !*** Loop over rows in mat
    DO j=1,m          !*** Loop over cols in mat
      transmat(i,j)=mat(j,i)
    ENDDO
  ENDDO

END FUNCTION transmat


! ********************************************************************

FUNCTION transmat2(mat)

  REAL, DIMENSION(:,:), INTENT(IN)  :: mat

  REAL, DIMENSION(SIZE(mat,2),SIZE(mat,1))  :: transmat2


  transmat2=RESHAPE(mat,(/SIZE(mat,2),SIZE(mat,1)/),ORDER=(/2,1/))

END FUNCTION transmat2

```

```fortran
! ********************************************************************



FUNCTION getmatf(m,n,filename)
!**** Function to input a matrix from a text file. The number of rows
!**** (m) and the number of columns (n) are input arguments to the
!**** function along with the name of the file

  INTEGER, INTENT(IN) :: m,n               !**** Dummy declaration
  CHARACTER(LEN=*), INTENT(IN) :: filename !**** Dummy declaration

  REAL, DIMENSION(m,n) :: getmatf   !**** Local Declaration
  INTEGER :: i

  !***** Open the file passed in as the string "filename" on unit one

  OPEN(UNIT=1,FILE=filename, FORM="FORMATTED",STATUS="OLD",ACTION="READ")

  DO i=1,m !**** Do for each row
    READ(UNIT=1,FMT=*) getmatf(i,:) !*** Read in row at a time
  ENDDO

  CLOSE(UNIT=1) !***** Close the file

END FUNCTION getmatf

! ********************************************************************

SUBROUTINE outmatf(mat,filename)
!**** Function to output a matrix to a text file. The matrix to be
!**** written and the name of the file are the arguments


  CHARACTER(LEN=*), INTENT(IN) :: filename !**** Dummy declaration
  REAL, DIMENSION(:,:), INTENT(IN)  :: mat   !**** Dummy Declaration

  INTEGER :: m,i

  m=SIZE(mat,1)

  !***** Open the file passed in as the string "filename" on unit one

  OPEN(UNIT=1,FILE=filename, FORM="FORMATTED",STATUS="REPLACE",ACTION="WRITE")

  DO i=1,m !**** Do for each row
    WRITE(UNIT=1,FMT=*) mat(i,:) !*** Write out row at a time
  ENDDO

  CLOSE(UNIT=1) !***** Close the file

END SUBROUTINE outmatf


! ********************************************************************



FUNCTION getmatf_u(m,n,filename)
!**** Function to input a matrix from a binary file. The number of rows
!**** (m) and the number of columns (n) are input arguments to the
!**** function along with the name of the file

  INTEGER, INTENT(IN) :: m,n               !**** Dummy declaration
  CHARACTER(LEN=*), INTENT(IN) :: filename !**** Dummy declaration

  REAL, DIMENSION(m,n) :: getmatf_u  !**** Local Declaration
```

```fortran

  !***** Open the file passed in as the string "filename" on unit one
  OPEN(UNIT=1,FILE=filename, FORM="UNFORMATTED",STATUS="OLD",ACTION="READ")

  READ(UNIT=1) getmatf_u !*** Read WHOLE array

  CLOSE(UNIT=1) !***** Close the file

END FUNCTION getmatf_u

! ********************************************************************


SUBROUTINE outmatf_u(mat,filename)
!**** Function to output a matrix to a binary file. The matrix to be
!**** written and the name of the file are the arguments


  CHARACTER(LEN=*), INTENT(IN) :: filename !**** Dummy declaration
  REAL, DIMENSION(:,:), INTENT(IN)  :: mat   !**** Dummy Declaration

  !***** Open the file passed in as the string "filename" on unit one

  OPEN(UNIT=1,FILE=filename, FORM="UNFORMATTED",STATUS="REPLACE",ACTION="WRITE")

  !*** Write out whole array as no need now to write row by row
  !*** as we can not look at binary files like we can text files

  WRITE(UNIT=1) mat !*** Write out WHOLE array

  CLOSE(UNIT=1) !***** Close the file

END SUBROUTINE outmatf_u


END MODULE part6_mod
*******************************************

```