

```

1  ! =====
2  !           2-Dimensional Finite Element Analysis
3  !           of Steady Ideal Flows Governed by Laplace Eq.
4  !
5  !           * Galerkin F.E.M.
6  !           * Gaussian Elimination used to solve the Linear Systems
7  !
8  !                                           2006. 4.22. coded by S.Tanaka
9  !                                           ( Fortran90 Version )
10 ! =====
11 !-- MAIN PROGRAM -
12 program Laplace
13 implicit none
14 !Mesh
15 integer                :: node, nelm          ! Total number of nodes and elements
16 integer, allocatable  :: nc(:, :)           ! Node connection data in element
17 double precision, allocatable :: xx(:), yy(:) ! X, Y in Cartesian coordinate planes
18 double precision, allocatable :: area(:)      ! Area of Element
19 double precision, allocatable :: dx(:, :), dy(:, :) ! \frac{\partial N}{\partial x}
20 !Boundary Condition
21 integer                :: ipbc              ! Total number of nodes on \Gamma_{1}
22 integer, allocatable  :: npbc(:)           ! Number of node on \Gamma_{1}
23 double precision, allocatable :: fpbc(:)     ! \hat{\phi} on \Gamma_{1}
24 !Physical values
25 double precision      :: kx, ky            ! Coefficient of permeability
26 double precision, allocatable :: phi(:)     ! Potential values on node
27 double precision, allocatable :: ea(:, :)   ! L.H.S. Matrix
28 double precision, allocatable :: ue(:), ve(:) ! Velocity ( on Element )
29
30 integer :: i, j, n, m
31
32 character(50) :: mesfile, dbcfile, outfile
33 !
34 ! FILE I/O
35 open(9, file = 'file.dat', status = 'unknown' )
36 read(9,*) kx, ky
37 read(9,'(a)') mesfile
38 read(9,'(a)') dbcfile
39 read(9,'(a)') outfile
40 close(9)
41 open(10, file = mesfile, status = 'old')
42 open(11, file = dbcfile, status = 'old')
43 open(60, file = outfile, status = 'replace')
44
45 ! READ MESH
46 read(10,*) node, nelm
47 allocate( xx(node), yy(node), nc(3,nelm) )
48 do i = 1, node
49 read(10,*) n, xx(n), yy(n)
50 enddo
51 do j = 1, nelm
52 read(10,*) m, ( nc(i,m), i = 1, 3 )
53 enddo
54 close(10)
55
56 ! READ BOUNDARY CONDITION
57 read(11,*) ipbc
58 allocate( npbc(ipbc), fpbc(ipbc) )
59 do i = 1, ipbc
60 read(11,*) n, npbc(n), fpbc(n)
61 enddo
62 close(11)
63
64 ! Dynamic Memory Allocation
65 allocate( phi(node), ea(node,node), ue(nelm), ve(nelm), &
66 area(nelm), dx(3,nelm), dy(3,nelm) )
67 !=====
68 call makeSF & ! Make Shape Function
69 ( node, nelm, xx, yy, nc, &
70 area, dx, dy )
71 call makLHS & ! Make L.H.S. Matrix
72 ( node, nelm, nc, dx, dy, area, ea, &
73 kx, ky )
74 call boundc & ! Impose Boundary Condition

```

```

75     ( node,    ea,    ipbc,    npbc,    fpbc,    phi )
76     call sweep & ! Solver
77     ( node,    ea,    phi )
78     call calvel & ! Calculate Velocity
79     ( node,    nelm,    nc,    dx,    dy,    phi, &
80     ue,    ve,    kx,    ky )
81 !=====
82     call output & ! Result output
83     ( node,    nelm,    phi,    ue,    ve )
84 end program Laplace
85 !-- MAIN PROGRAM --
86 !
87 !-- SUBROUTINES --
88 ! -----
89     subroutine makeSF & ! Make Shape Function
90     ( node,    nelm,    xx,    yy,    nc,    &
91     area,    dx,    dy )
92 ! -----
93     implicit none
94     integer, intent(in) :: nelm, node, nc(3,nelm)
95     double precision, intent(in) :: xx(node), yy(node)
96     double precision, intent(out) :: area(nelm), dx(3,nelm), dy(3,nelm)
97     integer :: m, n1, n2, n3
98     double precision :: x1, x2, x3, y1, y2, y3, a02, a02i
99 !
100    do m = 1, nelm
101        n1 = nc(1,m)
102        n2 = nc(2,m)
103        n3 = nc(3,m)
104        x1 = xx(n1)
105        x2 = xx(n2)
106        x3 = xx(n3)
107        y1 = yy(n1)
108        y2 = yy(n2)
109        y3 = yy(n3)
110        a02 = (x1 - x2) * (y1 - y3) - (x1 - x3) * (y1 - y2)
111        if( a02 <= 0.0d0 ) then
112            write(6,*) ' AREA FAILURE ', m
113        endif
114        a02i = 1.d0 / a02
115        area(m) = 0.5d0 * a02
116        dx(1,m) = (y2 - y3) * a02i
117        dx(2,m) = (y3 - y1) * a02i
118        dx(3,m) = (y1 - y2) * a02i
119        dy(1,m) = (x3 - x2) * a02i
120        dy(2,m) = (x1 - x3) * a02i
121        dy(3,m) = (x2 - x1) * a02i
122    enddo
123 !
124 end subroutine makeSF
125 !
126 ! -----
127     subroutine makLHS & ! Make L.H.S. Matrix
128     ( node,    nelm,    nc,    dx,    dy,    area,    ea, &
129     kx,    ky )
130 ! -----
131     implicit none
132     integer, intent(in) :: node, nelm, nc(3,nelm)
133     double precision, intent(in) :: dx(3,nelm), dy(3,nelm), area(nelm), kx, ky
134     double precision, intent(out) :: ea(node,node)
135     double precision :: a01, dx1, dx2, dx3, dy1, dy2, dy3
136     double precision :: ea11, ea12, ea13, ea22, ea23, ea33
137     integer :: m, n1, n2, n3
138
139     ea(:, :) = 0.0d0
140     do m = 1, nelm
141         n1 = nc(1,m)
142         n2 = nc(2,m)
143         n3 = nc(3,m)
144         dx1 = dx(1,m)
145         dx2 = dx(2,m)
146         dx3 = dx(3,m)
147         dy1 = dy(1,m)
148         dy2 = dy(2,m)

```

```

149     dy3 = dy(3,m)
150     a01 = area(m)
151     ea11 = (dx1 * dx1 * kx + dy1 * dy1 * ky) * a01
152     ea12 = (dx1 * dx2 * kx + dy1 * dy2 * ky) * a01
153     ea13 = (dx1 * dx3 * kx + dy1 * dy3 * ky) * a01
154     ea22 = (dx2 * dx2 * kx + dy2 * dy2 * ky) * a01
155     ea23 = (dx2 * dx3 * kx + dy2 * dy3 * ky) * a01
156     ea33 = (dx3 * dx3 * kx + dy3 * dy3 * ky) * a01
157     ea(n1,n1) = ea(n1,n1) + ???????
158     ea(n1,n2) = ea(n1,n2) + ???????
159     ea(n1,n3) = ea(n1,n3) + ???????
160     ea(n2,n1) = ea(n2,n1) + ???????
161     ea(n2,n2) = ea(n2,n2) + ???????
162     ea(n2,n3) = ea(n2,n3) + ???????
163     ea(n3,n1) = ea(n3,n1) + ???????
164     ea(n3,n2) = ea(n3,n2) + ???????
165     ea(n3,n3) = ea(n3,n3) + ???????
166     enddo
167 !
168     end subroutine makLHS
169 !
170 ! -----
171     subroutine boundc & ! Impose Boundary Condition
172     ( node, ea, ibc, nbc, fbc, xx )
173 ! -----
174     implicit none
175     integer, intent(in) :: node, ibc, nbc(ibc)
176     double precision, intent(in) :: fbc(ibc)
177     double precision, intent(out) :: xx(node)
178     double precision, intent(inout) :: ea(node,node)
179     integer :: n, ib, in
180     double precision :: f, ea_nn
181 !
182     xx = 0.0d0
183     do ib = 1, ibc
184         n = nbc(ib)
185         f = fbc(ib)
186         ea_nn = ea(n,n)
187         do in = 1, node
188             xx(in) = xx(in) - ea(in,n) * f
189             ea(n, in) = 0.0d0
190             ea(in, n) = 0.0d0
191         enddo
192         ea(n,n) = 1.0d0
193         xx(n) = f
194     enddo
195 !
196     end subroutine boundc
197 !
198 ! -----
199     subroutine sweep (ndof, a, x) ! Solve Linear-Systems
200 ! -----
201     implicit none
202     integer, intent(in) :: ndof
203     double precision, intent(inout) :: a(ndof,ndof), x(ndof)
204 !
205     integer :: i, j, k, il, l
206     double precision :: ai, cc
207 !
208     do i = 1, ndof
209         if( dabs(a(i,i)) <= 1.0d0-10 ) then
210             write(6,*) 'Diagonal Value',i,'is ZERO!'
211             stop
212         endif
213 !
214         ai = 1.0d0 / a(i,i)
215         x(i) = x(i) * ai
216         do j = 1, ndof
217             a(i,j) = a(i,j) * ai
218         enddo
219 !
220         if( i /= ndof ) then
221             il = i + 1
222             do k = il, ndof

```

```

223         cc = a(k,i)
224         do j = i1, ndof
225             a(k,j) = a(k,j) - cc * a(i,j)
226         enddo
227         x(k) = x(k) - cc * x(i)
228     enddo
229 endif
230 enddo
231 !
232 do i = 1, ndof-1
233     j = ndof - i
234     do k = 1, i
235         l = ndof+1-k
236         x(j) = x(j) - a(j,l) * x(l)
237     enddo
238 enddo
239 !
240 end subroutine sweep
241 !
242 ! -----
243 subroutine calvel & ! Calculate Velocity
244     ( node, nelm, nc, dx, dy, phi, &
245     ue, ve, kx, ky )
246 ! -----
247 implicit none
248 integer, intent(in) :: node, nelm, nc(3,nelm)
249 double precision, intent(in) :: dx(3,nelm), dy(3,nelm), phi(node), kx, ky
250 double precision, intent(out) :: ue(nelm), ve(nelm)
251 integer :: i, m
252 !
253 ue(:) = 0.0d0
254 ve(:) = 0.0d0
255 do m = 1, nelm
256     do i = 1, 3
257         ue(m) = ue(m) - ??????????
258         ve(m) = ve(m) - ??????????
259     enddo
260 enddo
261 end subroutine calvel
262 !
263 ! -----
264 subroutine output & ! Result output
265     ( node, nelm, phi, ue, ve )
266 ! -----
267 implicit none
268 integer, intent(in) :: node, nelm
269 double precision, intent(in) :: phi(node), ue(nelm), ve(nelm)
270 integer :: n
271 !
272 write(60,600) ( n, phi(n), n =1, node )
273 write(60,601) ( n, ue(n), ve(n), n =1, nelm )
274 close(60)
275 600 format(i7, d15.6)
276 601 format(i7,2d15.6)
277 !
278 end subroutine output

```