

```

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
12:   REAL, DIMENSION(m,n) :: mat1
13:   REAL, DIMENSION(m,n) :: mat2
14:   REAL, DIMENSION(m) :: eigvec
15:   REAL :: tol=0.000001,eigval
16:   LOGICAL :: flag
17:
18:   !mat1=getmat(m,n) !** Input matrix1 from the keyboard
19:
20:   mat1(1,:)=(/1.5,3/)
21:   mat1(2,:)=(/6.3,5/)
22:   mat1(3,:)=(/2.8,5/)
23:
24:   !** mat1(1,:)=(/-4.14,0/)
25:   !** mat1(2,:)=(/-5.13,0/)
26:   !** mat1(3,:)=(/-1.0,2/)
27:
28:   PRINT*, "Original Matrix"
29:   PRINT*, ""
30:   CALL outmat(mat1)
31:
32:   mat2=transmat2(mat1)
33:
34:   PRINT*, ""
35:   PRINT*, "Matrix Transpose"
36:   PRINT*, ""
37:
38:   CALL outmat(mat2)
39:
40:
41: END PROGRAM part4
42: *****
43:
44:
45: *****
46: ./part4_mod.f90
47:
48: MODULE part4_mod
49:
50: IMPLICIT NONE
51:
52: CONTAINS
53:
54: ! *****
55:
56: FUNCTION infnorm(vec)
57: !** Calculates the infinity norm of vector vec.
58:
59: !** Dummy declarations
60: REAL, DIMENSION(:), INTENT(IN) :: vec
61:
62: !** Local declarations
63: REAL :: infnorm
64:
65: infnorm=MAXVAL(ABS(vec))
66:
67: END FUNCTION infnorm

```

```

68:
69: ! *****
70: FUNCTION twonorm(vec)
71: !** Calculates the Euclidean norm of vector vec.
72:
73:
74: !** Dummy declarations
75: REAL, DIMENSION(:), INTENT(IN) :: vec
76:
77: !** Local declarations
78: REAL :: twonorm
79:
80: twonorm=SQRT(SUM(vec**2))
81:
82: END FUNCTION twonorm
83:
84: ! *****
85:
86: FUNCTION cont(y,x,tol)
87: !** Returns a logical type. If .TRUE. then the tolerance has been met or
88: !** the maximum number of iterations has been exceeded.
89:
90: !** Dummy variables
91: REAL, DIMENSION(:), INTENT(IN) :: y,x
92: REAL, INTENT(IN) :: tol
93:
94: !** Local declarations
95: LOGICAL,DIMENSION(2) :: cont
96: INTEGER, SAVE :: iter=0
97:
98: cont(1)=(infnorm(ABS(y-x)) < tol)
99: cont(2)=(iter>10000)
100: iter=iter+1
101:
102: END FUNCTION cont
103:
104: ! *****
105:
106: FUNCTION power1(mat,x,tol,eigv,flag)
107: !** Func. to calc. the dominant eigenvalue and corresponding normalised
108: !** eigenvector of the (n) by (n) matrix [mat] the initial guess is the
109: !** input vector (x) and the method is considered to have converged if
110: !** absolute error is less than the given tolerance (tol). The
111: !** eigenvalue is returned in (eigv) and the eigenvector is
112: !** returned in x
113:
114: !** Dummy arguments
115: REAL, DIMENSION(:,:), INTENT(IN) :: mat
116: REAL, DIMENSION(:), INTENT(INOUT) :: x
117: REAL, INTENT(IN) :: tol
118: REAL, INTENT(OUT) :: eigv
119: LOGICAL,DIMENSION(:), INTENT(OUT) :: flag
120:
121: INTEGER :: power1
122:
123: !** Local Declarations
124: REAL, DIMENSION(SIZE(x)) :: y
125: INTEGER, DIMENSION(1) :: i
126: power1=0
127: flag=.FALSE.
128: !** Normalise initial estimate
129: x=x/infnorm(x)
130:
131: DO
132: IF (flag(1) .OR. flag(2)) EXIT !** Iterate while "flag" is .TRUE.
133: y=mulmatvec(mat,x) !** Calculate new eigenvector estimate "y"
134: i=MAXLOC(ABS(y)) !** Find location of largest value
135: eigv=y(i(1))/x(i(1)) !** Estimate eigenvalue

```

```

135: y=y/infnorm(y)      !** Normalise new eigenvector estimate
136: IF (eigv < 0) y=-y  !** Negate y or not converge
137: flag=cont(y,x,tol)  !** Decide if another iteration is needed
138: x=y                 !** Update eigenvector "x" for next iteration
139: power1=power1+1
140: ENDDO
141:
142:
143: END FUNCTION power1
144:
145: ! *****
146:
147: FUNCTION getmat(m,n)
148: !**** Function to input a matrix from the keyboard. The number of rows
149: !**** (m) and the number of columns (n) are input arguments to the
150: !**** function
151:
152: INTEGER, INTENT(IN) :: m,n      !**** Dummy declaration
153: REAL, DIMENSION(m,n) :: getmat !**** Local Declaration
154:
155: INTEGER :: i
156:
157: DO i=1,m
158: PRINT '( "Enter matrix row :",i2)',i !** Prompt for row number
159: READ*,getmat(i,:) !** Read in row
160: ENDDO
161:
162: END FUNCTION getmat
163:
164: ! *****
165:
166: SUBROUTINE outmat(mat)
167: !**** Subroutine to output a matrix to the screen.
168:
169: REAL, DIMENSION(:,:), INTENT(IN) :: mat !** Dummy declaration
170:
171: INTEGER :: i
172:
173: DO i=1,SIZE(mat,1)
174: PRINT*,mat(i,:)
175: ENDDO
176:
177: END SUBROUTINE outmat
178:
179: ! *****
180:
181: FUNCTION mulmat(mat1,mat2)
182: !**** Function to input two matrices [mat1] & [mat2] and check if
183: !**** [mat1]*[mat2] is a valid matrix multiplication. If it is valid the
184: !**** matrix product [mat1]*[mat2] is returned.
185:
186: REAL, DIMENSION(:,:), INTENT(IN) :: mat1
187: REAL, DIMENSION(:,:), INTENT(IN) :: mat2
188:
189: INTEGER :: m,n,k,i,j,p
190:
191: REAL, DIMENSION(SIZE(mat1,1),SIZE(mat2,2)) :: mulmat
192:
193: m=SIZE(mat1,1) ; n=SIZE(mat1,2) ; k=SIZE(mat2,2)
194:
195: !**** Perform the matrix multiplication
196: !**** using three DO loops
197:
198: IF (SIZE(mat2,1) == n) THEN
199: DO i=1,m
200: DO j=1,k
201: mulmat(i,j)=0

```

```

202: DO p=1,n
203: mulmat(i,j)=mulmat(i,j)+mat1(i,p)*mat2(p,j)
204: ENDDO
205: ENDDO
206: ENDDO
207: ELSE
208: PRINT*, "Size mismatch in mulmat"
209: ENDIF
210:
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:
226: REAL, DIMENSION(SIZE(mat,1)) :: mulmatvec
227:
228: m=SIZE(mat,1) ; n=SIZE(mat,2) ; k=SIZE(mat,1)
229:
230: !**** Perform the matrix multiplication
231: !**** using three DO loops
232:
233: IF (n=k) THEN
234: DO i=1,m
235: mulmatvec(i)=0
236: mulmatvec(i)=mulmatvec(i)+SUM(mat(i,:)*vec(:))
237: ENDDO
238: ELSE
239: PRINT*, "Size mismatch in mulmatvec!"
240: ENDIF
241:
242: END FUNCTION mulmatvec
243:
244: ! *****
245:
246: FUNCTION transmat(mat)
247:
248: !**** Dummy arguments
249: REAL, DIMENSION(:,:), INTENT(IN) :: mat
250: INTEGER :: m,n,i,j
251: REAL, DIMENSION(SIZE(mat,2),SIZE(mat,1)) :: transmat !** Return
252:
253: !**** Findout the no. of rows and cols in the matrix
254: m=SIZE(mat,1) ; n=SIZE(mat,2)
255:
256:
257: !**** Perform the transpose
258: !**** using two DO loops
259:
260: DO i=1,m
261: DO j=1,n
262: transmat(i,j)=mat(j,i)
263: ENDDO
264: ENDDO
265:
266: END FUNCTION transmat
267:
268: ! *****

```

```
269:
270: FUNCTION transmat2(mat)
271:
272:   REAL, DIMENSION(:,:), INTENT(IN) :: mat
273:
274:   REAL, DIMENSION(SIZE(mat,2),SIZE(mat,1)) :: transmat2
275:
276:
277:   transmat2=RESHAPE(mat,(/SIZE(mat,2),SIZE(mat,1)/),ORDER=(/2,1/))
278: END FUNCTION transmat2
279:
280:
281: ! *****
282:
283: END MODULE part4_mod
284: *****
285:
```