

```

1:
2: *****
3: ./code/linear_mod.f90
4:
5: MODULE linear_mod
6:
7: IMPLICIT NONE
8:
9: CONTAINS
10: ! *****
11: ! *****
12: FUNCTION gauss_elim(mat1,vv)
13: *****
14: !**** Function to reduce a square matrix to upper triangular form
15: !****
16: !**** Dummy Variables *****
17: REAL,DIMENSION(:,:), INTENT(INOUT) :: mat1 !** Matrix to be reduced
18: REAL,DIMENSION(:,:), INTENT(INOUT) :: vv !** RHS vector
19:
20: LOGICAL :: gauss_elim !** Set return type for function gauss_elim
21:
22: REAL :: lambda !** multiplier variable
23: INTEGER :: n,i,j
24:
25: gauss_elim=.TRUE. !** Set default to success
26: n=SIZE(mat1,1)
27:
28: DO j=1,n-1
29: IF (mat1(j,j)==0) gauss_elim=find_pivot(mat1,vv,j) !** Loop over each pivot column
30: IF (gauss_elim.EQV(.FALSE.)) EXIT !** Check zero pivot
31: DO i=j+1,n !** Loop to zero all the elements under pivot
32: IF (mat1(j,i).NE. 0) THEN
33: lambda=mat1(i,j)/mat1(j,j)
34: mat1(i,j:n)=mat1(i,j:n)-lambda*mat1(j,j:n)
35: vv(i)=vv(i)-lambda*vv(j) !** Keep RHS vector consistent
36: ENDIF
37: END DO
38: END DO
39: IF (mat1(n,n)==0) gauss_elim=.FALSE. !** Check last element
40: END FUNCTION gauss_elim
41:
42: ! *****
43: ! *****
44: FUNCTION find_pivot(mat1,vv,index)
45: REAL,DIMENSION(:,:), INTENT(INOUT) :: mat1
46: INTEGER, INTENT(IN) :: index
47: !** When pivot is zero find valid row for interchange
48: !** and make the interchange
49:
50: REAL,DIMENSION(:,:), INTENT(INOUT) :: vv
51:
52: !**** Local Declarations *****
53:
54: INTEGER :: loop
55: LOGICAL :: find_pivot
56: REAL,DIMENSION(SIZE(mat1,1)) :: tmp_row
57: REAL :: tmp_elem
58:
59: find_pivot=.FALSE.
60: DO loop=index+1,SIZE(mat1,1)
61: IF (mat1(loop,index).NE. 0) THEN
62: find_pivot=.TRUE. !** Found good pivot element
63: tmp_row=mat1(loop,:) !** Make a copy of new found row in temp
64: mat1(loop,:)=mat1(index,:) !**
65: mat1(index,:)=tmp_row !** Swap rows in matrix
66: tmp_elem=vv(loop) !**
67:

```

```

68: vv(loop)=vv(index) !** Swap elements in RHS Vector
69: vv(index)=tmp_elem !**
70: EXIT
71: ENDDIF
72: END DO
73:
74: END FUNCTION find_pivot
75:
76: ! *****
77: FUNCTION back_sub(mat1,vv)
78: !** Perform back substitution on an upper triang. matrix
79:
80: REAL,DIMENSION(:,:), INTENT(IN) :: mat1
81: REAL,DIMENSION(:,:), INTENT(IN) :: vv
82: REAL,DIMENSION(SIZE(vv)) :: back_sub
83: INTEGER :: n
84: INTEGER :: i
85:
86: n=SIZE(vv) ; back_sub(n)=vv(n)/mat1(n,n)
87:
88: DO i=n-1,1,-1
89: back_sub(i)=(vv(i)-SUM(mat1(i,i+1:n)*back_sub(i+1:n)))/mat1(i,i)
90: END DO
91:
92: END FUNCTION back_sub
93:
94: ! *****
95: ! *****
96: FUNCTION getmat(m,n)
97: !**** Function to input a matrix from the keyboard. The number of rows
98: !**** (m) and the number of columns (n) are input arguments to the
99: !**** function
100:
101: INTEGER, INTENT(IN) :: m,n !**** Dummy declaration
102: REAL, DIMENSION(m,n) :: getmat !**** Local Declaration
103:
104: INTEGER :: i
105:
106: DO i=1,m
107: PRINT '( "Enter matrix row :",i2)',i !** Prompt for row number
108: READ*,getmat(i,:) !** Read in row
109: ENDDO
110:
111: END FUNCTION getmat
112:
113: ! *****
114: ! *****
115: FUNCTION getvec(m)
116: !**** Function to input a vector from the keyboard. The number of elements
117: !**** (m) are input arguments to the function
118: !**** is (m)
119:
120: INTEGER, INTENT(IN) :: m !**** Dummy declaration
121: REAL, DIMENSION(m) :: getvec !**** Local Declaration
122:
123: INTEGER :: i
124:
125: DO i=1,m
126: PRINT '( "Enter vector element :",i2)',i !** Prompt for row number
127: READ*,getvec(i) !** Read in row
128: ENDDO
129:
130: END FUNCTION getvec
131:
132: ! *****
133: ! *****
134: SUBROUTINE outmat(mat)

```

```

135: !*** Subroutine to output a matrix to the screen.
136:
137: REAL, DIMENSION(:,:), INTENT(IN) :: mat !*** Dummy declaration
138:
139: INTEGER :: i
140:
141: DO i=1,SIZE(mat,1)
142: PRINT*,mat(i,:)
143: ENDDO
144:
145: END SUBROUTINE outmat
146:
147: ! *****
148:
149: END MODULE linear_mod
150:
151:
152: ! ===== RESULTS =====
153: !
154: ! Upper triangular matrix is.
155: ! 1.000000 -1.000000 2.000000 -1.000000
156: ! 0.000000E+00 2.000000 -1.000000 1.000000
157: ! 0.000000E+00 0.000000E+00 -1.000000 -1.000000
158: ! 0.000000E+00 0.000000E+00 0.000000E+00 2.000000
159:
160: ! Solution vector is.
161:
162: ! -7.000000 3.000000 2.000000 2.000000
163:
164: *****
165:
166: *****
167: *****
168: ./code/linear.f90
169:
170:
171: PROGRAM gaussian
172:
173: !****
174: !*** Main program to solve gaussian elimination
175: !*** problems.
176: !****
177:
178: USE linear_mod !*** Module holding required routines
179:
180: IMPLICIT NONE
181:
182: INTEGER, PARAMETER :: n=4
183: REAL, DIMENSION(n,n) :: mat1 !** Matrix to reduce
184: REAL, DIMENSION(n) :: vv !** RHS Vector
185:
186: mat1(1,:)=(/1,-1,-2,-1/)
187: mat1(2,:)=(/2,-2,3,-3/)
188: mat1(3,:)=(/1,1,0/)
189: mat1(4,:)=(/1,-1,4,3/)
190:
191: vv=(/-8,-20,-2,4/)
192:
193: IF (gauss_elim(mat1,vv)) THEN
194: PRINT*, "Upper triangular matrix is."
195: CALL outmat(mat1)
196: PRINT*, "Solution vector is."
197: PRINT*, back_sub(mat1,vv)
198: !PRINT*, vv
199: ELSE
200: PRINT*, "The Matrix could not be reduced to triangular form."
201: ENDIF

```

```

202:
203: END PROGRAM gaussian
204: *****
205: *****

```