```
 1:
 2: ****************************************
 3: ./part4.f90
 4:
 5: PROGRAM part4
 6:
 7:   USE part4_mod
 8:
 9:   IMPLICIT NONE
10:
11:   INTEGER, PARAMETER :: m=3,n=3,max_iters=500
12:   REAL, DIMENSION(m,n) :: mat1
13:   REAL, DIMENSION(m) :: eigvec
14:   REAL :: tol=0.000001,eigval
15:   LOGICAL,DIMENSION(2) :: flag
16:   INTEGER :: iters
17:
18:   !mat1=getmat(m,n) !*** Input matrix1 from the keyboard
19:
20:   !** Explicitly assign matrix to mat1 for testing
21:   mat1(1,:)=(/1,5,3/)
22:   mat1(2,:)=(/6,3,5/)
23:   mat1(3,:)=(/2,8,5/)
24:   eigvec=(/1,1,1/)
25:
26:   !** Call the power method
27:   iters=power1(mat1,eigvec,tol,eigval,flag,max_iters)
28:
29:   PRINT*,"Estimate of eigenvalue : ",eigval
30:   PRINT '("Estimate of eigenvector : ",3f11.6)',eigvec(1),eigvec(2),eigvec(3)
31:
32:   IF (flag(1)) THEN
33:     PRINT*,"The method converged to the solution within the given tolerence."
34:   ELSE
35:      PRINT*,"The method did not to the solution within the given tolerence."
36:   ENDIF
37:
38:   IF (flag(2)) THEN
39:     PRINT*,"The maximum number of iterations was exceeded."
40:   ELSE
41:     PRINT*,"The maximum number of iterations was not exceeded."
42:     PRINT*,"Number of iterations taken : ",iters
43:   ENDIF
44:
45: END PROGRAM part4
46:
47: !******* OUTPUT FROM THE CODE WHEN RUN IS *********
48: ! Estimate of eigenvalue  :    13.06310
49: ! Estimate of eigenvector :    0.603874   0.856918   1.000000
50: ! The method converged to the solution within the given tolerence.
51: ! The maximum number of iterations was not exceeded.
52: ! Number of iterations taken :          12
53: ****************************************
54:
55:
56: ****************************************
57: ./part4_mod.f90
58:
59: MODULE part4_mod
60:
61:   IMPLICIT NONE
62:
63:   CONTAINS
64:
65:   ! ************************************************************************
66:
67:   FUNCTION infnorm(vec)
```



```
68:   !** Calculates the infinity norm of vector vec.
69:     !*** Dummy declarations
70:     REAL, DIMENSION(:), INTENT(IN)  :: vec
71:     !*** Local declarations
72:     REAL :: infnorm
73:
74:     infnorm=MAXVAL(ABS(vec))
75:
76:   END FUNCTION infnorm
77:
78:  ! ************************************************************************
79:
80:   FUNCTION twonorm(vec)
81:   !** Calculates the Euclidean norm of vector vec.
82:     !*** Dummy declarations
83:     REAL, DIMENSION(:), INTENT(IN)  :: vec
84:     !*** Local declarations
85:     REAL :: twonorm
86:
87:     twonorm=SQRT(SUM(vec**2))
88:
89:   END FUNCTION twonorm
90:
91:  ! ************************************************************************
92:
93:   FUNCTION cont(y,x,tol,max_iters)
94:   !*** Returns a logical type. If .TRUE. then the tolerance has been met or
95:   !*** the maximum number of iterations has been exceeded.
96:
97:     !*** Dummy variables
98:     REAL, DIMENSION(:), INTENT(IN) :: y,x
99:     REAL, INTENT(IN) :: tol
100:    INTEGER, INTENT(IN) :: max_iters
101:    !*** Local declarations
102:    LOGICAL,DIMENSION(2) :: cont
103:    INTEGER, SAVE :: iter=0
104:
105:    cont(1)=(infnorm(ABS(y-x)) < tol)
106:    cont(2)=(iter>max_iters)
107:    iter=iter+1
108:
109:  END FUNCTION cont
110:
111:  ! ************************************************************************
112:
113:  FUNCTION power1(mat,x,tol,eigv,flag,max_iters)
114:  !**** Func. to calc. the dominant eigenvalue and corresponding normalised
115:  !**** eigenvector of the (n) by (n) matrix [mat] the initial guess is the
116:  !**** input vector (x) and the method is considered to have converged if
117:  !**** absolute error is less than the given tolerance (tol). The
118:  !**** eigenvalue is returned (eigv) and the eigenvector is returned (x)
119:
120:    !**** Dummy arguments
121:    REAL, DIMENSION(:,:), INTENT(IN)  :: mat
122:    REAL, DIMENSION(:), INTENT(INOUT)  :: x
123:    REAL, INTENT(IN) :: tol
124:    REAL, INTENT(OUT) :: eigv
125:    LOGICAL,DIMENSION(:), INTENT(OUT) :: flag
126:    INTEGER :: max_iters
127:
128:    !**** Local Declarations
129:    REAL, DIMENSION(SIZE(x))  :: y
130:    INTEGER :: power1
131:
132:    power1=0
133:    flag=.FALSE.
134:    !**** Normalise initial estimate
```

```
  135:     x=x/infnorm(x)
  136:
  137:     DO                          !** Iterate while "flag" is .TRUE.
  138:       y=mulmatvec(mat,x)        !** Calculate new eigenvector estimate "y"
  139:       eigv=y(1)/x(1)        !** Estimate eigenvalue
  140:       y=y/infnorm(y)            !** Normalise new eigenvector estimate
  141:       flag=cont(y,x,tol,max_iters) !** Decide if another iteration is needed
  142:       x=y                       !** Update eigenvector "x" for next iteration
  143:       power1=power1+1
  144:       IF (flag(1) .OR. flag(2))  EXIT !** EXIT loop if any flag is true
  145:     ENDDO
  146:
  147:    END FUNCTION power1
  148:
  149: ! *********************************************************************
  150:
  151: FUNCTION getmat(m,n)
  152: !**** Function to input a matrix from the keyboard. The number of rows
  153: !**** (m) and the number of columns (n) are input arguments to the
  154: !**** function
  155:
  156:   INTEGER, INTENT(IN) :: m,n       !**** Dummy declaration
  157:   REAL, DIMENSION(m,n) :: getmat   !**** Local Declaration
  158:    INTEGER :: i
  159:
  160:   DO i=1,m
  161:     PRINT '("Enter matrix row :",i2)',i !*** Prompt for row number
  162:     READ*,getmat(i,:)                   !*** Read in row
  163:   ENDDO
  164:
  165: END FUNCTION getmat
  166:
  167: ! *********************************************************************
  168:
  169: SUBROUTINE outmat(mat)
  170: !**** Subroutine to output a matrix to the screen.
  171:
  172:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat  !*** Dummy declaration
  173:
  174:   INTEGER :: i
  175:
  176:   DO i=1,SIZE(mat,1)
  177:     PRINT*,mat(i,:)
  178:   ENDDO
  179:
  180: END SUBROUTINE outmat
  181:
  182: ! *********************************************************************
  183:
  184: FUNCTION mulmat(mat1,mat2)
  185: !**** Function to input two matrices [mat1] & [mat2] and check if
  186: !**** [mat1]*[mat2] is a valid matrix multiplication. If it is valid the
  187: !**** matrix product [mat1]*[mat2] is returned.
  188:
  189:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat1
  190:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat2
  191:    INTEGER :: m,n,k,i,j,p
  192:   REAL, DIMENSION(SIZE(mat1,1),SIZE(mat2,2))  :: mulmat
  193:
  194:   m=SIZE(mat1,1) ; n=SIZE(mat1,2) ; k=SIZE(mat2,2)
  195:
  196:   !**** Perform the matrix multiplication
  197:   !**** using three DO loops
  198:
  199:   IF (SIZE(mat2,1) == n) THEN
  200:     DO i=1,m               !*** For each row of getmat (mat3)
  201:       DO j=1,k             !*** For each column of getmat (mat3)
```



```
  202:         mulmat(i,j)=0      !*** initialise to zero
  203:         DO p=1,n
  204:           mulmat(i,j)=mulmat(i,j)+mat1(i,p)*mat2(p,j)
  205:         ENDDO
  206:       ENDDO
  207:     ENDDO
  208:   ELSE
  209:     PRINT*,"Size mismatch in mulmat"
  210:   ENDIF
  211:
  212: END FUNCTION mulmat
  213:
  214: ! *********************************************************************
  215:
  216: FUNCTION mulmatvec(mat,vec)
  217: !**** Function to input a matrix [mat] and a vector [v] check if matrix
  218: !**** vector multiplication is valid  w.r.t. their sizes. If it is then
  219: !**** this function returns the matrix vector product.
  220:
  221:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat
  222:   REAL, DIMENSION(:), INTENT(IN)  :: vec
  223:
  224:   INTEGER :: m,n,k,i,j
  225:   REAL, DIMENSION(SIZE(mat,1))  :: mulmatvec
  226:
  227:   m=SIZE(mat,1) ; n=SIZE(mat,2) ; k=SIZE(mat,1)
  228:
  229:   !**** Perform the matrix multiplication using three DO loops
  230:   IF (n==k) THEN
  231:     DO i=1,m
  232:       mulmatvec(i)=0
  233:       DO j=1,k
  234:           mulmatvec(i)=mulmatvec(i)+mat(i,j)*vec(j)
  235:       ENDDO
  236:     ENDDO
  237:   ELSE
  238:     PRINT*,"Size mismatch in mulmatvec!"
  239:   ENDIF
  240:
  241: END FUNCTION mulmatvec
  242:
  243: ! *********************************************************************
  244:
  245: FUNCTION transmat(mat)
  246:
  247:   !**** Dummy arguments
  248:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat
  249:   INTEGER :: m,n,i,j
  250:   REAL, DIMENSION(SIZE(mat,2),SIZE(mat,1))  :: transmat !*** Return
  251:
  252:   !**** Findout the no. of rows and cols in the matrix
  253:   m=SIZE(mat,1) ; n=SIZE(mat,2)
  254:
  255:   !**** Perform the transpose using two DO loops
  256:   DO i=1,n             !*** Loop over rows in mat
  257:     DO j=1,m           !*** Loop over cols in mat
  258:       transmat(i,j)=mat(j,i)
  259:     ENDDO
  260:   ENDDO
  261:
  262: END FUNCTION transmat
  263:
  264: END MODULE part4_mod
  265: ****************************************
  266:
```