```
 1:
 2: ****************************************
 3: ./part5_mod.f90
 4:
 5: MODULE part5_mod
 6:
 7:   IMPLICIT NONE
 8:
 9:   CONTAINS
10:
11:   ! ************************************************************************
12:
13:   FUNCTION infnorm(vec)
14:   !** Calculates the infinity norm of vector vec.
15:     !*** Dummy declarations
16:     REAL, DIMENSION(:), INTENT(IN)  :: vec
17:     !*** Local declarations
18:     REAL :: infnorm
19:
20:     infnorm=MAXVAL(ABS(vec))
21:
22:   END FUNCTION infnorm
23:
24:  ! ************************************************************************
25:
26:   FUNCTION twonorm(vec)
27:   !** Calculates the Euclidean norm of vector vec.
28:     !*** Dummy declarations
29:     REAL, DIMENSION(:), INTENT(IN)  :: vec
30:     !*** Local declarations
31:     REAL :: twonorm
32:
33:     twonorm=SQRT(SUM(vec**2))
34:
35:   END FUNCTION twonorm
36:
37:  ! ************************************************************************
38:
39:   FUNCTION cont(y,x,tol,max_iters)
40:   !*** Returns a logical type. If .TRUE. then the tolerance has been met or
41:   !*** the maximum number of iterations has been exceeded.
42:
43:     !*** Dummy variables
44:     REAL, DIMENSION(:), INTENT(IN) :: y,x
45:     REAL, INTENT(IN) :: tol
46:     INTEGER, INTENT(IN) :: max_iters
47:     !*** Local declarations
48:     LOGICAL,DIMENSION(2) :: cont
49:     INTEGER, SAVE :: iter=0
50:
51:     cont(1)=(infnorm(ABS(y-x)) < tol)
52:     cont(2)=(iter>max_iters)
53:     iter=iter+1
54:
55:   END FUNCTION cont
56:
57:   ! ************************************************************************
58:
59:   FUNCTION power1(mat,x,tol,eigv,flag,max_iters)
60:   !**** Func. to calc. the dominant eigenvalue and corresponding normalised
61:   !**** eigenvector of the (n) by (n) matrix [mat] the initial guess is the
62:   !**** input vector (x) and the method is considered to have converged if
63:   !**** absolute error is less than the given tolerance (tol). The
64:   !**** eigenvalue is returned in (eigv) and the eigenvector is
65:   !**** returned in x
66:
67:     !**** Dummy arguments
```



```
68:     REAL, DIMENSION(:,:), INTENT(IN)  :: mat
69:     REAL, DIMENSION(:), INTENT(INOUT)  :: x
70:     REAL, INTENT(IN) :: tol
71:     REAL, INTENT(OUT) :: eigv
72:     LOGICAL,DIMENSION(:), INTENT(OUT) :: flag
73:     INTEGER :: max_iters
74:
75:
76:     !**** Local Declarations
77:     REAL, DIMENSION(SIZE(x))  :: y
78:     INTEGER :: power1
79:     INTEGER, DIMENSION(1) :: loc
80:
81:     power1=0
82:     flag=.FALSE.
83:     !**** Normalise initial estimate
84:     x=x/infnorm(x)
85:
86:     DO                              !** Iterate while "flag" is .TRUE.
87:       y=mulmatvec(mat,x)            !** Calculate new eigenvector estimate "y"
88:       loc=MAXLOC(ABS(y))            !** Location of largest magnitude in ABS()
89:       eigv=y(loc(1))/x(loc(1))      !** Estimate eigenvalue
90:       y=y/infnorm(y)                !** Normalise new eigenvector estimate
91:       flag=cont(y,x,tol,max_iters) !** Decide if another iteration is needed
92:       x=y                           !** Update eigenvector "x" for next iteration
93:       power1=power1+1
94:       IF (flag(1) .OR. flag(2))  EXIT !** EXIT loop if any flag is true
95:     ENDDO
96:
97:    END FUNCTION power1
98:
99: ! ************************************************************************
100:
101: FUNCTION getmat(m,n)
102: !**** Function to input a matrix from the keyboard. The number of rows
103: !**** (m) and the number of columns (n) are input arguments to the
104: !**** function
105:
106:   INTEGER, INTENT(IN) :: m,n       !**** Dummy declaration
107:   REAL, DIMENSION(m,n) :: getmat   !**** Local Declaration
108:
109:   INTEGER :: i
110:
111:   DO i=1,m
112:     PRINT '("Enter matrix row :",i2)',i !*** Prompt for row number
113:     READ*,getmat(i,:)                  !*** Read in row
114:   ENDDO
115:
116: END FUNCTION getmat
117:
118: ! ************************************************************************
119:
120: SUBROUTINE outmat(mat)
121: !**** Subroutine to output a matrix to the screen.
122:
123:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat  !*** Dummy declaration
124:
125:   INTEGER :: i
126:
127:   DO i=1,SIZE(mat,1)
128:     PRINT*,mat(i,:)
129:   ENDDO
130:
131: END SUBROUTINE outmat
132:
133: ! ************************************************************************
134:
```

```
  135: FUNCTION mulmat(mat1,mat2)
  136: !**** Function to input two matrices [mat1] & [mat2] and check if
  137: !**** [mat1]*[mat2] is a valid matrix multiplication. If it is valid the
  138: !**** matrix product [mat1]*[mat2] is returned.
  139:
  140:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat1
  141:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat2
  142:
  143:   INTEGER :: m,n,k,i,j,p
  144:
  145:   REAL, DIMENSION(SIZE(mat1,1),SIZE(mat2,2))  :: mulmat
  146:
  147:   m=SIZE(mat1,1) ; n=SIZE(mat1,2) ; k=SIZE(mat2,2)
  148:
  149:   !**** Perform the matrix multiplication
  150:   !**** using three DO loops
  151:
  152:   IF (SIZE(mat2,1) == n) THEN
  153:     DO i=1,m               !*** For each row of getmat (mat3)
  154:       DO j=1,k             !*** For each column of getmat (mat3)
  155:         mulmat(i,j)=0      !*** initialise to zero
  156:         DO p=1,n
  157:           mulmat(i,j)=mulmat(i,j)+mat1(i,p)*mat2(p,j)
  158:         ENDDO
  159:       ENDDO
  160:     ENDDO
  161:   ELSE
  162:     PRINT*,"Size mismatch in mulmat"
  163:   ENDIF
  164:
  165: END FUNCTION mulmat
  166:
  167: ! ********************************************************************
  168:
  169: FUNCTION mulmatvec(mat,vec)
  170: !**** Function to input a matrix [mat] and a vector [v] check if matrix
  171: !**** vector multiplication is valid  w.r.t. their sizes. If it is then
  172: !**** this function returns the matrix vector product.
  173:
  174:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat
  175:   REAL, DIMENSION(:), INTENT(IN)  :: vec
  176:
  177:   INTEGER :: m,n,k,i,j
  178:   REAL, DIMENSION(SIZE(mat,1))  :: mulmatvec
  179:
  180:   m=SIZE(mat,1) ; n=SIZE(mat,2) ; k=SIZE(mat,1)
  181:
  182:   !**** Perform the matrix multiplication using three DO loops
  183:   IF (n==k) THEN
  184:     DO i=1,m
  185:       mulmatvec(i)=0
  186:       DO j=1,k
  187:           mulmatvec(i)=mulmatvec(i)+mat(i,j)*vec(j)
  188:       ENDDO
  189:     ENDDO
  190:   ELSE
  191:     PRINT*,"Size mismatch in mulmatvec!"
  192:   ENDIF
  193:
  194: END FUNCTION mulmatvec
  195:
  196: ! ********************************************************************
  197:
  198: FUNCTION transmat(mat)
  199:
  200:   !**** Dummy arguments
  201:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat
```



```
  202:   INTEGER :: m,n,i,j
  203:   REAL, DIMENSION(SIZE(mat,2),SIZE(mat,1))  :: transmat !*** Return
  204:
  205:   !**** Findout the no. of rows and cols in the matrix
  206:   m=SIZE(mat,1) ; n=SIZE(mat,2)
  207:
  208:   !**** Perform the transpose using two DO loops
  209:   DO i=1,n            !*** Loop over rows in mat
  210:     DO j=1,m          !*** Loop over cols in mat
  211:       transmat(i,j)=mat(j,i)
  212:     ENDDO
  213:   ENDDO
  214:
  215: END FUNCTION transmat
  216:
  217:
  218: ! ********************************************************************
  219:
  220: FUNCTION transmat2(mat)
  221:
  222:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat
  223:
  224:   REAL, DIMENSION(SIZE(mat,2),SIZE(mat,1))  :: transmat2
  225:
  226:
  227:   transmat2=RESHAPE(mat,(/SIZE(mat,2),SIZE(mat,1)/),ORDER=(/2,1/))
  228:
  229: END FUNCTION transmat2
  230:
  231: ! ********************************************************************
  232:
  233: END MODULE part5_mod
  234: ****************************************
  235:
  236:
  237: ****************************************
  238: ./part5.f90
  239:
  240: PROGRAM part5
  241:
  242:   USE part5_mod
  243:
  244:   IMPLICIT NONE
  245:
  246:   INTEGER, PARAMETER :: m=3,n=3,max_iters=500
  247:   REAL, DIMENSION(m,n) :: mat1
  248:   REAL, DIMENSION(m) :: eigvec
  249:   REAL :: tol=0.000001,eigval
  250:   LOGICAL,DIMENSION(2) :: flag
  251:   INTEGER :: iters
  252:
  253:   !mat1=getmat(m,n) !*** Input matrix1 from the keyboard
  254:
  255:   !** Explicitly assign matrix to mat1 for testing
  256:   mat1(1,:)=(/1,5,3/)
  257:   mat1(2,:)=(/6,3,5/)
  258:   mat1(3,:)=(/2,8,5/)
  259:   eigvec=(/1,1,1/)
  260:
  261:   !** Call the power method
  262:   iters=power1(mat1,eigvec,tol,eigval,flag,max_iters)
  263:
  264:   PRINT*,"Estimate of eigenvalue : ",eigval
  265:   PRINT '("Estimate of eigenvector : ",3f11.6)',eigvec(1),eigvec(2),eigvec(3)
  266:
  267:   IF (flag(1)) THEN
  268:     PRINT*,"The method converged to the solution within the given tolerence."
```

```
  269:   ELSE
  270:      PRINT*,"The method did not to the solution within the given tolerence."
  271:   ENDIF
  272:
  273:   IF (flag(2)) THEN
  274:     PRINT*,"The maximum number of iterations was exceeded."
  275:   ELSE
  276:     PRINT*,"The maximum number of iterations was not exceeded."
  277:     PRINT*,"Number of iterations taken : ",iters
  278:   ENDIF
  279:
  280: END PROGRAM part5
  281:
  282: !******* OUTPUT FROM THE CODE WHEN RUN IS *********
  283: ! Estimate of eigenvalue  :    13.06310
  284: ! Estimate of eigenvector :    0.603874   0.856918   1.000000
  285: ! The method converged to the solution within the given tolerence.
  286: ! The maximum number of iterations was not exceeded.
  287: ! Number of iterations taken :           12
  288: ****************************************
  289:
```