

```

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:
16: !** Dummy declarations
17: REAL, DIMENSION(:), INTENT(IN) :: vec
18:
19: !** Local declarations
20: REAL :: infnorm
21:
22: infnorm=MAXVAL(ABS(vec))
23:
24: END FUNCTION infnorm
25: ! *****
26: ! *****
27:
28: FUNCTION twonorm(vec)
29: !** Calculates the Euclidean norm of vector vec.
30:
31: !** Dummy declarations
32: REAL, DIMENSION(:), INTENT(IN) :: vec
33:
34: !** Local declarations
35: REAL :: twonorm
36:
37: twonorm=SQRT(SUM(vec**2))
38:
39: END FUNCTION twonorm
40: ! *****
41: ! *****
42:
43: FUNCTION cont(y,x,tol)
44: !** Returns a logical type. If .TRUE. then the tolerance has been met or
45: !** the maximum number of iterations has been exceeded.
46:
47: !** Dummy variables
48: REAL, DIMENSION(:), INTENT(IN) :: y,x
49: REAL, INTENT(IN) :: tol
50:
51: !** Local declarations
52: LOGICAL cont, converged, unditers
53: INTEGER, SAVE :: iter=0
54:
55: converged=(infnorm(ABS(y-x)) < tol)
56: unditers=(iter<100)
57: cont=( (.NOT. converged) .AND. unditers)
58: iter=iter+1
59:
60: END FUNCTION cont
61: ! *****
62: ! *****
63:
64: SUBROUTINE power1(mat,x,tol,eigv,converged)
65: !** Func. to calc. the dominant eigenvalue and corresponding normalised
66: !** eigenvector of the (n) by (n) matrix [mat] the initial guess is the
67: !** input vector (x) and the method is considered to have converged if

```

```

68: !** absolute error is less than the given tolerance (tol). The
69: !** eigenvalue is returned in (eigv) and the eigenvector is
70: !** returned in x
71:
72: !** Dummy arguments
73: REAL, DIMENSION(:), INTENT(IN) :: mat
74: REAL, DIMENSION(:), INTENT(INOUT) :: x
75: REAL, INTENT(IN) :: tol
76: REAL, INTENT(OUT) :: eigv
77: LOGICAL, INTENT(OUT) :: converged
78:
79: !** Local Declarations
80: REAL, DIMENSION(SIZE(x)) :: y
81: LOGICAL :: flag=.TRUE. !** Always do one iteration
82: INTEGER, DIMENSION(1) :: i
83: !** Normalise initial estimate
84: x=x/infnorm(x)
85:
86: DO
87: IF (.NOT. flag) EXIT
88: y=mulmatvec(mat,x)
89: i=MAXLOC(ABS(y))
90: eigv=y(i(1))/x(i(1))
91: y=y/infnorm(y)
92: flag=cont(y,x,tol)
93: converged=(infnorm(ABS(y-x)) < tol) !** Set return converge test
94: x=y
95: ENDDO
96:
97:
98: END SUBROUTINE power1
99: ! *****
100: ! *****
101:
102: FUNCTION getmat(m,n)
103: !** Function to input a matrix from the keyboard. The number of rows
104: !** (m) and the number of columns (n) are input arguments to the
105: !** function
106:
107: INTEGER, INTENT(IN) :: m,n !** Dummy declaration
108: REAL, DIMENSION(m,n) :: getmat !** Local Declaration
109:
110: INTEGER :: i
111:
112: DO i=1,m
113: PRINT '( "Enter matrix row : ",i2)', i !** Prompt for row number
114: READ*, getmat(i,:) !** Read in row
115: ENDDO
116:
117: END FUNCTION getmat
118: ! *****
119: ! *****
120:
121: SUBROUTINE outmat(mat)
122: !** Subroutine to output a matrix to the screen.
123:
124: REAL, DIMENSION(:), INTENT(IN) :: mat !** Dummy declaration
125:
126: INTEGER :: i
127:
128: DO i=1,SIZE(mat,1)
129: PRINT*, mat(i,:)
130: ENDDO
131:
132: END SUBROUTINE outmat
133: ! *****
134: ! *****

```

```

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

```

```

202: FUNCTION transmat(mat)
203:
204: !**** Dummy arguments
205: REAL, DIMENSION(:,:), INTENT(IN) :: mat
206: INTEGER :: m,n,i,j
207: REAL, DIMENSION(SIZE(mat,2),SIZE(mat,1)) :: transmat !**** Return
208:
209: !**** Findout the no. of rows and cols in the matrix
210:
211: m=SIZE(mat,1) ; n=SIZE(mat,2)
212:
213: !**** Perform the transpose
214: !**** using two DO loops
215:
216: DO i=1,n
217:   DO j=1,m
218:     transmat(i,j)=mat(j,i)
219:   ENDDO
220: ENDDO
221:
222: END FUNCTION transmat
223:
224: ! ****
225:
226: FUNCTION transmat2(mat)
227:
228: REAL, DIMENSION(:,:), INTENT(IN) :: mat
229:
230: REAL, DIMENSION(SIZE(mat,2),SIZE(mat,1)) :: transmat2
231:
232:
233: transmat2=RESHAPE(mat,(/SIZE(mat,2),SIZE(mat,1)/),ORDER=(/2,1/))
234:
235: END FUNCTION transmat2
236:
237: ! ****
238:
239:
240: FUNCTION getmatf(m,n,filename)
241: !**** Function to input a matrix from a text file. The number of rows
242: !**** (m) and the number of columns (n) are input arguments to the
243: !**** function along with the name of the file
244:
245: INTEGER, INTENT(IN) :: m,n
246: CHARACTER(LEN=*) , INTENT(IN) :: filename !**** Dummy declaration
247:
248: REAL, DIMENSION(m,n) :: getmatf !**** Local Declaration
249: INTEGER :: i
250:
251: !**** Open the file passed in as the string "filename" on unit one
252:
253: OPEN(UNIT=1,FILE=filename,FORM="FORMATTED",STATUS="OLD",ACTION="READ")
254:
255: DO i=1,m !**** Do for each row
256:   READ(UNIT=1,FMT=*) getmatf(i,:) !**** Read in row at a time
257: ENDDO
258:
259: CLOSE(UNIT=1) !**** Close the file
260:
261: END FUNCTION getmatf
262:
263: ! ****
264:
265: SUBROUTINE outmatf(mat,filename)
266: !**** Function to output a matrix to a text file. The matrix to be
267: !**** written and the name of the file are the arguments
268:

```

```

269: CHARACTER(LEN=*) , INTENT(IN) :: filename !**** Dummy declaration
270: REAL, DIMENSION(:), INTENT(IN) :: mat !**** Dummy Declaration
271: INTEGER :: m,i
272: !**** Open the file passed in as the string "filename" on unit one
273: OPEN(UNIT=1, FILE=filename, FORM="FORMATTED", STATUS="REPLACE", ACTION="WRITE")
274: !**** Do for each row
275: DO i=1,m !**** Do for each row
276: WRITE(UNIT=1,FMT=*) mat(i,:) !**** Write out row at a time
277: ENDDO
278: CLOSE(UNIT=1) !**** Close the file
279: END SUBROUTINE outmatf
280: ! ****
281: FUNCTION getmatf_u(m,n,filename)
282: !**** Function to input a matrix from a binary file. The number of rows
283: !**** (m) and the number of columns (n) are input arguments to the
284: !**** function along with the name of the file
285: INTEGER, INTENT(IN) :: m,n !**** Dummy declaration
286: CHARACTER(LEN=*) , INTENT(IN) :: filename !**** Dummy declaration
287: REAL, DIMENSION(m,n) :: getmatf_u !**** Local Declaration
288: !**** Open the file passed in as the string "filename" on unit one
289: OPEN(UNIT=1, FILE=filename, FORM="UNFORMATTED", STATUS="OLD", ACTION="READ")
290: READ(UNIT=1) getmatf_u !**** Read WHOLE array
291: CLOSE(UNIT=1) !**** Close the file
292: END FUNCTION getmatf_u
293: ! ****
294: SUBROUTINE outmatf_u(mat,filename)
295: !**** Function to output a matrix to a binary file. The matrix to be
296: !**** written and the name of the file are the arguments
297: CHARACTER(LEN=*) , INTENT(IN) :: mat !**** Dummy Declaration
298: REAL, DIMENSION(LEN=*) , PARAMETER :: fname='mat.txt',
299: CHARACTER(LEN=*) , PARAMETER :: fname2='mat.dat',
300: !** We assume a file "mat.txt" has been created and exists
301: !** inside the current working directory.
302: mat1=getmatf(m,n,filename) !** Read in matrix from txt file
303: mat2=transpose(mat1) !** Take the transpose and place in mat2
304: CALL outmatf(mat2, fname) !** Write out transpose to same file
305: mat1=0 !** Destroy mat1
306: CALL outmatf_u(mat2, fname2) !** Output mat2 to binary file
307: mat1=getmatf_u(m,n, fname2) !** Read from binary file into mat1
308: CALL outmat(mat1) !** output new mat1 to the screen
309: END SUBROUTINE outmatf_u
310: ! ****
311: ! ****
312: ! ****
313: ! ****
314: ! ****
315: ! ****
316: ! ****
317: ! ****
318: ! ****
319: ! ****
320: ! ****
321: ! ****
322: ! ****
323: ! ****
324: ! ****
325: ! ****
326: ! ****
327: ! ****
328: ! ****
329: ! ****
330: ! ****
331: ! ****
332: ! ****
333: ! ****
334: ! ****
335: ! ****

```

```

336: ! ****
337: ! ****
338: ! ****
339: FUNCTION mulvecmat(vec,mat)
340: !**** Function to premultiply mat with vec
341: !**** Use assumed shape arrays for dummy arrays****
342: REAL, DIMENSION(:), INTENT(IN) :: vec !**** Dummy declaration
343: REAL, DIMENSION(:,:), INTENT(IN) :: mat !**** Dummy declaration
344: !**** Local Declarations ****
345: INTEGER :: j,m,n
346: REAL, DIMENSION(SIZE(mat,2)) :: mulvecmat
347: !**** Find out the matrix size for the DO loop limit ****
348: n=SIZE(mat,2)
349: m=SIZE(mat,1)
350: !**** Perform the vector matrix multiplication
351: !**** For each element of mulvecmat
352: DO j=1,n
353: mulvecmat(j)=SUM(vec*mat(:,j))
354: ENDDO
355: ELSE
356: PRINT*, "Size mismatch in mulvecmat"
357: ENDDIF
358: END FUNCTION mulvecmat
359: ! ****
360: END MODULE part5_mod
361: ! ****
362: ! ****
363: ! ****
364: ! ****
365: ! ****
366: ! ****
367: ! ****
368: ! ****
369: ! ****
370: ! ****
371: ! ****
372: ! ****
373: ! ****
374: ! ****
375: ! ****
376: ! ****
377: ! ****
378: ! ****
379: ! ****
380: ! ****
381: ! ****
382: ! ****
383: ! ****
384: ! ****
385: ! ****
386: ! ****
387: ! ****
388: ! ****
389: ! ****
390: ! ****
391: ! ****
392: ! ****
393: ! ****
394: ! ****
395: ! ****
396: ! ****
397: ! ****
398: ! ****
399: ! ****
400: ! ****
401: ! ****
402: ! ****

```

solutions.txt Fri Dec 16 12:49:59 2011 7

403: END PROGRAM part5
404:
405: *****
406: