

第 3 回演習プログラム

新領域創成科学研究科 人間環境学専攻
橋本 学

第 3 回演習資料のプログラムを以下に示します.

第 2 節のプログラム：モジュール mod_nodes3d, mod_localelement3d, mod_elements3d の作成

1. 節点モジュール mod_nodes3d.f90

```
1:      MODULE mod_nodes3d
2: !#####
3:
4:      IMPLICIT NONE
5:
6: !-----
7:
8:      TYPE :: struct_nodes3d
9:
10:     !-----
11:
12:     PRIVATE
13:
14:     !-----
15:     !
16:     ! n
17:     ! The total number of nodes
18:     !
19:     !-----
20:     !
21:     ! x(:, :)
22:     ! Cartesian coordinates of a node
```

```

23:      ! (x, y, z)
24:      !
25:      !-----
26:      !
27:      ! u(:)
28:      ! Unknown
29:      !
30:      ! bc(:)
31:      ! Boundary condition of unknown
32:      !
33:      !-----
34:
35:      INTEGER :: n
36:      INTEGER, ALLOCATABLE :: bc(:)
37:
38:      REAL(8), ALLOCATABLE :: x(:, :)
39:      REAL(8), ALLOCATABLE :: u(:)
40:
41:      !-----
42:
43:      END TYPE struct_nodes3d
44:
45: !-----
46:
47:      CONTAINS
48:
49:
50:      ! Set the total number of nodes
51: !#####
52:      SUBROUTINE set_nodes3d_n(ns3d, n)
53: !#####
54:
55:      TYPE(struct_nodes3d), INTENT(INOUT) :: ns3d
56:
57:      INTEGER, INTENT(IN) :: n
58:

```

```

59: !-----
60:
61:     ns3d%n = n
62:
63: !-----
64:
65:     RETURN
66:
67: !#####
68:     END SUBROUTINE set_nodes3d_n
69: !#####
70:
71:
72:     ! Get the total number of nodes
73: !#####
74:     SUBROUTINE get_nodes3d_n(ns3d, n)
75: !#####
76:
77:     TYPE(struct_nodes3d), INTENT(IN) :: ns3d
78:
79:     INTEGER, INTENT(OUT) :: n
80:
81: !-----
82:
83:     n = ns3d%n
84:
85: !-----
86:
87:     RETURN
88:
89: !#####
90:     END SUBROUTINE get_nodes3d_n
91: !#####
92:
93:
94:     ! Set Cartesian coordinates of a node

```

```

95: !#####
96:     SUBROUTINE set_nodes3d_x(ns3d, x)
97: !#####
98:
99:     TYPE(struct_nodes3d), INTENT(INOUT) :: ns3d
100:
101:     REAL(8), INTENT(IN) :: x(:, :)
102:
103: !-----
104:
105:     ns3d%x = x
106:
107: !-----
108:
109:     RETURN
110:
111: !#####
112:     END SUBROUTINE set_nodes3d_x
113: !#####
114:
115:
116:     ! Get Cartesian coordinates of a node
117: !#####
118:     SUBROUTINE get_nodes3d_x(ns3d, x)
119: !#####
120:
121:     TYPE(struct_nodes3d), INTENT(IN) :: ns3d
122:
123:     REAL(8), INTENT(OUT) :: x(:, :)
124:
125: !-----
126:
127:     INTEGER :: i
128:     INTEGER :: id
129:
130: !-----

```

```

131:
132:     x = ns3d%x
133:
134: !-----
135:
136:     RETURN
137:
138: !#####
139:     END SUBROUTINE get_nodes3d_x
140: !#####
141:
142:
143:     ! Set unknown
144: !#####
145:     SUBROUTINE set_nodes3d_u(ns3d, u)
146: !#####
147:
148:     TYPE(struct_nodes3d), INTENT(INOUT) :: ns3d
149:
150:     REAL(8), INTENT(IN) :: u(:)
151:
152: !-----
153:
154:     ns3d%u = u
155:
156: !-----
157:
158:     RETURN
159:
160: !#####
161:     END SUBROUTINE set_nodes3d_u
162: !#####
163:
164:
165:     ! Get unknown
166: !#####

```

```

167:      SUBROUTINE get_nodes3d_u(ns3d, u)
168: !#####
169:
170:      TYPE(struct_nodes3d), INTENT(IN) :: ns3d
171:
172:      REAL(8), INTENT(OUT) :: u(:)
173:
174: !-----
175:
176:      u = ns3d%u
177:
178: !-----
179:
180:      RETURN
181:
182: !#####
183:      END SUBROUTINE get_nodes3d_u
184: !#####
185:
186:
187:      ! Set boundary condition of unknown
188: !#####
189:      SUBROUTINE set_nodes3d_bc(ns3d, bc)
190: !#####
191:
192:      CLASS(struct_nodes3d), INTENT(INOUT) :: ns3d
193:
194:      INTEGER, INTENT(IN) :: bc(:)
195:
196: !-----
197:
198:      ns3d%bc = bc
199:
200: !-----
201:
202:      RETURN

```

```

203:
204: !#####
205:     END SUBROUTINE set_nodes3d_bc
206: !#####
207:
208:
209:     ! Get boundary condition of unknown
210: !#####
211:     SUBROUTINE get_nodes3d_bc(ns3d, bc)
212: !#####
213:
214:     CLASS(struct_nodes3d), INTENT(IN) :: ns3d
215:
216:     INTEGER, INTENT(OUT) :: bc(:)
217:
218: !-----
219:
220:     bc = ns3d%bc
221:
222: !-----
223:
224:     RETURN
225:
226: !#####
227:     END SUBROUTINE get_nodes3d_bc
228: !#####
229:
230:
231:     ! Initialize nodes3d
232: !#####
233:     SUBROUTINE init_nodes3d(ns3d, n)
234: !#####
235:
236:     TYPE(struct_nodes3d), INTENT(INOUT) :: ns3d
237:
238:     INTEGER, INTENT(IN) :: n

```

```

239:
240: !-----
241:
242:     ns3d%n = n
243:
244:     !-----
245:
246:     ALLOCATE ( ns3d%x (3, n) )
247:
248:     ns3d%x = 0.0D0
249:
250:     !-----
251:
252:     ALLOCATE ( ns3d%u (3*n) )
253:     ALLOCATE ( ns3d%bc (3*n) )
254:
255:     ns3d%u = 0.0D0
256:     ns3d%bc = 0
257:
258: !-----
259:
260:     RETURN
261:
262: !#####
263:     END SUBROUTINE init_nodes3d
264: !#####
265:
266:
267:     ! Delete nodes3d
268: !#####
269:     SUBROUTINE del_nodes3d (ns3d)
270: !#####
271:
272:     TYPE (struct_nodes3d), INTENT (INOUT) :: ns3d
273:
274: !-----

```



```

275:
276:     IF( ns3d%n .EQ. 0 ) THEN
277:
278:         RETURN
279:
280:     END IF
281:
282: !-----
283:
284:     ns3d%n = 0
285:
286:     !-----
287:
288:     DEALLOCATE( ns3d%x )
289:
290:     !-----
291:
292:     DEALLOCATE( ns3d%u )
293:     DEALLOCATE( ns3d%bc )
294:
295: !-----
296:
297:     RETURN
298:
299: !#####
300:     END SUBROUTINE del_nodes3d
301: !#####
302:
303:
304: !#####
305:     END MODULE mod_nodes3d

```

2. 有限要素 (計算空間) モジュール mod_localelement3d.f90

```

1:     MODULE mod_localelement3d
2: !#####

```

```

3:
4:     IMPLICIT NONE
5:
6: !-----
7:
8:     TYPE :: struct_localelement3d
9:
10:    !-----
11:
12:    PRIVATE
13:
14:    !-----
15:    !
16:    ! nboundaries
17:    ! The total number of local element boundaries
18:    !
19:    ! nedges
20:    ! The total number of local element edges
21:    !
22:    ! volume
23:    ! Local element volume
24:    !
25:    ! area_boundary
26:    ! Local element boundary area
27:    !
28:    ! nnodes
29:    ! The total number of local nodes
30:    !
31:    ! nnodes_boundary
32:    ! The total number of nodes on a local element boundary
33:    !
34:    ! nedges_boundary
35:    ! The total number of edges on a local element boundary
36:    !
37:    ! nnodes_edge
38:    ! The total number of nodes on a local element edge

```

```

39:      !
40:      ! xi(:, :)
41:      ! Cartesian coordinates of a local node
42:      ! xi, eta, zeta
43:      !
44:      ! table_na(:, :)
45:      ! Table of local element boundary no. and local node no.
46:      !
47:      !-----
48:
49:      INTEGER :: nboundaries
50:      INTEGER :: nedges
51:      INTEGER :: nedges_boundary
52:      INTEGER :: nnodes
53:      INTEGER :: nnodes_boundary
54:      INTEGER :: nnodes_edge
55:      INTEGER, ALLOCATABLE :: table_na(:, :)
56:
57:      REAL(8) :: volume
58:      REAL(8) :: area_boundary
59:      REAL(8), ALLOCATABLE :: xi(:, :)
60:
61:      !-----
62:
63:      END TYPE struct_localelement3d
64:
65: !-----
66:
67:      CONTAINS
68:
69:
70:      ! Set the total number of local element boundaries
71: !#####
72:      SUBROUTINE set_localelement3d_nboundaries(le3d, nboundaries)
73: !#####
74:

```

```

75:      TYPE(struct_localelement3d), INTENT(INOUT) :: le3d
76:
77:      INTEGER, INTENT(IN) :: nboundaries
78:
79: !-----
80:
81:      le3d%nboundaries = nboundaries
82:
83: !-----
84:
85:      RETURN
86:
87: !#####
88:      END SUBROUTINE set_localelement3d_nboundaries
89: !#####
90:
91:
92:      ! Get the total number of local element boundaries
93: !#####
94:      SUBROUTINE get_localelement3d_nboundaries(le3d, nboundaries)
95: !#####
96:
97:      TYPE(struct_localelement3d), INTENT(IN) :: le3d
98:
99:      INTEGER, INTENT(OUT) :: nboundaries
100:
101: !-----
102:
103:      nboundaries = le3d%nboundaries
104:
105: !-----
106:
107:      RETURN
108:
109: !#####
110:      END SUBROUTINE get_localelement3d_nboundaries

```

```

111: !#####
112:
113:
114:      ! Get the total number of local element edges
115: !#####
116:      SUBROUTINE get_localelement3d_nedges(le3d, nedges)
117: !#####
118:
119:      TYPE(struct_localelement3d), INTENT(IN) :: le3d
120:
121:      INTEGER, INTENT(OUT) :: nedges
122:
123: !-----
124:
125:      nedges = le3d%nedges
126:
127: !-----
128:
129:      RETURN
130:
131: !#####
132:      END SUBROUTINE get_localelement3d_nedges
133: !#####
134:
135:
136:      ! Get local element volume
137: !#####
138:      SUBROUTINE get_localelement3d_volume(le3d, volume)
139: !#####
140:
141:      TYPE(struct_localelement3d), INTENT(IN) :: le3d
142:
143:      REAL(8), INTENT(OUT) :: volume
144:
145: !-----
146:

```

```

147:      volume = le3d%volume
148:
149: !-----
150:
151:      RETURN
152:
153: !#####
154:      END SUBROUTINE get_localelement3d_volume
155: !#####
156:
157:
158:      ! Get local element boundary area
159: !#####
160:      SUBROUTINE get_localelement3d_area_boundary(le3d, area_boundary)
161: !#####
162:
163:      TYPE(struct_localelement3d), INTENT(IN) :: le3d
164:
165:      REAL(8), INTENT(OUT) :: area_boundary
166:
167: !-----
168:
169:      area_boundary = le3d%area_boundary
170:
171: !-----
172:
173:      RETURN
174:
175: !#####
176:      END SUBROUTINE get_localelement3d_area_boundary
177: !#####
178:
179:
180:      ! Set the total number of nodes in a local element
181: !#####
182:      SUBROUTINE set_localelement3d_nnodes(le3d, nnodes)

```

```

183: !#####
184:
185:     TYPE(struct_localelement3d), INTENT(INOUT) :: le3d
186:
187:     INTEGER, INTENT(IN) :: nnodes
188:
189: !-----
190:
191:     le3d%nnodes = nnodes
192:
193: !-----
194:
195:     RETURN
196:
197: !#####
198:     END SUBROUTINE set_localelement3d_nnodes
199: !#####
200:
201:
202:     ! Get the total number of nodes in a local element
203: !#####
204:     SUBROUTINE get_localelement3d_nnodes(le3d, nnodes)
205: !#####
206:
207:     TYPE(struct_localelement3d), INTENT(IN) :: le3d
208:
209:     INTEGER, INTENT(OUT) :: nnodes
210:
211: !-----
212:
213:     nnodes = le3d%nnodes
214:
215: !-----
216:
217:     RETURN
218:

```

```

219: !#####
220:     END SUBROUTINE get_localelement3d_nnodes
221: !#####
222:
223:
224:     ! Get the total number of nodes on a local element boundary
225: !#####
226:     SUBROUTINE get_localelement3d_nnodes_boundary &
227:         (le3d, nnodes_boundary)
228: !#####
229:
230:     TYPE(struct_localelement3d), INTENT(IN) :: le3d
231:
232:     INTEGER, INTENT(OUT) :: nnodes_boundary
233:
234: !-----
235:
236:     nnodes_boundary = le3d%nnodes_boundary
237:
238: !-----
239:
240:     RETURN
241:
242: !#####
243:     END SUBROUTINE get_localelement3d_nnodes_boundary
244: !#####
245:
246:
247:     ! Get the total number of edges on a local element boundary
248: !#####
249:     SUBROUTINE get_localelement3d_nedges_boundary &
250:         (le3d, nedges_boundary)
251: !#####
252:
253:     TYPE(struct_localelement3d), INTENT(IN) :: le3d
254:

```



```

255:      INTEGER, INTENT (OUT) :: nedges_boundary
256:
257: !-----
258:
259:      nedges_boundary = le3d%nedges_boundary
260:
261: !-----
262:
263:      RETURN
264:
265: !#####
266:      END SUBROUTINE get_localelement3d_nedges_boundary
267: !#####
268:
269:
270:      ! Get the total number of nodes on a local element edge
271: !#####
272:      SUBROUTINE get_localelement3d_nnodes_edge(le3d, nnodes_edge)
273: !#####
274:
275:      TYPE(struct_localelement3d), INTENT (IN) :: le3d
276:
277:      INTEGER, INTENT (OUT) :: nnodes_edge
278:
279: !-----
280:
281:      nnodes_edge = le3d%nnodes_edge
282:
283: !-----
284:
285:      RETURN
286:
287: !#####
288:      END SUBROUTINE get_localelement3d_nnodes_edge
289: !#####
290:

```

```

291:
292:      ! Get Cartesian coordinates of a local node
293: !#####
294:      SUBROUTINE get_localelement3d_xi(le3d, xi)
295: !#####
296:
297:      TYPE(struct_localelement3d), INTENT(IN) :: le3d
298:
299:      REAL(8), INTENT(OUT) :: xi(:, :)
300:
301: !-----
302:
303:      xi = le3d%xi
304:
305: !-----
306:
307:      RETURN
308:
309: !#####
310:      END SUBROUTINE get_localelement3d_xi
311: !#####
312:
313:
314:      ! Get table of local element boundary no. and local node no.
315: !#####
316:      SUBROUTINE get_localelement3d_table_na(le3d, table_na)
317: !#####
318:
319:      TYPE(struct_localelement3d), INTENT(IN) :: le3d
320:
321:      INTEGER, INTENT(OUT) :: table_na(:, :)
322:
323: !-----
324:
325:      table_na = le3d%table_na
326:

```

```

327: !-----
328:
329:     RETURN
330:
331: !#####
332:     END SUBROUTINE get_localelement3d_table_na
333: !#####
334:
335:
336:     ! Initialize localelement3d
337: !#####
338:     SUBROUTINE init_localelement3d      &
339:         (le3d, nboundaries, nnodes)
340: !#####
341:
342:     TYPE(struct_localelement3d), INTENT(INOUT) :: le3d
343:
344:     INTEGER, INTENT(IN) :: nboundaries
345:     INTEGER, INTENT(IN) :: nnodes
346:
347: !-----
348:
349:     INTEGER :: nnodes_boundary
350:     INTEGER :: nnodes_edge
351:
352: !-----
353:
354:     le3d%nboundaries = nboundaries
355:
356:     !-----
357:
358:     ! Hexahedral element
359:     IF( nboundaries .EQ. 6 ) THEN
360:
361:         le3d%nedges = 12
362:

```

```

363:      le3d%nedges_boundary = 4
364:
365:      le3d%volume = 8.0D0
366:
367:      le3d%area_boundary = 4.0D0
368:
369:      END IF
370:
371:      !-----
372:
373:      le3d%nnodes = nnodes
374:
375:      !-----
376:
377:      ALLOCATE( le3d%xi(3, nnodes) )
378:
379:      ! Hexahedral element
380:      IF( nboundaries .EQ. 6 ) THEN
381:
382:      ! Linear element
383:      IF( nnodes .EQ. 8 ) THEN
384:
385:      ! xi
386:      le3d%xi(1, 1) = -1.0D0
387:      le3d%xi(1, 2) = 1.0D0
388:      le3d%xi(1, 3) = 1.0D0
389:      le3d%xi(1, 4) = -1.0D0
390:      le3d%xi(1, 5) = -1.0D0
391:      le3d%xi(1, 6) = 1.0D0
392:      le3d%xi(1, 7) = 1.0D0
393:      le3d%xi(1, 8) = -1.0D0
394:      ! eta
395:      le3d%xi(2, 1) = -1.0D0
396:      le3d%xi(2, 2) = -1.0D0
397:      le3d%xi(2, 3) = 1.0D0
398:      le3d%xi(2, 4) = 1.0D0

```

```

399:      le3d%xi (2, 5) = -1.0D0
400:      le3d%xi (2, 6) = -1.0D0
401:      le3d%xi (2, 7) =  1.0D0
402:      le3d%xi (2, 8) =  1.0D0
403:      ! zeta
404:      le3d%xi (3, 1) = -1.0D0
405:      le3d%xi (3, 2) = -1.0D0
406:      le3d%xi (3, 3) = -1.0D0
407:      le3d%xi (3, 4) = -1.0D0
408:      le3d%xi (3, 5) =  1.0D0
409:      le3d%xi (3, 6) =  1.0D0
410:      le3d%xi (3, 7) =  1.0D0
411:      le3d%xi (3, 8) =  1.0D0
412:
413:      END IF
414:
415:      END IF
416:
417:      !-----
418:
419:      ! Hexahedral element
420:      IF( nboundaries .EQ. 6 ) THEN
421:
422:      ! Linear element
423:      IF( nnodes .EQ. 8 ) THEN
424:
425:      nnodes_boundary = 4
426:      nnodes_edge = 2
427:
428:      END IF
429:
430:      END IF
431:
432:      le3d%nnodes_boundary = nnodes_boundary
433:      le3d%nnodes_edge = nnodes_edge
434:

```

```

435:      !-----
436:
437:      ALLOCATE( le3d%table_na(nnodes_boundary, nboundaries) )
438:
439:      ! Hexahedral element
440:      IF( nboundaries .EQ. 6 ) THEN
441:
442:          ! ma = 1
443:          le3d%table_na(1, 1) = 4
444:          le3d%table_na(2, 1) = 3
445:          le3d%table_na(3, 1) = 2
446:          le3d%table_na(4, 1) = 1
447:          ! ma = 2
448:          le3d%table_na(1, 2) = 5
449:          le3d%table_na(2, 2) = 6
450:          le3d%table_na(3, 2) = 7
451:          le3d%table_na(4, 2) = 8
452:          ! ma = 3
453:          le3d%table_na(1, 3) = 1
454:          le3d%table_na(2, 3) = 2
455:          le3d%table_na(3, 3) = 6
456:          le3d%table_na(4, 3) = 5
457:          ! ma = 4
458:          le3d%table_na(1, 4) = 2
459:          le3d%table_na(2, 4) = 3
460:          le3d%table_na(3, 4) = 7
461:          le3d%table_na(4, 4) = 6
462:          ! ma = 5
463:          le3d%table_na(1, 5) = 3
464:          le3d%table_na(2, 5) = 4
465:          le3d%table_na(3, 5) = 8
466:          le3d%table_na(4, 5) = 7
467:          ! ma = 6
468:          le3d%table_na(1, 6) = 4
469:          le3d%table_na(2, 6) = 1
470:          le3d%table_na(3, 6) = 5

```

```

471:      le3d%table_na(4, 6) = 8
472:
473:      END IF
474:
475: !-----
476:
477:      RETURN
478:
479: !#####
480:      END SUBROUTINE init_localelement3d
481: !#####
482:
483:
484:      ! Delete localelement3d
485: !#####
486:      SUBROUTINE del_localelement3d(le3d)
487: !#####
488:
489:      TYPE(struct_localelement3d), INTENT(INOUT) :: le3d
490:
491: !-----
492:
493:      IF( le3d%nboundaries .EQ. 0 ) THEN
494:
495:          RETURN
496:
497:      END IF
498:
499: !-----
500:
501:      le3d%nboundaries = 0
502:      le3d%nedges = 0
503:
504:      !-----
505:
506:      le3d%volume = 0.0D0

```

```

507:      le3d%area_boundary = 0.0D0
508:
509:      !-----
510:
511:      le3d%nnodes = 0
512:      le3d%nnodes_boundary = 0
513:      le3d%nedges_boundary = 0
514:      le3d%nnodes_edge = 0
515:
516:      DEALLOCATE( le3d%xi )
517:
518:      !-----
519:
520:      DEALLOCATE( le3d%table_na )
521:
522: !-----
523:
524:      RETURN
525:
526: !#####
527:      END SUBROUTINE del_localelement3d
528: !#####
529:
530:
531: !#####
532:      END MODULE mod_localelement3d

```

3. 有限要素 (物理空間) モジュール mod_elements3d.f90

```

1:      MODULE mod_elements3d
2: !#####
3:
4:      USE mod_nodes3d
5:      USE mod_localelement3d
6:
7: !-----

```



```

8:
9:     IMPLICIT NONE
10:
11: !-----
12:
13:     TYPE :: struct_elements3d
14:
15:     !-----
16:
17:     PRIVATE
18:
19:     !-----
20:
21:     TYPE(struct_nodes3d), POINTER      :: ns3d => NULL()
22:     TYPE(struct_localelement3d), POINTER :: le3d => NULL()
23:
24:     !-----
25:     !
26:     ! n
27:     ! The total number of elements
28:     !
29:     ! connectivity(:, :)
30:     ! Connectivity between element no. and node no.
31:     !
32:     ! volume(:)
33:     ! Element volume
34:     !
35:     ! max_volume, max_ie_volume
36:     ! Maximum element volume and the element no.
37:     !
38:     ! min_volume_min, min_ie_volume
39:     ! Minimum element volume and the element no.
40:     !
41:     ! sum_volume
42:     ! The sum of element volumes
43:     !

```

```

44:      !-----
45:
46:      INTEGER :: n
47:      INTEGER, ALLOCATABLE :: connectivity(:, :)
48:      INTEGER :: ie_max_volume, ie_min_volume
49:
50:      REAL(8), ALLOCATABLE :: volume(:)
51:      REAL(8) :: max_volume, min_volume
52:      REAL(8) :: sum_volume
53:
54:      !-----
55:
56:      END TYPE struct_elements3d
57:
58: !-----
59:
60:      CONTAINS
61:
62:
63:      ! Set the total number of elements
64: !#####
65:      SUBROUTINE set_elements3d_n(es3d, n)
66: !#####
67:
68:      TYPE(struct_elements3d), INTENT(INOUT) :: es3d
69:
70:      INTEGER, INTENT(IN) :: n
71:
72: !-----
73:
74:      es3d%n = n
75:
76: !-----
77:
78:      RETURN
79:

```

```

80: !#####
81:     END SUBROUTINE set_elements3d_n
82: !#####
83:
84:
85:     ! Get the total number of elements
86: !#####
87:     SUBROUTINE get_elements3d_n(es3d, n)
88: !#####
89:
90:     TYPE(struct_elements3d), INTENT(IN) :: es3d
91:
92:     INTEGER, INTENT(OUT) :: n
93:
94: !-----
95:
96:     n = es3d%n
97:
98: !-----
99:
100:    RETURN
101:
102: !#####
103:    END SUBROUTINE get_elements3d_n
104: !#####
105:
106:
107:    ! Set connectivity between element no. and node no.
108: !#####
109:    SUBROUTINE set_elements3d_connectivity(es3d, connectivity)
110: !#####
111:
112:    TYPE(struct_elements3d), INTENT(INOUT) :: es3d
113:
114:    INTEGER, INTENT(IN) :: connectivity(:, :)
115:

```

```

116: !-----
117:
118:     es3d%connectivity = connectivity
119:
120: !-----
121:
122:     RETURN
123:
124: !#####
125:     END SUBROUTINE set_elements3d_connectivity
126: !#####
127:
128:
129:     ! Get connectivity between element no. and node no.
130: !#####
131:     SUBROUTINE get_elements3d_connectivity(es3d, connectivity)
132: !#####
133:
134:     TYPE(struct_elements3d), INTENT(IN) :: es3d
135:
136:     INTEGER, INTENT(OUT) :: connectivity(:, :)
137:
138: !-----
139:
140:     connectivity = es3d%connectivity
141:
142: !-----
143:
144:     RETURN
145:
146: !#####
147:     END SUBROUTINE get_elements3d_connectivity
148: !#####
149:
150:
151:     ! Get element volume

```

```

152: !#####
153:     SUBROUTINE get_elements3d_volume      &
154:         (es3d, volume,                    &
155:         max_volume, ie_max_volume, &
156:         min_volume, ie_min_volume, &
157:         sum_volume)
158: !#####
159:
160:     TYPE(struct_elements3d), INTENT(IN) :: es3d
161:
162:     REAL(8), INTENT(OUT) :: volume(:)
163:     REAL(8), INTENT(OUT) :: max_volume
164:     INTEGER, INTENT(OUT) :: ie_max_volume
165:     REAL(8), INTENT(OUT) :: min_volume
166:     INTEGER, INTENT(OUT) :: ie_min_volume
167:     REAL(8), INTENT(OUT) :: sum_volume
168:
169: !-----
170:
171:     volume = es3d%volume
172:
173:     max_volume = es3d%max_volume
174:     ie_max_volume = es3d%ie_max_volume
175:
176:     min_volume = es3d%min_volume
177:     ie_min_volume = es3d%ie_min_volume
178:
179:     sum_volume = es3d%sum_volume
180:
181: !-----
182:
183:     RETURN
184:
185: !#####
186:     END SUBROUTINE get_elements3d_volume
187: !#####

```

```

188:
189:
190:     ! Initialize elements3d
191: !#####
192:     SUBROUTINE init_elements3d(es3d, ns3d, le3d, n)
193: !#####
194:
195:     TYPE(struct_elements3d), INTENT(INOUT) :: es3d
196:
197:     TYPE(struct_nodes3d), TARGET, INTENT(IN)      :: ns3d
198:     TYPE(struct_localelement3d), TARGET, INTENT(IN) :: le3d
199:
200:     INTEGER, INTENT(IN) :: n
201:
202: !-----
203:
204:     INTEGER :: le3d_nnodes
205:
206: !-----
207:
208:     es3d%ns3d => ns3d
209:     es3d%le3d => le3d
210:
211: !-----
212:
213:     CALL get_localelement3d_nnodes(es3d%le3d, le3d_nnodes)
214:
215: !-----
216:
217:     es3d%n = n
218:
219:     !-----
220:
221:     ALLOCATE( es3d%connectivity(le3d_nnodes, n) )
222:
223:     es3d%connectivity = 0

```

```

224:
225:      !-----
226:
227:      ALLOCATE( es3d%volume(n) )
228:
229:      es3d%volume = 0.0D0
230:
231:      !-----
232:
233:      es3d%max_volume = 0.0D0
234:      es3d%ie_max_volume = 0
235:
236:      es3d%min_volume = 0.0D0
237:      es3d%ie_min_volume = 0
238:
239:      !-----
240:
241:      es3d%sum_volume = 0.0D0
242:
243: !-----
244:
245:      RETURN
246:
247: !#####
248:      END SUBROUTINE init_elements3d
249: !#####
250:
251:
252:      ! Calculate elements3d
253: !#####
254:      SUBROUTINE cal_elements3d(es3d)
255: !#####
256:
257:      TYPE(struct_elements3d), INTENT(INOUT) :: es3d
258:
259: !-----

```

```

260:
261:     INTEGER :: ie
262:
263: !-----
264:
265:     es3d%max_volume = MAXVAL( es3d%volume )
266:
267:     es3d%ie_max_volume = 0
268:
269:     DO ie = 1, es3d%n
270:
271:         IF( es3d%volume(ie) .EQ. es3d%max_volume ) THEN
272:
273:             es3d%ie_max_volume = ie
274:
275:         END IF
276:
277:     END DO
278:
279: !-----
280:
281:     es3d%min_volume = MINVAL( es3d%volume )
282:
283:     es3d%ie_min_volume = 0
284:
285:     DO ie = 1, es3d%n
286:
287:         IF( es3d%volume(ie) .EQ. es3d%min_volume ) THEN
288:
289:             es3d%ie_min_volume = ie
290:
291:         END IF
292:
293:     END DO
294:
295: !-----

```



```

296:
297:     es3d%sum_volume = SUM( es3d%volume )
298:
299: !-----
300:
301:     RETURN
302:
303: !#####
304:     END SUBROUTINE cal_elements3d
305: !#####
306:
307:
308:     ! Delete elements3d
309: !#####
310:     SUBROUTINE del_elements3d(es3d)
311: !#####
312:
313:     TYPE(struct_elements3d), INTENT(INOUT) :: es3d
314:
315: !-----
316:
317:     IF( es3d%n .EQ. 0 ) THEN
318:
319:         RETURN
320:
321:     END IF
322:
323: !-----
324:
325:     NULLIFY( es3d%ns3d )
326:     NULLIFY( es3d%le3d )
327:
328: !-----
329:
330:     es3d%n = 0
331:

```

```

332:      !-----
333:
334:      DEALLOCATE( es3d%connectivity )
335:
336:      !-----
337:
338:      DEALLOCATE( es3d%volume )
339:
340:      !-----
341:
342:      es3d%max_volume = 0.0D0
343:      es3d%ie_max_volume = 0
344:
345:      es3d%min_volume = 0.0D0
346:      es3d%ie_min_volume = 0
347:
348:      !-----
349:
350:      es3d%sum_volume = 0.0D0
351:
352:      !-----
353:
354:      RETURN
355:
356:      !#####
357:      END SUBROUTINE del_elements3d
358:      !#####
359:
360:
361:      !#####
362:      END MODULE mod_elements3d

```

4. アプリケーションモジュール mod_appli.f90

```

1:      MODULE mod_appli
2:      !#####

```

```

3:
4:     USE mod_nodes3d
5:     USE mod_localelement3d
6:     USE mod_elements3d
7:
8: !-----
9:
10:    IMPLICIT NONE
11:
12: !-----
13:
14:    TYPE(struct_nodes3d), POINTER      :: ns3d
15:    TYPE(struct_localelement3d), POINTER :: le3d
16:    TYPE(struct_elements3d), POINTER   :: es3d
17:
18: !-----
19:
20:    CONTAINS
21:
22:
23:    ! Start appli
24: !#####
25:    SUBROUTINE start_appli()
26: !#####
27:
28:    INTEGER :: ns3d_n
29:    INTEGER :: le3d_nboundaries
30:    INTEGER :: le3d_nnodes
31:    INTEGER :: es3d_n
32:    INTEGER, ALLOCATABLE :: es3d_connectivity(:, :)
33:
34:    REAL(8), ALLOCATABLE :: ns3d_x(:, :)
35:
36:    CHARACTER(1) :: dataname
37:
38: !-----

```

```

39:
40:     ALLOCATE( ns3d )
41:     ALLOCATE( le3d )
42:     ALLOCATE( es3d )
43:
44: !-----
45:
46:     ns3d_n = 8
47:     le3d_nboundaries = 6
48:     le3d_nnodes = 8
49:     es3d_n = 1
50:
51: !-----
52:
53:     CALL init_nodes3d(ns3d, ns3d_n)
54:
55:     CALL init_localelement3d          &
56:         (le3d, le3d_nboundaries, le3d_nnodes)
57:
58:     CALL init_elements3d(es3d, ns3d, le3d, es3d_n)
59:
60: !-----
61:
62:     ALLOCATE( ns3d_x(3, ns3d_n) )
63:
64:     ! Node 1
65:     ns3d_x(1, 1) = 0.0D0
66:     ns3d_x(2, 1) = 0.0D0
67:     ns3d_x(3, 1) = 0.0D0
68:     ! Node 2
69:     ns3d_x(1, 2) = 1.0D0
70:     ns3d_x(2, 2) = 0.0D0
71:     ns3d_x(3, 2) = 0.0D0
72:     ! Node 3
73:     ns3d_x(1, 3) = 0.0D0
74:     ns3d_x(2, 3) = 1.0D0

```

```

75:      ns3d_x(3, 3) = 0.0D0
76:      ! Node 4
77:      ns3d_x(1, 4) = 1.0D0
78:      ns3d_x(2, 4) = 1.0D0
79:      ns3d_x(3, 4) = 0.0D0
80:      ! Node 5
81:      ns3d_x(1, 5) = 0.0D0
82:      ns3d_x(2, 5) = 0.0D0
83:      ns3d_x(3, 5) = 1.0D0
84:      ! Node 6
85:      ns3d_x(1, 6) = 1.0D0
86:      ns3d_x(2, 6) = 0.0D0
87:      ns3d_x(3, 6) = 1.0D0
88:      ! Node 7
89:      ns3d_x(1, 7) = 0.0D0
90:      ns3d_x(2, 7) = 1.0D0
91:      ns3d_x(3, 7) = 1.0D0
92:      ! Node 8
93:      ns3d_x(1, 8) = 1.0D0
94:      ns3d_x(2, 8) = 1.0D0
95:      ns3d_x(3, 8) = 1.0D0
96:
97:      CALL set_nodes3d_x(ns3d, ns3d_x)
98:
99:      ALLOCATE( es3d_connectivity(1e3d_nnodes, es3d_n) )
100:
101:      ! Element 1
102:      es3d_connectivity(1, 1) = 1
103:      es3d_connectivity(2, 1) = 2
104:      es3d_connectivity(3, 1) = 4
105:      es3d_connectivity(4, 1) = 3
106:      es3d_connectivity(5, 1) = 5
107:      es3d_connectivity(6, 1) = 6
108:      es3d_connectivity(7, 1) = 8
109:      es3d_connectivity(8, 1) = 7
110:

```

```

111:      CALL set_elements3d_connectivity(es3d, es3d_connectivity)
112:
113: !-----
114:
115:      DEALLOCATE( ns3d_x )
116:      DEALLOCATE( es3d_connectivity )
117:
118: !-----
119:
120:      RETURN
121:
122: !#####
123:      END SUBROUTINE start_appli
124: !#####
125:
126:
127:      ! Run appli
128: !#####
129:      SUBROUTINE run_appli()
130: !#####
131:
132:      INTEGER :: ns3d_n
133:      INTEGER, ALLOCATABLE :: ns3d_bc(:)
134:      INTEGER :: le3d_nboundaries
135:      INTEGER :: le3d_nnodes
136:      INTEGER :: es3d_n
137:      INTEGER, ALLOCATABLE :: es3d_connectivity(:, :)
138:      INTEGER :: es3d_ie_max_volume
139:      INTEGER :: es3d_ie_min_volume
140:      INTEGER :: i
141:      INTEGER :: id
142:      INTEGER :: na
143:      INTEGER :: ie
144:
145:      REAL(8), ALLOCATABLE :: ns3d_x(:, :)
146:      REAL(8), ALLOCATABLE :: ns3d_u(:)

```

```

147:      REAL (8), ALLOCATABLE :: es3d_volume(:)
148:      REAL (8) :: es3d_max_volume
149:      REAL (8) :: es3d_min_volume
150:      REAL (8) :: es3d_sum_volume
151:
152: !-----
153:
154:      CALL get_nodes3d_n(ns3d, ns3d_n)
155:
156:      ALLOCATE( ns3d_x(3, ns3d_n) )
157:      CALL get_nodes3d_x(ns3d, ns3d_x)
158:      ALLOCATE( ns3d_u(3*ns3d_n) )
159:      ns3d_u = 0.0D0
160:      ALLOCATE( ns3d_bc(3*ns3d_n) )
161:      ns3d_bc = 0
162:
163:      CALL get_localelement3d_nboundaries(le3d, le3d_nboundaries)
164:      CALL get_localelement3d_nnodes(le3d, le3d_nnodes)
165:
166:      CALL get_elements3d_n(es3d, es3d_n)
167:      ALLOCATE( es3d_volume(es3d_n) )
168:      CALL get_elements3d_volume          &
169:          (es3d, es3d_volume,            &
170:           es3d_max_volume, es3d_ie_max_volume, &
171:           es3d_min_volume, es3d_ie_min_volume, &
172:           es3d_sum_volume)
173:      ALLOCATE( es3d_connectivity(le3d_nnodes, es3d_n) )
174:      CALL get_elements3d_connectivity(es3d, es3d_connectivity)
175:
176: !-----
177:
178:      OPEN(14, FILE = 'mesh.inp')
179:
180:      WRITE(14, '( 5(I8, 1X) )') ns3d_n, es3d_n, 3, 13, 0
181:
182:      DO id = 1, ns3d_n

```

```

183:
184:     WRITE( 14, ' ( (I8, 1X), 3(E17.8, 1X) )' ) &
185:         id, ( ns3d_x(i, id), i = 1, 3 )
186:
187:     END DO
188:
189:     DO ie = 1, es3d_n
190:
191:         WRITE( 14, ' ( 2(I8, 1X), (A5, 1X), 27(I8, 1X) )' ) &
192:             ie, 1, ' hex', &
193:             ( es3d_connectivity(na, ie), na = 1, le3d_nnodes )
194:
195:     END DO
196:
197:     WRITE(14, ' ( 4(I8, 1X) )' ) 1, 3
198:     WRITE(14, ' (A)' ) 'DISPLACEMENT, m'
199:
200:     DO id = 1, ns3d_n
201:
202:         WRITE( 14, ' ( (I8, 1X), 3(E17.8, 1X) )' ) &
203:             id, ( ns3d_u( 3*(id-1)+i ), i = 1, 3 )
204:
205:     END DO
206:
207:     WRITE(14, ' ( 14I8 )' ) 1, 1
208:     WRITE(14, ' ( (A, 1X) )' ) 'VOLUME, m3'
209:
210:     DO ie = 1, es3d_n
211:
212:         WRITE( 14, ' ( (I8, 1X), (E17.8, 1X) )' ) &
213:             ie, es3d_volume(ie)
214:
215:     END DO
216:
217:     CLOSE(14)
218:

```



```

219: !-----
220:
221:     DEALLOCATE( ns3d_x )
222:     DEALLOCATE( ns3d_u )
223:
224:     DEALLOCATE( es3d_volume )
225:     DEALLOCATE( es3d_connectivity )
226:
227: !-----
228:
229:     RETURN
230:
231: !#####
232:     END SUBROUTINE run_appli
233: !#####
234:
235:
236: !#####
237:     SUBROUTINE finish_appli()
238: !#####
239:
240:     CALL del_nodes3d(ns3d)
241:     CALL del_localelement3d(le3d)
242:     CALL del_elements3d(es3d)
243:
244: !-----
245:
246:     DEALLOCATE( ns3d )
247:     DEALLOCATE( le3d )
248:     DEALLOCATE( es3d )
249:
250: !-----
251:
252:     RETURN
253:
254: !#####

```

```

255:      END SUBROUTINE finish_appli
256: !#####
257:
258:
259: !#####
260:      END MODULE mod_appli

```

5. メインプログラム main_appli.f90

```

1:      PROGRAM main_appli
2: !#####
3:
4:      USE mod_appli
5:
6: !-----
7:
8:      IMPLICIT NONE
9:
10: !-----
11:
12:      ! Start appli
13:      CALL start_appli()
14:
15: !-----
16:
17:      ! Run appli
18:      CALL run_appli()
19:
20: !-----
21:
22:      ! Finish appli
23:      CALL finish_appli()
24:
25: !-----
26:
27:      STOP

```

```

28:
29: !#####
30:     END PROGRAM main_appli

```

第 3 節のプログラム：モジュール mod_rectmesher3d の作成

6. 直方体メッシャーモジュール mod_rectmesher.f90

```

1:     MODULE mod_rectmesher3d
2: !#####
3:
4:     USE mod_nodes3d
5:     USE mod_localelement3d
6:     USE mod_elements3d
7:
8: !-----
9:
10:    IMPLICIT NONE
11:
12: !-----
13:
14:    TYPE :: struct_rectmesher3d
15:
16:    !-----
17:
18:    PRIVATE
19:
20:    !-----
21:
22:    TYPE(struct_nodes3d), POINTER      :: ns3d => NULL()
23:    TYPE(struct_localelement3d), POINTER :: le3d => NULL()
24:    TYPE(struct_elements3d), POINTER    :: es3d => NULL()
25:
26:    !-----
27:    !

```

```

28:      ! n_x(3)
29:      ! The total number of divisions
30:      ! n_x, n_y, n_z
31:      !
32:      ! x_start(3)
33:      ! Cartesian coordinates of the start point
34:      ! (x_start, y_start, z_start)
35:      !
36:      ! x_end(3)
37:      ! Cartesian coordinates of the end point
38:      ! (x_end, y_end, z_end)
39:      !
40:      ! x_center(3)
41:      ! Cartesian coordinates of the center point
42:      ! (x_center, y_center, z_center)
43:      !
44:      ! length_x(3)
45:      ! Distance between the start and end points
46:      !
47:      !-----
48:
49:      INTEGER :: n_x(3)
50:
51:      REAL(8) :: x_start(3)
52:      REAL(8) :: x_end(3)
53:      REAL(8) :: x_center(3)
54:      REAL(8) :: length_x(3)
55:
56:      !-----
57:
58:      END TYPE struct_rectmesher3d
59:
60: !-----
61:
62:      PRIVATE :: cal_rectmesher3d_connectivity_hex
63:      PRIVATE :: cal_rectmesher3d_coordinates_hex

```

```

64:
65: !-----
66:
67:     CONTAINS
68:
69:
70:     ! Get the total number of divisions
71: !#####
72:     SUBROUTINE get_rectmesher3d_n(rm3d, n)
73: !#####
74:
75:     TYPE(struct_rectmesher3d), INTENT(INOUT) :: rm3d
76:
77:     INTEGER, INTENT(OUT) :: n(3)
78:
79: !-----
80:
81:     n(1) = rm3d%n_x(1)
82:     n(2) = rm3d%n_x(2)
83:     n(3) = rm3d%n_x(3)
84:
85: !-----
86:
87:     RETURN
88:
89: !#####
90:     END SUBROUTINE get_rectmesher3d_n
91: !#####
92:
93:
94:     ! Get Cartesian coordinates of the start and end points
95: !#####
96:     SUBROUTINE get_rectmesher3d_x_start_x_end &
97:         (rm3d, x_start, x_end, x_center, &
98:         length_x)
99: !#####

```

```

100:
101:     TYPE(struct_rectmesher3d), INTENT(IN) :: rm3d
102:
103:     REAL(8), INTENT(OUT) :: x_start(3)
104:     REAL(8), INTENT(OUT) :: x_end(3)
105:     REAL(8), INTENT(OUT) :: x_center(3)
106:     REAL(8), INTENT(OUT) :: length_x(3)
107:
108: !-----
109:
110:     x_start(1) = rm3d%x_start(1)
111:     x_start(2) = rm3d%x_start(2)
112:     x_start(3) = rm3d%x_start(3)
113:
114:     x_end(1) = rm3d%x_end(1)
115:     x_end(2) = rm3d%x_end(2)
116:     x_end(3) = rm3d%x_end(3)
117:
118:     x_center(1) = rm3d%x_center(1)
119:     x_center(2) = rm3d%x_center(2)
120:     x_center(3) = rm3d%x_center(3)
121:
122:     length_x(1) = rm3d%length_x(1)
123:     length_x(2) = rm3d%length_x(2)
124:     length_x(3) = rm3d%length_x(3)
125:
126: !-----
127:
128:     RETURN
129:
130: !#####
131:     END SUBROUTINE get_rectmesher3d_x_start_x_end
132: !#####
133:
134:
135:     ! Initialize rectmesher3d

```

```

136: !#####
137:     SUBROUTINE init_rectmesher3d      &
138:         (rm3d, ns3d, le3d, es3d, &
139:         n_x, x_start, x_end)
140: !#####
141:
142:     TYPE(struct_rectmesher3d), INTENT(INOUT) :: rm3d
143:
144:     TYPE(struct_nodes3d), TARGET, INTENT(INOUT) :: ns3d
145:     TYPE(struct_localelement3d), TARGET, INTENT(INOUT) :: le3d
146:     TYPE(struct_elements3d), TARGET, INTENT(INOUT) :: es3d
147:
148:     INTEGER, INTENT(IN) :: n_x(3)
149:     REAL(8), INTENT(IN) :: x_start(3)
150:     REAL(8), INTENT(IN) :: x_end(3)
151:
152: !-----
153:
154:     INTEGER :: ns3d_n
155:     INTEGER :: le3d_nboundaries
156:     INTEGER :: le3d_nnodes
157:     INTEGER :: es3d_n
158:
159: !-----
160:
161:     rm3d%ns3d => ns3d
162:     rm3d%le3d => le3d
163:     rm3d%es3d => es3d
164:
165: !-----
166:
167:     rm3d%n_x(1) = n_x(1)
168:     rm3d%n_x(2) = n_x(2)
169:     rm3d%n_x(3) = n_x(3)
170:
171:     rm3d%x_start(1) = x_start(1)

```

```

172:      rm3d%x_start(2) = x_start(2)
173:      rm3d%x_start(3) = x_start(3)
174:
175:      rm3d%x_end(1) = x_end(1)
176:      rm3d%x_end(2) = x_end(2)
177:      rm3d%x_end(3) = x_end(3)
178:
179:      rm3d%x_center(1) = 0.5D0*( x_start(1)+x_end(1) )
180:      rm3d%x_center(2) = 0.5D0*( x_start(2)+x_end(2) )
181:      rm3d%x_center(3) = 0.5D0*( x_start(3)+x_end(3) )
182:
183:      rm3d%length_x(1) = x_end(1)-x_start(1)
184:      rm3d%length_x(2) = x_end(2)-x_start(2)
185:      rm3d%length_x(3) = x_end(3)-x_start(3)
186:
187: !-----
188:
189:      ns3d_n = ( rm3d%n_x(1)+1 )*( rm3d%n_x(2)+1 )*( rm3d%n_x(3)+1 )
190:
191:      le3d_nboundaries = 6
192:      le3d_nnodes      = 8
193:
194:      es3d_n = rm3d%n_x(1)*rm3d%n_x(2)*rm3d%n_x(3)
195:
196: !-----
197:
198:      CALL set_nodes3d_n(ns3d, ns3d_n)
199:
200:      CALL set_elements3d_n(es3d, es3d_n)
201:
202:      CALL set_localelement3d_nboundaries(le3d, le3d_nboundaries)
203:      CALL set_localelement3d_nnodes(le3d, le3d_nnodes)
204:
205: !-----
206:
207:      WRITE( 6, ' ( (A, I8) )' )          &

```



```

208:      'The total number of nodes: ', ns3d_n
209:      WRITE( 6, ' ( (A, I8) )' )      &
210:      'The total number of local element boundaries: ', &
211:      le3d_nboundaries
212:      WRITE( 6, ' ( (A, I8) )' )      &
213:      'The total number of local element nodes: ', &
214:      le3d_nnodes
215:      WRITE( 6, ' ( (A, I8) )' )      &
216:      'The total number of elements: ', es3d_n
217:      WRITE(6, *)
218:
219: !-----
220:
221:      RETURN
222:
223: !#####
224:      END SUBROUTINE init_rectmesher3d
225: !#####
226:
227:
228:      ! Calculate rectmesher3d
229: !#####
230:      SUBROUTINE cal_rectmesher3d(rm3d)
231: !#####
232:
233:      TYPE(struct_rectmesher3d), INTENT(INOUT) :: rm3d
234:
235: !-----
236:
237:      CALL cal_rectmesher3d_connectivity_hex(rm3d)
238:
239:      CALL cal_rectmesher3d_coordinates_hex(rm3d)
240:
241: !-----
242:
243:      RETURN

```

```

244:
245: !#####
246:     END SUBROUTINE cal_rectmesher3d
247: !#####
248:
249:
250:     ! Calculate rectmesher3d (connectivity, hexahedral elements)
251: !#####
252:     SUBROUTINE cal_rectmesher3d_connectivity_hex (rm3d)
253: !#####
254:
255:     TYPE (struct_rectmesher3d), INTENT (INOUT) :: rm3d
256:
257: !-----
258:
259:     INTEGER :: le3d_nnodes
260:     INTEGER :: es3d_n
261:     INTEGER, ALLOCATABLE :: es3d_connectivity(:, :)
262:     INTEGER :: n_x_1(3)
263:     INTEGER :: i, j, k
264:     INTEGER :: ie
265:
266: !-----
267:
268:     CALL get_localelement3d_nnodes (rm3d%le3d, le3d_nnodes)
269:
270:     CALL get_elements3d_n (rm3d%es3d, es3d_n)
271:     ALLOCATE ( es3d_connectivity (le3d_nnodes, es3d_n) )
272:
273: !-----
274:
275:     n_x_1 (1) = rm3d%n_x (1)+1
276:     n_x_1 (2) = rm3d%n_x (2)+1
277:     n_x_1 (3) = rm3d%n_x (3)+1
278:
279: !-----

```

```

280:
281:     ie = 0
282:
283:     !-----
284:
285:     DO k = 1, rm3d%n_x(3)
286:
287:         DO j = 1, rm3d%n_x(2)
288:
289:             DO i = 1, rm3d%n_x(1)
290:
291:                 !-----
292:
293:                 ! Element no.
294:                 ie = ie+1
295:
296:                 IF( le3d_nnodes .EQ. 8 ) THEN
297:
298:                     es3d_connectivity(1, ie)          &
299:                     = n_x_1(1)*n_x_1(2)*( k-1 )+n_x_1(1)*( j-1 )+i
300:                     es3d_connectivity(2, ie) = es3d_connectivity(1, ie)+1
301:                     es3d_connectivity(4, ie) = es3d_connectivity(1, ie)+n_x_1(1)
302:                     es3d_connectivity(3, ie) = es3d_connectivity(4, ie)+1
303:                     es3d_connectivity(5, ie) = es3d_connectivity(1, ie)+n_x_1(1)*n_x_1(2)
304:                     es3d_connectivity(6, ie) = es3d_connectivity(5, ie)+1
305:                     es3d_connectivity(8, ie) = es3d_connectivity(5, ie)+n_x_1(1)
306:                     es3d_connectivity(7, ie) = es3d_connectivity(8, ie)+1
307:
308:                 END IF
309:
310:                 !-----
311:
312:             END DO
313:
314:         END DO
315:

```

```

316:      END DO
317:
318:      !-----
319:
320:      CALL set_elements3d_connectivity(rm3d%es3d, es3d_connectivity)
321:
322: !-----
323:
324:      DEALLOCATE( es3d_connectivity )
325:
326: !-----
327:
328:      RETURN
329:
330: !#####
331:      END SUBROUTINE cal_rectmesher3d_connectivity_hex
332: !#####
333:
334:
335:      ! Calculate rectmesher3d (coordinates, hexahedral elements)
336: !#####
337:      SUBROUTINE cal_rectmesher3d_coordinates_hex(rm3d)
338: !#####
339:
340:      TYPE(struct_rectmesher3d), INTENT(INOUT) :: rm3d
341:
342: !-----
343:
344:      INTEGER :: ns3d_n
345:      INTEGER :: le3d_nnodes
346:      INTEGER :: es3d_n
347:      INTEGER, ALLOCATABLE :: es3d_connectivity(:, :)
348:      INTEGER :: i, j, k
349:      INTEGER :: id
350:      INTEGER :: na
351:      INTEGER :: ie

```

```

352:
353:     REAL (8), ALLOCATABLE :: ns3d_x(:, :)
354:     REAL (8) :: x_vertex(3, 8)
355:     REAL (8), ALLOCATABLE :: x_local(:, :)
356:     REAL (8) :: delta_x(3)
357:
358: !-----
359:
360:     CALL get_nodes3d_n(rm3d%ns3d, ns3d_n)
361:     ALLOCATE( ns3d_x(3, ns3d_n) )
362:
363:     CALL get_localelement3d_nnodes(rm3d%le3d, le3d_nnodes)
364:
365:     CALL get_elements3d_n(rm3d%es3d, es3d_n)
366:     ALLOCATE( es3d_connectivity(le3d_nnodes, es3d_n) )
367:     CALL get_elements3d_connectivity(rm3d%es3d, es3d_connectivity)
368:
369:     ALLOCATE( x_local(3, le3d_nnodes) )
370:
371: !-----
372:
373:     delta_x(1) = ( rm3d%x_end(1)-rm3d%x_start(1) )/DFLOAT( rm3d%n_x(1) )
374:     delta_x(2) = ( rm3d%x_end(2)-rm3d%x_start(2) )/DFLOAT( rm3d%n_x(2) )
375:     delta_x(3) = ( rm3d%x_end(3)-rm3d%x_start(3) )/DFLOAT( rm3d%n_x(3) )
376:
377: !-----
378:
379:     ie = 0
380:
381: !-----
382:
383:     ! Coordinates of Vertex 1
384:     x_vertex(1, 1) = rm3d%x_start(1)
385:     x_vertex(2, 1) = rm3d%x_start(2)
386:     x_vertex(3, 1) = rm3d%x_start(3)
387:

```

```

388:      !-----
389:
390:      DO k = 1, rm3d%n_x(3)
391:
392:      !-----
393:
394:      DO j = 1, rm3d%n_x(2)
395:
396:      !-----
397:
398:      DO i = 1, rm3d%n_x(1)
399:
400:      !-----
401:
402:      ! Coordinates of Vertex 2
403:      x_vertex(1, 2) = x_vertex(1, 1)+delta_x(1)
404:      x_vertex(2, 2) = x_vertex(2, 1)
405:      x_vertex(3, 2) = x_vertex(3, 1)
406:
407:      ! Coordinates of Vertex 3
408:      x_vertex(1, 3) = x_vertex(1, 2)
409:      x_vertex(2, 3) = x_vertex(2, 2)+delta_x(2)
410:      x_vertex(3, 3) = x_vertex(3, 2)
411:
412:      ! Coordinates of Vertex 4
413:      x_vertex(1, 4) = x_vertex(1, 1)
414:      x_vertex(2, 4) = x_vertex(2, 1)+delta_x(2)
415:      x_vertex(3, 4) = x_vertex(3, 1)
416:
417:      ! Coordinates of Vertex 5
418:      x_vertex(1, 5) = x_vertex(1, 1)
419:      x_vertex(2, 5) = x_vertex(2, 1)
420:      x_vertex(3, 5) = x_vertex(3, 1)+delta_x(3)
421:
422:      ! Coordinates of Vertex 6
423:      x_vertex(1, 6) = x_vertex(1, 5)+delta_x(1)

```

```

424:      x_vertex(2, 6) = x_vertex(2, 5)
425:      x_vertex(3, 6) = x_vertex(3, 5)
426:
427:      ! Coordinates of Vertex 7
428:      x_vertex(1, 7) = x_vertex(1, 6)
429:      x_vertex(2, 7) = x_vertex(2, 6)+delta_x(2)
430:      x_vertex(3, 7) = x_vertex(3, 6)
431:
432:      ! Coordinates of Vertex 8
433:      x_vertex(1, 8) = x_vertex(1, 5)
434:      x_vertex(2, 8) = x_vertex(2, 5)+delta_x(2)
435:      x_vertex(3, 8) = x_vertex(3, 5)
436:
437:      !-----
438:
439:      IF( i .EQ. 1 ) THEN
440:
441:          ! x-coordinte of Vertex 1
442:          x_vertex(1, 1) = rm3d%x_start(1)
443:          ! x-coordinte of Vertex 4
444:          x_vertex(1, 4) = rm3d%x_start(1)
445:          ! x-coordinte of Vertex 5
446:          x_vertex(1, 5) = rm3d%x_start(1)
447:          ! x-coordinte of Vertex 8
448:          x_vertex(1, 8) = rm3d%x_start(1)
449:
450:      ELSE IF( i .EQ. rm3d%n_x(1) ) THEN
451:
452:          ! x-coordinte of Vertex 2
453:          x_vertex(1, 2) = rm3d%x_end(1)
454:          ! x-coordinte of Vertex 3
455:          x_vertex(1, 3) = rm3d%x_end(1)
456:          ! x-coordinte of Vertex 6
457:          x_vertex(1, 6) = rm3d%x_end(1)
458:          ! x-coordinte of Vertex 7
459:          x_vertex(1, 7) = rm3d%x_end(1)

```

```

460:
461:     END IF
462:
463:     IF ( j .EQ. 1 ) THEN
464:
465:         ! y-coordinte of Vertex 1
466:         x_vertex(2, 1) = rm3d%x_start(2)
467:         ! y-coordinte of Vertex 2
468:         x_vertex(2, 2) = rm3d%x_start(2)
469:         ! y-coordinte of Vertex 5
470:         x_vertex(2, 5) = rm3d%x_start(2)
471:         ! y-coordinte of Vertex 6
472:         x_vertex(2, 6) = rm3d%x_start(2)
473:
474:     ELSE IF ( j .EQ. rm3d%n_x(2) ) THEN
475:
476:         ! y-coordinte of Vertex 3
477:         x_vertex(2, 3) = rm3d%x_end(2)
478:         ! y-coordinte of Vertex 4
479:         x_vertex(2, 4) = rm3d%x_end(2)
480:         ! y-coordinte of Vertex 7
481:         x_vertex(2, 7) = rm3d%x_end(2)
482:         ! y-coordinte of Vertex 8
483:         x_vertex(2, 8) = rm3d%x_end(2)
484:
485:     END IF
486:
487:     IF ( k .EQ. 1 ) THEN
488:
489:         ! z-coordinte of Vertex 1
490:         x_vertex(3, 1) = rm3d%x_start(3)
491:         ! z-coordinte of Vertex 2
492:         x_vertex(3, 2) = rm3d%x_start(3)
493:         ! z-coordinte of Vertex 3
494:         x_vertex(3, 3) = rm3d%x_start(3)
495:         ! z-coordinte of Vertex 4

```



```

496:      x_vertex(3, 4) = rm3d%x_start(3)
497:
498:      ELSE IF( k .EQ. rm3d%n_x(3) ) THEN
499:
500:        ! z-coordinte of Vertex 5
501:        x_vertex(3, 5) = rm3d%x_end(3)
502:        ! z-coordinte of Vertex 6
503:        x_vertex(3, 6) = rm3d%x_end(3)
504:        ! z-coordinte of Vertex 7
505:        x_vertex(3, 7) = rm3d%x_end(3)
506:        ! z-coordinte of Vertex 8
507:        x_vertex(3, 8) = rm3d%x_end(3)
508:
509:      END IF
510:
511:      !-----
512:
513:      ! Element no.
514:      ie = ie+1
515:
516:      IF( le3d_nnodes .EQ. 8 ) THEN
517:
518:        ! na = 1, 2, 3, 4, 5, 6, 7, 8
519:        DO na = 1, 8
520:
521:          x_local(1, na) = x_vertex(1, na)
522:          x_local(2, na) = x_vertex(2, na)
523:          x_local(3, na) = x_vertex(3, na)
524:
525:        END DO
526:
527:      END IF
528:
529:      DO na = 1, le3d_nnodes
530:
531:        id = es3d_connectivity(na, ie)

```

```

532:
533:     ns3d_x(1, id) = x_local(1, na)
534:     ns3d_x(2, id) = x_local(2, na)
535:     ns3d_x(3, id) = x_local(3, na)
536:
537:     END DO
538:
539:     !-----
540:
541:     ! Coordinates of Vertex 1
542:     x_vertex(1, 1) = x_vertex(1, 2)
543:     x_vertex(2, 1) = x_vertex(2, 2)
544:     x_vertex(3, 1) = x_vertex(3, 2)
545:
546:     !-----
547:
548:     END DO
549:
550:     !-----
551:
552:     ! Coordinates of Vertex 1
553:     x_vertex(1, 1) = rm3d%x_start(1)
554:     x_vertex(2, 1) = x_vertex(2, 2)+delta_x(2)
555:     x_vertex(3, 1) = x_vertex(3, 2)
556:
557:     !-----
558:
559:     END DO
560:
561:     !-----
562:
563:     ! Coordinates of Vertex 1
564:     x_vertex(1, 1) = rm3d%x_start(1)
565:     x_vertex(2, 1) = rm3d%x_start(2)
566:     x_vertex(3, 1) = x_vertex(3, 2)+delta_x(3)
567:

```

```

568:      !-----
569:
570:      END DO
571:
572:      !-----
573:
574:      CALL set_nodes3d_x(rm3d%ns3d, ns3d_x)
575:
576: !-----
577:
578:      DEALLOCATE( ns3d_x )
579:
580:      DEALLOCATE( es3d_connectivity )
581:
582:      DEALLOCATE( x_local )
583:
584: !-----
585:
586:      RETURN
587:
588: !#####
589:      END SUBROUTINE cal_rectmesher3d_coordinates_hex
590: !#####
591:
592:
593: !#####
594:      SUBROUTINE del_rectmesher3d(rm3d)
595: !#####
596:
597:      TYPE(struct_rectmesher3d), INTENT(INOUT) :: rm3d
598:
599: !-----
600:
601:      NULLIFY( rm3d%ns3d )
602:      NULLIFY( rm3d%le3d )
603:      NULLIFY( rm3d%es3d )

```

```

604:
605: !-----
606:
607:     rm3d%n_x = 0
608:     rm3d%x_start = 0.0D0
609:     rm3d%x_end   = 0.0D0
610:     rm3d%x_center = 0.0D0
611:     rm3d%length_x = 0.0D0
612:
613: !-----
614:
615:     RETURN
616:
617: !#####
618:     END SUBROUTINE del_rectmesher3d
619: !#####
620:
621:
622: !#####
623:     END MODULE mod_rectmesher3d

```

7. アプリケーションモジュール mod_appli.f90

```

1:     MODULE mod_appli
2: !#####
3:
4:     USE mod_nodes3d
5:     USE mod_localelement3d
6:     USE mod_elements3d
7:     USE mod_rectmesher3d
8:
9: !-----
10:
11:     IMPLICIT NONE
12:
13: !-----

```

```

14:
15:     TYPE(struct_nodes3d), POINTER      :: ns3d
16:     TYPE(struct_localelement3d), POINTER :: le3d
17:     TYPE(struct_elements3d), POINTER   :: es3d
18:     TYPE(struct_rectmesher3d), POINTER :: rm3d
19:
20: !-----
21:
22:     CONTAINS
23:
24:
25:     ! Start appli
26: !#####
27:     SUBROUTINE start_appli()
28: !#####
29:
30:     INTEGER :: ns3d_n
31:     INTEGER :: le3d_nboundaries
32:     INTEGER :: le3d_nnodes
33:     INTEGER :: es3d_n
34:     INTEGER :: rm3d_n_x(3)
35:
36:     REAL(8) :: rm3d_x_start(3)
37:     REAL(8) :: rm3d_x_end(3)
38:
39:     CHARACTER(1) :: dataname
40:
41: !-----
42:
43:     ALLOCATE( ns3d )
44:     ALLOCATE( le3d )
45:     ALLOCATE( es3d )
46:     ALLOCATE( rm3d )
47:
48: !-----
49:

```

```

50:      OPEN(13, FILE = 'param_meshing.dat')
51:
52:      READ(13, *) dataname
53:      READ(13, *) rm3d_n_x(1), rm3d_n_x(2), rm3d_n_x(3)
54:      READ(13, *) dataname
55:      READ(13, *) rm3d_x_start(1), rm3d_x_start(2), rm3d_x_start(3)
56:      READ(13, *) dataname
57:      READ(13, *) rm3d_x_end(1), rm3d_x_end(2), rm3d_x_end(3)
58:      READ(13, *) dataname
59:
60:      CLOSE(13)
61:
62: !-----
63:
64:      CALL init_rectmesher3d          &
65:          (rm3d, ns3d, le3d, es3d,    &
66:          rm3d_n_x, rm3d_x_start, rm3d_x_end)
67:
68: !-----
69:
70:      CALL get_nodes3d_n(ns3d, ns3d_n)
71:
72:      CALL get_localelement3d_nboundaries(le3d, le3d_nboundaries)
73:      CALL get_localelement3d_nnodes(le3d, le3d_nnodes)
74:
75:      CALL get_elements3d_n(es3d, es3d_n)
76:
77: !-----
78:
79:      CALL init_nodes3d(ns3d, ns3d_n)
80:
81:      CALL init_localelement3d          &
82:          (le3d, le3d_nboundaries, le3d_nnodes)
83:
84:      CALL init_elements3d(es3d, ns3d, le3d, es3d_n)
85:

```

```

86: !-----
87:
88:     RETURN
89:
90: !#####
91:     END SUBROUTINE start_appli
92: !#####
93:
94:
95:     ! Run appli
96: !#####
97:     SUBROUTINE run_appli()
98: !#####
99:
100:    INTEGER :: ns3d_n
101:    INTEGER, ALLOCATABLE :: ns3d_bc(:)
102:    INTEGER :: le3d_nboundaries
103:    INTEGER :: le3d_nnodes
104:    INTEGER :: es3d_n
105:    INTEGER, ALLOCATABLE :: es3d_connectivity(:, :)
106:    INTEGER :: es3d_ie_max_volume
107:    INTEGER :: es3d_ie_min_volume
108:    INTEGER :: i
109:    INTEGER :: id
110:    INTEGER :: na
111:    INTEGER :: ie
112:
113:    REAL(8), ALLOCATABLE :: ns3d_x(:, :)
114:    REAL(8), ALLOCATABLE :: ns3d_u(:)
115:    REAL(8), ALLOCATABLE :: es3d_volume(:)
116:    REAL(8) :: es3d_max_volume
117:    REAL(8) :: es3d_min_volume
118:    REAL(8) :: es3d_sum_volume
119:
120: !-----
121:

```

```

122:      CALL cal_rectmesher3d(rm3d)
123:
124: !-----
125:
126:      CALL get_nodes3d_n(ns3d, ns3d_n)
127:
128:      ALLOCATE( ns3d_x(3, ns3d_n) )
129:      CALL get_nodes3d_x(ns3d, ns3d_x)
130:      ALLOCATE( ns3d_u(3*ns3d_n) )
131:      ns3d_u = 0.0D0
132:      ALLOCATE( ns3d_bc(3*ns3d_n) )
133:      ns3d_bc = 0
134:
135:      CALL get_localelement3d_nboundaries(le3d, le3d_nboundaries)
136:      CALL get_localelement3d_nnodes(le3d, le3d_nnodes)
137:
138:      CALL get_elements3d_n(es3d, es3d_n)
139:      ALLOCATE( es3d_volume(es3d_n) )
140:      CALL get_elements3d_volume          &
141:      (es3d, es3d_volume,                  &
142:      es3d_max_volume, es3d_ie_max_volume, &
143:      es3d_min_volume, es3d_ie_min_volume, &
144:      es3d_sum_volume)
145:      ALLOCATE( es3d_connectivity(le3d_nnodes, es3d_n) )
146:      CALL get_elements3d_connectivity(es3d, es3d_connectivity)
147:
148: !-----
149:
150:      OPEN(10, FILE = 'mesh.dat')
151:
152:      WRITE(10, '(A)') '!NODE'
153:
154:      DO id = 1, ns3d_n
155:
156:      WRITE( 10, '( I8, 3(A, E17.8) )' )          &
157:      id, ( ', ', ns3d_x(i, id), i = 1, 3 )

```



```

158:
159:     END DO
160:
161:     WRITE( 10, ' (A, 3(A, I3) )' )          &
162:         '!ELEMENT', ', ', ', ', le3d_nboundaries, &
163:         ', ', ', ', le3d_nnodes, ', ', ', 2
164:
165:
166:     DO ie = 1, es3d_n
167:
168:         WRITE( 10, ' ( I8, 27(A, I8) )' )      &
169:             ie, ( ', ', es3d_connectivity(na, ie), &
170:                 na = 1, le3d_nnodes )
171:
172:     END DO
173:
174:     WRITE(10, ' (A)' ) '!END'
175:
176:     CLOSE(10)
177:
178: !-----
179:
180:     OPEN(11, FILE = 'ic.dat')
181:
182:     WRITE(11, ' (A)' ) '!DISPLACEMENT'
183:
184:     DO id = 1, ns3d_n
185:
186:         WRITE( 11, ' (I8, 3(A, E17.8) )' )      &
187:             id, ( ', ', ns3d_u( 3*(id-1)+i ), i = 1, 3 )
188:
189:     END DO
190:
191:     WRITE(11, ' (A)' ) '!END'
192:
193:     CLOSE(11)

```

```

194:
195: !-----
196:
197:     OPEN(12, FILE = 'bc.dat')
198:
199:     WRITE(12, '(A)') '!DISPLACEMENT'
200:
201:     DO id = 1, ns3d_n
202:
203:         WRITE( 12, '(I8, 3(A, I8) )' )          &
204:             id, ( ' ', ' ', ns3d_bc( 3*(id-1)+i ), i = 1, 3 )
205:
206:     END DO
207:
208:     WRITE(12, '(A)') '!END'
209:
210:     CLOSE(12)
211:
212: !-----
213:
214:     OPEN(14, FILE = 'mesh.inp')
215:
216:     WRITE(14, '( 5(I8, 1X) )') ns3d_n, es3d_n, 3, 13, 0
217:
218:     DO id = 1, ns3d_n
219:
220:         WRITE( 14, '( (I8, 1X), 3(E17.8, 1X) )' ) &
221:             id, ( ns3d_x(i, id), i = 1, 3 )
222:
223:     END DO
224:
225:     DO ie = 1, es3d_n
226:
227:         WRITE( 14, '( 2(I8, 1X), (A5, 1X), 27(I8, 1X) )' )          &
228:             ie, 1, ' hex',                                          &
229:             ( es3d_connectivity(na, ie), na = 1, le3d_nnodes )

```

```

230:
231:     END DO
232:
233:     WRITE(14, '( 4(I8, 1X) )') 1, 3
234:     WRITE(14, '(A)') 'DISPLACEMENT, m'
235:
236:     DO id = 1, ns3d_n
237:
238:         WRITE( 14, '( (I8, 1X), 3(E17.8, 1X) )' )      &
239:             id, ( ns3d_u( 3*(id-1)+i ), i = 1, 3 )
240:
241:     END DO
242:
243:     WRITE(14, '( 14I8 )') 1, 1
244:     WRITE(14, '( (A, 1X) )') 'VOLUME, m3'
245:
246:     DO ie = 1, es3d_n
247:
248:         WRITE( 14, '( (I8, 1X), (E17.8, 1X) )' ) &
249:             ie, es3d_volume(ie)
250:
251:     END DO
252:
253:     CLOSE(14)
254:
255: !-----
256:
257:     DEALLOCATE( ns3d_x )
258:     DEALLOCATE( ns3d_u )
259:
260:     DEALLOCATE( es3d_volume )
261:     DEALLOCATE( es3d_connectivity )
262:
263: !-----
264:
265:     RETURN

```

```

266:
267: !#####
268:     END SUBROUTINE run_appli
269: !#####
270:
271:
272: !#####
273:     SUBROUTINE finish_appli()
274: !#####
275:
276:     CALL del_nodes3d(ns3d)
277:     CALL del_localelement3d(le3d)
278:     CALL del_elements3d(es3d)
279:     CALL del_rectmesher3d(rm3d)
280:
281: !-----
282:
283:     DEALLOCATE( ns3d )
284:     DEALLOCATE( le3d )
285:     DEALLOCATE( es3d )
286:     DEALLOCATE( rm3d )
287:
288: !-----
289:
290:     RETURN
291:
292: !#####
293:     END SUBROUTINE finish_appli
294: !#####
295:
296:
297: !#####
298:     END MODULE mod_appli

```

第 4 節のプログラム：有限要素の体積計算

8. 有限要素 (計算空間) モジュール mod_localelement3d.f90

```

1:      MODULE mod_localelement3d
2: !#####
3:
4:      IMPLICIT NONE
5:
6: !-----
7:
8:      TYPE :: struct_localelement3d
9:
10:     !-----
11:
12:     PRIVATE
13:
14:     !-----
15:     !
16:     ! nboundaries
17:     ! The total number of local element boundaries
18:     !
19:     ! nedges
20:     ! The total number of local element edges
21:     !
22:     ! volume
23:     ! Local element volume
24:     !
25:     ! area_boundary
26:     ! Local element boundary area
27:     !
28:     ! nnodes
29:     ! The total number of local nodes
30:     !
31:     ! nnodes_boundary
32:     ! The total number of nodes on a local element boundary

```

```

33:      !
34:      ! nedges_boundary
35:      ! The total number of edges on a local element boundary
36:      !
37:      ! nnodes_edge
38:      ! The total number of nodes on a local element edge
39:      !
40:      ! xi(:, :)
41:      ! Cartesian coordinates of a local node
42:      ! xi, eta, zeta
43:      !
44:      ! table_na(:, :)
45:      ! Table of local element boundary no. and local node no.
46:      !
47:      !-----
48:      !
49:      ! nqps
50:      ! The total number of quadrature points
51:      !
52:      ! xi_qp(:, :)
53:      ! Cartesian coordinates of a quadrature point
54:      ! xi_qp, eta_qp, zeta_qp
55:      !
56:      ! w_qp(:, :)
57:      ! Weight of a quadrature point
58:      ! w_qp_xi, w_qp_eta, w_qp_zeta
59:      !
60:      ! n_qp(:, :)
61:      ! Shape function at a quadrature point
62:      ! N_qp
63:      !
64:      ! dndxi_qp(:, :, :)
65:      ! Partial derivatives of a shape function
66:      ! at a quadrature point
67:      ! (dN/dxi)_qp, (dN/deta)_qp, (dN/dzeta)_qp
68:      !

```

```

69:      !-----
70:
71:      INTEGER :: nboundaries
72:      INTEGER :: nedges
73:      INTEGER :: nedges_boundary
74:      INTEGER :: nnodes
75:      INTEGER :: nnodes_boundary
76:      INTEGER :: nnodes_edge
77:      INTEGER, ALLOCATABLE :: table_na(:, :)
78:      INTEGER :: nqps
79:
80:      REAL(8) :: volume
81:      REAL(8) :: area_boundary
82:      REAL(8), ALLOCATABLE :: xi(:, :)
83:      REAL(8), ALLOCATABLE :: xi_qp(:, :)
84:      REAL(8), ALLOCATABLE :: w_qp(:, :)
85:      REAL(8), ALLOCATABLE :: n_qp(:, :)
86:      REAL(8), ALLOCATABLE :: dndxi_qp(:, :, :)
87:
88:      !-----
89:
90:      END TYPE struct_localelement3d
91:
92: !-----
93:
94:      PRIVATE :: init_localelement3d_quadrature
95:
96: !-----
97:
98:      CONTAINS
99:
100:
101:      ! Set the total number of local element boundaries
102: !#####
103:      SUBROUTINE set_localelement3d_nboundaries(le3d, nboundaries)
104: !#####

```

```

105:
106:     TYPE(struct_localelement3d), INTENT(INOUT) :: le3d
107:
108:     INTEGER, INTENT(IN) :: nboundaries
109:
110: !-----
111:
112:     le3d%nboundaries = nboundaries
113:
114: !-----
115:
116:     RETURN
117:
118: !#####
119:     END SUBROUTINE set_localelement3d_nboundaries
120: !#####
121:
122:
123:     ! Get the total number of local element boundaries
124: !#####
125:     SUBROUTINE get_localelement3d_nboundaries(le3d, nboundaries)
126: !#####
127:
128:     TYPE(struct_localelement3d), INTENT(IN) :: le3d
129:
130:     INTEGER, INTENT(OUT) :: nboundaries
131:
132: !-----
133:
134:     nboundaries = le3d%nboundaries
135:
136: !-----
137:
138:     RETURN
139:
140: !#####

```



```

141:      END SUBROUTINE get_localelement3d_nboundaries
142: !#####
143:
144:
145:      ! Get the total number of local element edges
146: !#####
147:      SUBROUTINE get_localelement3d_nedges(le3d, nedges)
148: !#####
149:
150:      TYPE(struct_localelement3d), INTENT(IN) :: le3d
151:
152:      INTEGER, INTENT(OUT) :: nedges
153:
154: !-----
155:
156:      nedges = le3d%nedges
157:
158: !-----
159:
160:      RETURN
161:
162: !#####
163:      END SUBROUTINE get_localelement3d_nedges
164: !#####
165:
166:
167:      ! Get local element volume
168: !#####
169:      SUBROUTINE get_localelement3d_volume(le3d, volume)
170: !#####
171:
172:      TYPE(struct_localelement3d), INTENT(IN) :: le3d
173:
174:      REAL(8), INTENT(OUT) :: volume
175:
176: !-----

```

```

177:
178:     volume = le3d%volume
179:
180: !-----
181:
182:     RETURN
183:
184: !#####
185:     END SUBROUTINE get_localelement3d_volume
186: !#####
187:
188:
189:     ! Get local element boundary area
190: !#####
191:     SUBROUTINE get_localelement3d_area_boundary(le3d, area_boundary)
192: !#####
193:
194:     TYPE(struct_localelement3d), INTENT(IN) :: le3d
195:
196:     REAL(8), INTENT(OUT) :: area_boundary
197:
198: !-----
199:
200:     area_boundary = le3d%area_boundary
201:
202: !-----
203:
204:     RETURN
205:
206: !#####
207:     END SUBROUTINE get_localelement3d_area_boundary
208: !#####
209:
210:
211:     ! Set the total number of nodes in a local element
212: !#####

```

```

213:      SUBROUTINE set_localelement3d_nnodes(le3d, nnodes)
214: !#####
215:
216:      TYPE(struct_localelement3d), INTENT(INOUT) :: le3d
217:
218:      INTEGER, INTENT(IN) :: nnodes
219:
220: !-----
221:
222:      le3d%nnodes = nnodes
223:
224: !-----
225:
226:      RETURN
227:
228: !#####
229:      END SUBROUTINE set_localelement3d_nnodes
230: !#####
231:
232:
233:      ! Get the total number of nodes in a local element
234: !#####
235:      SUBROUTINE get_localelement3d_nnodes(le3d, nnodes)
236: !#####
237:
238:      TYPE(struct_localelement3d), INTENT(IN) :: le3d
239:
240:      INTEGER, INTENT(OUT) :: nnodes
241:
242: !-----
243:
244:      nnodes = le3d%nnodes
245:
246: !-----
247:
248:      RETURN

```

```

249:
250: !#####
251:     END SUBROUTINE get_localelement3d_nnodes
252: !#####
253:
254:
255:     ! Get the total number of nodes on a local element boundary
256: !#####
257:     SUBROUTINE get_localelement3d_nnodes_boundary &
258:         (le3d, nnodes_boundary)
259: !#####
260:
261:     TYPE(struct_localelement3d), INTENT(IN) :: le3d
262:
263:     INTEGER, INTENT(OUT) :: nnodes_boundary
264:
265: !-----
266:
267:     nnodes_boundary = le3d%nnodes_boundary
268:
269: !-----
270:
271:     RETURN
272:
273: !#####
274:     END SUBROUTINE get_localelement3d_nnodes_boundary
275: !#####
276:
277:
278:     ! Get the total number of edges on a local element boundary
279: !#####
280:     SUBROUTINE get_localelement3d_nedges_boundary &
281:         (le3d, nedges_boundary)
282: !#####
283:
284:     TYPE(struct_localelement3d), INTENT(IN) :: le3d

```

```

285:
286:     INTEGER, INTENT (OUT) :: nedges_boundary
287:
288: !-----
289:
290:     nedges_boundary = le3d%nedges_boundary
291:
292: !-----
293:
294:     RETURN
295:
296: !#####
297:     END SUBROUTINE get_localelement3d_nedges_boundary
298: !#####
299:
300:
301:     ! Get the total number of nodes on a local element edge
302: !#####
303:     SUBROUTINE get_localelement3d_nnodes_edge(le3d, nnodes_edge)
304: !#####
305:
306:     TYPE(struct_localelement3d), INTENT (IN) :: le3d
307:
308:     INTEGER, INTENT (OUT) :: nnodes_edge
309:
310: !-----
311:
312:     nnodes_edge = le3d%nnodes_edge
313:
314: !-----
315:
316:     RETURN
317:
318: !#####
319:     END SUBROUTINE get_localelement3d_nnodes_edge
320: !#####

```

```

321:
322:
323:      ! Get Cartesian coordinates of a local node
324: !#####
325:      SUBROUTINE get_localelement3d_xi(le3d, xi)
326: !#####
327:
328:      TYPE(struct_localelement3d), INTENT(IN) :: le3d
329:
330:      REAL(8), INTENT(OUT) :: xi(:, :)
331:
332: !-----
333:
334:      xi = le3d%xi
335:
336: !-----
337:
338:      RETURN
339:
340: !#####
341:      END SUBROUTINE get_localelement3d_xi
342: !#####
343:
344:
345:      ! Get table of local element boundary no. and local node no.
346: !#####
347:      SUBROUTINE get_localelement3d_table_na(le3d, table_na)
348: !#####
349:
350:      TYPE(struct_localelement3d), INTENT(IN) :: le3d
351:
352:      INTEGER, INTENT(OUT) :: table_na(:, :)
353:
354: !-----
355:
356:      table_na = le3d%table_na

```

```

357:
358: !-----
359:
360:     RETURN
361:
362: !#####
363:     END SUBROUTINE get_localelement3d_table_na
364: !#####
365:
366:
367:     ! Set the total number of quadrature points
368: !#####
369:     SUBROUTINE set_localelement3d_nqps(le3d, nqps)
370: !#####
371:
372:     TYPE(struct_localelement3d), INTENT(INOUT) :: le3d
373:
374:     INTEGER, INTENT(IN) :: nqps
375:
376: !-----
377:
378:     le3d%nqps = nqps
379:
380: !-----
381:
382:     RETURN
383:
384: !#####
385:     END SUBROUTINE set_localelement3d_nqps
386: !#####
387:
388:
389:     ! Get the total number of quadrature points
390: !#####
391:     SUBROUTINE get_localelement3d_nqps(le3d, nqps)
392: !#####

```

```

393:
394:     TYPE(struct_localelement3d), INTENT(IN) :: le3d
395:
396:     INTEGER, INTENT(OUT) :: nqps
397:
398: !-----
399:
400:     nqps = le3d%nqps
401:
402: !-----
403:
404:     RETURN
405:
406: !#####
407:     END SUBROUTINE get_localelement3d_nqps
408: !#####
409:
410:
411:     ! Get Cartesian coordinates and weight of a quadrature point
412: !#####
413:     SUBROUTINE get_localelement3d_xi_w_qp(le3d, xi_qp, w_qp)
414: !#####
415:
416:     TYPE(struct_localelement3d), INTENT(IN) :: le3d
417:
418:     REAL(8), INTENT(OUT) :: xi_qp(:, :)
419:     REAL(8), INTENT(OUT) :: w_qp(:, :)
420:
421: !-----
422:
423:     xi_qp = le3d%xi_qp
424:     w_qp = le3d%w_qp
425:
426: !-----
427:
428:     RETURN

```



```

429:
430: !#####
431:     END SUBROUTINE get_localelement3d_xi_w_qp
432: !#####
433:
434:
435:     ! Get shape function at a quadrature point
436: !#####
437:     SUBROUTINE get_localelement3d_n_qp(le3d, n_qp, dndxi_qp)
438: !#####
439:
440:     TYPE(struct_localelement3d), INTENT(IN) :: le3d
441:
442:     REAL(8), INTENT(OUT) :: n_qp(:, :)
443:     REAL(8), INTENT(OUT) :: dndxi_qp(:, :, :)
444:
445: !-----
446:
447:     n_qp = le3d%n_qp
448:     dndxi_qp = le3d%dndxi_qp
449:
450: !-----
451:
452:     RETURN
453:
454: !#####
455:     END SUBROUTINE get_localelement3d_n_qp
456: !#####
457:
458:
459:     ! Initialize localelement3d
460: !#####
461:     SUBROUTINE init_localelement3d          &
462:         (le3d, nboundaries, nnodes, nqps)
463: !#####
464:

```

```

465:      TYPE(struct_localelement3d), INTENT(INOUT) :: le3d
466:
467:      INTEGER, INTENT(IN) :: nboundaries
468:      INTEGER, INTENT(IN) :: nnodes
469:      INTEGER, INTENT(IN) :: nqps
470:
471: !-----
472:
473:      INTEGER :: nnodes_boundary
474:      INTEGER :: nnodes_edge
475:
476: !-----
477:
478:      le3d%nboundaries = nboundaries
479:
480: !-----
481:
482:      ! Hexahedral element
483:      IF( nboundaries .EQ. 6 ) THEN
484:
485:          le3d%nedges = 12
486:
487:          le3d%nedges_boundary = 4
488:
489:          le3d%volume = 8.0D0
490:
491:          le3d%area_boundary = 4.0D0
492:
493:      END IF
494:
495: !-----
496:
497:      le3d%nnodes = nnodes
498:
499: !-----
500:

```

```

501:      ALLOCATE( le3d%xi(3, nnodes) )
502:
503:      ! Hexahedral element
504:      IF( nboundaries .EQ. 6 ) THEN
505:
506:      ! Linear element
507:      IF( nnodes .EQ. 8 ) THEN
508:
509:      ! xi
510:      le3d%xi(1, 1) = -1.0D0
511:      le3d%xi(1, 2) = 1.0D0
512:      le3d%xi(1, 3) = 1.0D0
513:      le3d%xi(1, 4) = -1.0D0
514:      le3d%xi(1, 5) = -1.0D0
515:      le3d%xi(1, 6) = 1.0D0
516:      le3d%xi(1, 7) = 1.0D0
517:      le3d%xi(1, 8) = -1.0D0
518:      ! eta
519:      le3d%xi(2, 1) = -1.0D0
520:      le3d%xi(2, 2) = -1.0D0
521:      le3d%xi(2, 3) = 1.0D0
522:      le3d%xi(2, 4) = 1.0D0
523:      le3d%xi(2, 5) = -1.0D0
524:      le3d%xi(2, 6) = -1.0D0
525:      le3d%xi(2, 7) = 1.0D0
526:      le3d%xi(2, 8) = 1.0D0
527:      ! zeta
528:      le3d%xi(3, 1) = -1.0D0
529:      le3d%xi(3, 2) = -1.0D0
530:      le3d%xi(3, 3) = -1.0D0
531:      le3d%xi(3, 4) = -1.0D0
532:      le3d%xi(3, 5) = 1.0D0
533:      le3d%xi(3, 6) = 1.0D0
534:      le3d%xi(3, 7) = 1.0D0
535:      le3d%xi(3, 8) = 1.0D0
536:

```

```

537:      END IF
538:
539:      END IF
540:
541:      !-----
542:
543:      ! Hexahedral element
544:      IF( nboundaries .EQ. 6 ) THEN
545:
546:          ! Linear element
547:          IF( nnodes .EQ. 8 ) THEN
548:
549:              nnodes_boundary = 4
550:              nnodes_edge = 2
551:
552:          END IF
553:
554:      END IF
555:
556:      le3d%nnodes_boundary = nnodes_boundary
557:      le3d%nnodes_edge = nnodes_edge
558:
559:      !-----
560:
561:      ALLOCATE( le3d%table_na(nnodes_boundary, nboundaries) )
562:
563:      ! Hexahedral element
564:      IF( nboundaries .EQ. 6 ) THEN
565:
566:          ! ma = 1
567:          le3d%table_na(1, 1) = 4
568:          le3d%table_na(2, 1) = 3
569:          le3d%table_na(3, 1) = 2
570:          le3d%table_na(4, 1) = 1
571:
572:          ! ma = 2
572:          le3d%table_na(1, 2) = 5

```

```

573:      le3d%table_na(2, 2) = 6
574:      le3d%table_na(3, 2) = 7
575:      le3d%table_na(4, 2) = 8
576:      ! ma = 3
577:      le3d%table_na(1, 3) = 1
578:      le3d%table_na(2, 3) = 2
579:      le3d%table_na(3, 3) = 6
580:      le3d%table_na(4, 3) = 5
581:      ! ma = 4
582:      le3d%table_na(1, 4) = 2
583:      le3d%table_na(2, 4) = 3
584:      le3d%table_na(3, 4) = 7
585:      le3d%table_na(4, 4) = 6
586:      ! ma = 5
587:      le3d%table_na(1, 5) = 3
588:      le3d%table_na(2, 5) = 4
589:      le3d%table_na(3, 5) = 8
590:      le3d%table_na(4, 5) = 7
591:      ! ma = 6
592:      le3d%table_na(1, 6) = 4
593:      le3d%table_na(2, 6) = 1
594:      le3d%table_na(3, 6) = 5
595:      le3d%table_na(4, 6) = 8
596:
597:      END IF
598:
599: !-----
600:
601:      CALL init_localelement3d_quadrature(le3d, nqps)
602:
603: !-----
604:
605:      RETURN
606:
607: !#####
608:      END SUBROUTINE init_localelement3d

```

```

609: !#####
610:
611:
612:      ! Initialize localelement3d (Gaussian quadrature)
613: !#####
614:      SUBROUTINE init_localelement3d_quadrature(le3d, nqps)
615: !#####
616:
617:      TYPE(struct_localelement3d), INTENT(INOUT) :: le3d
618:
619:      INTEGER, INTENT(IN) :: nqps
620:
621: !-----
622:
623:      INTEGER :: na
624:      INTEGER :: i, j, k
625:      INTEGER :: nqps_tot
626:      INTEGER :: ijk
627:
628:      REAL(8) :: coord(7, 7)
629:      REAL(8) :: weight(7, 7)
630:      REAL(8) :: xi, eta, zeta
631:      REAL(8) :: n_xi, n_eta, n_zeta
632:      REAL(8) :: dn_xi, dn_eta, dn_zeta
633:
634: !-----
635:
636:      ! Coordinate and weight of Gaussian quadrature points
637:
638:      ! Hexahedral element
639:      IF( le3d%nboundaries .EQ. 6 ) THEN
640:
641:      !-----
642:
643:      ! 1 point
644:      coord(1, 1) = 0.0D0

```

```

645:      ! 2 points
646:      coord(1, 2) = -0.577350269189626D0
647:      coord(2, 2) =  0.577350269189626D0
648:      ! 3 points
649:      coord(1, 3) = -0.774596669241483D0
650:      coord(2, 3) =  0.0D0
651:      coord(3, 3) =  0.774596669241483D0
652:      ! 4 points
653:      coord(1, 4) = -0.861136311594053D0
654:      coord(2, 4) = -0.339981043584856D0
655:      coord(3, 4) =  0.339981043584856D0
656:      coord(4, 4) =  0.861136311594053D0
657:      ! 5 points
658:      coord(1, 5) = -0.906179845938664D0
659:      coord(2, 5) = -0.538469310105683D0
660:      coord(3, 5) =  0.0D0
661:      coord(4, 5) =  0.538469310105683D0
662:      coord(5, 5) =  0.906179845938664D0
663:      ! 6 points
664:      coord(1, 6) = -0.932469514203152D0
665:      coord(2, 6) = -0.661209386466265D0
666:      coord(3, 6) = -0.238619186083197D0
667:      coord(4, 6) =  0.238619186083197D0
668:      coord(5, 6) =  0.661209386466265D0
669:      coord(6, 6) =  0.932469514203152D0
670:      ! 7 points
671:      coord(1, 7) = -0.949107912342758D0
672:      coord(2, 7) = -0.741531185599394D0
673:      coord(3, 7) = -0.405845151377397D0
674:      coord(4, 7) =  0.0D0
675:      coord(5, 7) =  0.405845151377397D0
676:      coord(6, 7) =  0.741531185599394D0
677:      coord(7, 7) =  0.949107912342758D0
678:
679:      !-----
680:

```

```

681:      ! 1 point
682:      weight(1, 1) = 2.0D0
683:      ! 2 points
684:      weight(1, 2) = 1.0D0
685:      weight(2, 2) = 1.0D0
686:      ! 3 points
687:      weight(1, 3) = 0.5555555555555556D0
688:      weight(2, 3) = 0.888888888888889D0
689:      weight(3, 3) = 0.5555555555555556D0
690:      ! 4 points
691:      weight(1, 4) = 0.3478548451D0
692:      weight(2, 4) = 0.6521451548D0
693:      weight(3, 4) = 0.6521451548D0
694:      weight(4, 4) = 0.3478548451D0
695:      ! 5 points
696:      weight(1, 5) = 0.236926885056189D0
697:      weight(2, 5) = 0.478628670499366D0
698:      weight(3, 5) = 0.568888888888889D0
699:      weight(4, 5) = 0.478628670499366D0
700:      weight(5, 5) = 0.236926885056189D0
701:      ! 6 points
702:      weight(1, 6) = 0.171324492379170D0
703:      weight(2, 6) = 0.360761573048139D0
704:      weight(3, 6) = 0.467913934572691D0
705:      weight(4, 6) = 0.467913934572691D0
706:      weight(5, 6) = 0.360761573048139D0
707:      weight(6, 6) = 0.171324492379170D0
708:      ! 7 points
709:      weight(1, 7) = 0.129484966168870D0
710:      weight(2, 7) = 0.279705391489277D0
711:      weight(3, 7) = 0.381830050505119D0
712:      weight(4, 7) = 0.417959183673469D0
713:      weight(5, 7) = 0.381830050505119D0
714:      weight(6, 7) = 0.279705391489277D0
715:      weight(7, 7) = 0.129484966168870D0
716:

```



```

717:      !-----
718:
719:      END IF
720:
721: !-----
722:
723:      le3d%nqps = nqps
724:
725:      ALLOCATE( le3d%xi_qp(3, nqps*nqps*nqps) )
726:      ALLOCATE( le3d%w_qp(3, nqps*nqps*nqps) )
727:
728:      !-----
729:
730:      ! Hexahedral element
731:      IF( le3d%nboundaries .EQ. 6 ) THEN
732:
733:      !-----
734:
735:      ijk = 0
736:
737:      DO k = 1, nqps
738:
739:      DO j = 1, nqps
740:
741:      DO i = 1, nqps
742:
743:      ijk = ijk+1
744:
745:      le3d%xi_qp(1, ijk) = coord(i, nqps)
746:      le3d%xi_qp(2, ijk) = coord(j, nqps)
747:      le3d%xi_qp(3, ijk) = coord(k, nqps)
748:
749:      le3d%w_qp(1, ijk) = weight(i, nqps)
750:      le3d%w_qp(2, ijk) = weight(j, nqps)
751:      le3d%w_qp(3, ijk) = weight(k, nqps)
752:

```

```

753:      END DO
754:
755:      END DO
756:
757:      END DO
758:
759:      !-----
760:
761:      END IF
762:
763: !-----
764:
765:      nqps_tot = le3d%nqps*le3d%nqps*le3d%nqps
766:
767:      ALLOCATE( le3d%n_qp(le3d%nnodes, nqps_tot) )
768:      ALLOCATE( le3d%dndxi_qp(3, le3d%nnodes, nqps_tot) )
769:
770:      !-----
771:
772:      ! Linear element
773:      IF( le3d%nnodes .EQ. 8 ) THEN
774:
775:      !-----
776:
777:      DO ijk = 1, nqps_tot
778:
779:          xi   = le3d%xi_qp(1, ijk)
780:          eta  = le3d%xi_qp(2, ijk)
781:          zeta = le3d%xi_qp(3, ijk)
782:
783:          DO na = 1, 8
784:
785:              n_xi   = 0.5D0*( 1.0D0+le3d%xi(1, na)*xi   )
786:              n_eta  = 0.5D0*( 1.0D0+le3d%xi(2, na)*eta  )
787:              n_zeta = 0.5D0*( 1.0D0+le3d%xi(3, na)*zeta )
788:

```

```

789:      dn_xi   = 0.5D0*le3d%xi(1, na)
790:      dn_eta  = 0.5D0*le3d%xi(2, na)
791:      dn_zeta = 0.5D0*le3d%xi(3, na)
792:
793:      le3d%n_qp(na, ijk) = n_xi*n_eta*n_zeta
794:
795:      le3d%dndxi_qp(1, na, ijk) = dn_xi*n_eta *n_zeta
796:      le3d%dndxi_qp(2, na, ijk) = n_xi *dn_eta*n_zeta
797:      le3d%dndxi_qp(3, na, ijk) = n_xi *n_eta *dn_zeta
798:
799:      END DO
800:
801:      END DO
802:
803:      !-----
804:
805:      END IF
806:
807: !-----
808:
809:      RETURN
810:
811: !#####
812:      END SUBROUTINE init_localelement3d_quadrature
813: !#####
814:
815:
816:      ! Delete localelement3d
817: !#####
818:      SUBROUTINE del_localelement3d(le3d)
819: !#####
820:
821:      TYPE(struct_localelement3d), INTENT(INOUT) :: le3d
822:
823: !-----
824:

```

```

825:      IF( le3d%nboundaries .EQ. 0 ) THEN
826:
827:      RETURN
828:
829:      END IF
830:
831: !-----
832:
833:      le3d%nboundaries = 0
834:      le3d%nedges = 0
835:
836:      !-----
837:
838:      le3d%volume = 0.0D0
839:      le3d%area_boundary = 0.0D0
840:
841:      !-----
842:
843:      le3d%nnodes = 0
844:      le3d%nnodes_boundary = 0
845:      le3d%nedges_boundary = 0
846:      le3d%nnodes_edge = 0
847:
848:      DEALLOCATE( le3d%xi )
849:
850:      !-----
851:
852:      DEALLOCATE( le3d%table_na )
853:
854:      !-----
855:
856:      le3d%nqps = 0
857:
858:      DEALLOCATE( le3d%xi_qp )
859:      DEALLOCATE( le3d%w_qp )
860:

```

```

861:      DEALLOCATE( le3d%n_qp )
862:      DEALLOCATE( le3d%dndxi_qp )
863:
864: !-----
865:
866:      RETURN
867:
868: !#####
869:      END SUBROUTINE del_localelement3d
870: !#####
871:
872:
873: !#####
874:      END MODULE mod_localelement3d

```

9. 有限要素 (物理空間) モジュール mod_elements3d.f90

```

1:      MODULE mod_elements3d
2: !#####
3:
4:      USE mod_nodes3d
5:      USE mod_localelement3d
6:
7: !-----
8:
9:      IMPLICIT NONE
10:
11: !-----
12:
13:      TYPE :: struct_elements3d
14:
15:      !-----
16:
17:      PRIVATE
18:
19:      !-----

```

```

20:
21:     TYPE(struct_nodes3d), POINTER      :: ns3d => NULL()
22:     TYPE(struct_localelement3d), POINTER :: le3d => NULL()
23:
24:     !-----
25:     !
26:     ! n
27:     ! The total number of elements
28:     !
29:     ! connectivity(:, :)
30:     ! Connectivity between element no. and node no.
31:     !
32:     ! volume(:)
33:     ! Element volume
34:     !
35:     ! max_volume, max_ie_volume
36:     ! Maximum element volume and the element no.
37:     !
38:     ! min_volume_min, min_ie_volume
39:     ! Minimum element volume and the element no.
40:     !
41:     ! sum_volume
42:     ! The sum of element volumes
43:     !
44:     !-----
45:
46:     INTEGER :: n
47:     INTEGER, ALLOCATABLE :: connectivity(:, :)
48:     INTEGER :: ie_max_volume, ie_min_volume
49:
50:     REAL(8), ALLOCATABLE :: volume(:)
51:     REAL(8) :: max_volume, min_volume
52:     REAL(8) :: sum_volume
53:
54:     !-----
55:

```

```

56:      END TYPE struct_elements3d
57:
58: !-----
59:
60:      CONTAINS
61:
62:
63:      ! Set the total number of elements
64: !#####
65:      SUBROUTINE set_elements3d_n(es3d, n)
66: !#####
67:
68:      TYPE(struct_elements3d), INTENT(INOUT) :: es3d
69:
70:      INTEGER, INTENT(IN) :: n
71:
72: !-----
73:
74:      es3d%n = n
75:
76: !-----
77:
78:      RETURN
79:
80: !#####
81:      END SUBROUTINE set_elements3d_n
82: !#####
83:
84:
85:      ! Get the total number of elements
86: !#####
87:      SUBROUTINE get_elements3d_n(es3d, n)
88: !#####
89:
90:      TYPE(struct_elements3d), INTENT(IN) :: es3d
91:

```

```

92:      INTEGER, INTENT (OUT) :: n
93:
94: !-----
95:
96:      n = es3d%n
97:
98: !-----
99:
100:     RETURN
101:
102: !#####
103:     END SUBROUTINE get_elements3d_n
104: !#####
105:
106:
107:     ! Set connectivity between element no. and node no.
108: !#####
109:     SUBROUTINE set_elements3d_connectivity(es3d, connectivity)
110: !#####
111:
112:     TYPE(struct_elements3d), INTENT (INOUT) :: es3d
113:
114:     INTEGER, INTENT (IN) :: connectivity(:, :)
115:
116: !-----
117:
118:     es3d%connectivity = connectivity
119:
120: !-----
121:
122:     RETURN
123:
124: !#####
125:     END SUBROUTINE set_elements3d_connectivity
126: !#####
127:

```



```

128:
129:      ! Get connectivity between element no. and node no.
130: !#####
131:      SUBROUTINE get_elements3d_connectivity(es3d, connectivity)
132: !#####
133:
134:      TYPE(struct_elements3d), INTENT(IN) :: es3d
135:
136:      INTEGER, INTENT(OUT) :: connectivity(:, :)
137:
138: !-----
139:
140:      connectivity = es3d%connectivity
141:
142: !-----
143:
144:      RETURN
145:
146: !#####
147:      END SUBROUTINE get_elements3d_connectivity
148: !#####
149:
150:
151:      ! Get element volume
152: !#####
153:      SUBROUTINE get_elements3d_volume      &
154:          (es3d, volume,                    &
155:          max_volume, ie_max_volume, &
156:          min_volume, ie_min_volume, &
157:          sum_volume)
158: !#####
159:
160:      TYPE(struct_elements3d), INTENT(IN) :: es3d
161:
162:      REAL(8), INTENT(OUT) :: volume(:)
163:      REAL(8), INTENT(OUT) :: max_volume

```

```

164:    INTEGER, INTENT (OUT) :: ie_max_volume
165:    REAL (8), INTENT (OUT) :: min_volume
166:    INTEGER, INTENT (OUT) :: ie_min_volume
167:    REAL (8), INTENT (OUT) :: sum_volume
168:
169: !-----
170:
171:    volume = es3d%volume
172:
173:    max_volume = es3d%max_volume
174:    ie_max_volume = es3d%ie_max_volume
175:
176:    min_volume = es3d%min_volume
177:    ie_min_volume = es3d%ie_min_volume
178:
179:    sum_volume = es3d%sum_volume
180:
181: !-----
182:
183:    RETURN
184:
185: !#####
186:    END SUBROUTINE get_elements3d_volume
187: !#####
188:
189:
190:    ! Initialize elements3d
191: !#####
192:    SUBROUTINE init_elements3d(es3d, ns3d, le3d, n)
193: !#####
194:
195:    TYPE(struct_elements3d), INTENT (INOUT) :: es3d
196:
197:    TYPE(struct_nodes3d), TARGET, INTENT (IN) :: ns3d
198:    TYPE(struct_localelement3d), TARGET, INTENT (IN) :: le3d
199:

```

```

200:      INTEGER, INTENT(IN) :: n
201:
202: !-----
203:
204:      INTEGER :: le3d_nnodes
205:
206: !-----
207:
208:      es3d%ns3d => ns3d
209:      es3d%le3d => le3d
210:
211: !-----
212:
213:      CALL get_localelement3d_nnodes(es3d%le3d, le3d_nnodes)
214:
215: !-----
216:
217:      es3d%n = n
218:
219:      !-----
220:
221:      ALLOCATE( es3d%connectivity(le3d_nnodes, n) )
222:
223:      es3d%connectivity = 0
224:
225:      !-----
226:
227:      ALLOCATE( es3d%volume(n) )
228:
229:      es3d%volume = 0.0D0
230:
231:      !-----
232:
233:      es3d%max_volume = 0.0D0
234:      es3d%ie_max_volume = 0
235:

```

```

236:      es3d%min_volume = 0.000
237:      es3d%ie_min_volume = 0
238:
239:      !-----
240:
241:      es3d%sum_volume = 0.000
242:
243: !-----
244:
245:      RETURN
246:
247: !#####
248:      END SUBROUTINE init_elements3d
249: !#####
250:
251:
252:      ! Calculate elements3d
253: !#####
254:      SUBROUTINE cal_elements3d(es3d)
255: !#####
256:
257:      TYPE(struct_elements3d), INTENT(INOUT) :: es3d
258:
259: !-----
260:
261:      INTEGER :: ns3d_n
262:      INTEGER :: le3d_nnodes
263:      INTEGER :: le3d_nqps
264:      INTEGER :: nqps_tot
265:      INTEGER :: i
266:      INTEGER :: id
267:      INTEGER :: na
268:      INTEGER :: ie
269:      INTEGER :: ijk
270:
271:      REAL(8), ALLOCATABLE :: ns3d_x(:, :)

```

```

272:      REAL (8), ALLOCATABLE :: le3d_xi_qp(:, :)
273:      REAL (8), ALLOCATABLE :: le3d_w_qp(:, :)
274:      REAL (8), ALLOCATABLE :: le3d_n_qp(:, :)
275:      REAL (8), ALLOCATABLE :: le3d_dndxi_qp(:, :, :)
276:      REAL (8), ALLOCATABLE :: x_local(:, :)
277:      REAL (8) :: w_xi, w_eta, w_zeta
278:      REAL (8) :: g1(3), g2(3), g3(3)
279:      REAL (8) :: det_j
280:      REAL (8) :: w_w_w_det_j
281:
282: !-----
283:
284:      CALL get_nodes3d_n(es3d%ns3d, ns3d_n)
285:      ALLOCATE( ns3d_x(3, ns3d_n) )
286:      CALL get_nodes3d_x(es3d%ns3d, ns3d_x)
287:
288:      CALL get_localelement3d_nnodes(es3d%le3d, le3d_nnodes)
289:      CALL get_localelement3d_nqps(es3d%le3d, le3d_nqps)
290:      nqps_tot = le3d_nqps*le3d_nqps*le3d_nqps
291:      ALLOCATE( le3d_xi_qp(3, nqps_tot) )
292:      ALLOCATE( le3d_w_qp(3, nqps_tot) )
293:      CALL get_localelement3d_xi_w_qp      &
294:          (es3d%le3d, le3d_xi_qp, le3d_w_qp)
295:      ALLOCATE( le3d_n_qp(le3d_nnodes, nqps_tot) )
296:      ALLOCATE( le3d_dndxi_qp(3, le3d_nnodes, nqps_tot) )
297:      CALL get_localelement3d_n_qp      &
298:          (es3d%le3d, le3d_n_qp, le3d_dndxi_qp)
299:
300:      ALLOCATE( x_local(3, le3d_nnodes) )
301:
302: !-----
303:
304:      DO ie = 1, es3d%n
305:
306:          !-----
307:

```

```

308:      es3d%volume(ie) = 0.0D0
309:
310:      !-----
311:
312:      DO na = 1, le3d_nnodes
313:
314:         id = es3d%connectivity(na, ie)
315:
316:         DO i = 1, 3
317:
318:            x_local(i, na) = ns3d_x(i, id)
319:
320:         END DO
321:
322:      END DO
323:
324:      !-----
325:
326:      DO ijk = 1, nqps_tot
327:
328:         !-----
329:
330:         ! Covariant basis vector
331:         DO i = 1, 3
332:
333:            g1(i) = 0.0D0
334:            g2(i) = 0.0D0
335:            g3(i) = 0.0D0
336:
337:            DO na = 1, le3d_nnodes
338:
339:               g1(i) = g1(i)+le3d_dndxi_qp(1, na, ijk)*x_local(i, na)
340:               g2(i) = g2(i)+le3d_dndxi_qp(2, na, ijk)*x_local(i, na)
341:               g3(i) = g3(i)+le3d_dndxi_qp(3, na, ijk)*x_local(i, na)
342:
343:            END DO

```

```

344:
345:     END DO
346:
347:     !-----
348:
349:     ! Jacobian
350:     det_j = g1(1)*( g2(2)*g3(3)-g2(3)*g3(2) ) &
351:             +g1(2)*( g2(3)*g3(1)-g2(1)*g3(3) ) &
352:             +g1(3)*( g2(1)*g3(2)-g2(2)*g3(1) )
353:
354:     w_w_w_det_j                                     &
355:     = le3d_w_qp(1, ijk)*le3d_w_qp(2, ijk)*le3d_w_qp(3, ijk) &
356:       *det_j
357:
358:     !-----
359:
360:     es3d%volume(ie) = es3d%volume(ie)+w_w_w_det_j
361:
362:     !-----
363:
364:     END DO
365:
366:     !-----
367:
368:     END DO
369:
370: !-----
371:
372:     es3d%max_volume = MAXVAL( es3d%volume )
373:
374:     es3d%ie_max_volume = 0
375:
376:     DO ie = 1, es3d%n
377:
378:         IF( es3d%volume(ie) .EQ. es3d%max_volume ) THEN
379:

```

```

380:      es3d%ie_max_volume = ie
381:
382:      END IF
383:
384:      END DO
385:
386:      !-----
387:
388:      es3d%min_volume = MINVAL( es3d%volume )
389:
390:      es3d%ie_min_volume = 0
391:
392:      DO ie = 1, es3d%n
393:
394:         IF( es3d%volume(ie) .EQ. es3d%min_volume ) THEN
395:
396:            es3d%ie_min_volume = ie
397:
398:         END IF
399:
400:      END DO
401:
402:      !-----
403:
404:      es3d%sum_volume = SUM( es3d%volume )
405:
406:      !-----
407:
408:      DEALLOCATE( ns3d_x )
409:
410:      DEALLOCATE( le3d_xi_qp )
411:      DEALLOCATE( le3d_w_qp )
412:      DEALLOCATE( le3d_n_qp )
413:      DEALLOCATE( le3d_dndx_i_qp )
414:
415:      DEALLOCATE( x_local )

```



```

416:
417: !-----
418:
419:     RETURN
420:
421: !#####
422:     END SUBROUTINE cal_elements3d
423: !#####
424:
425:
426:     ! Delete elements3d
427: !#####
428:     SUBROUTINE del_elements3d(es3d)
429: !#####
430:
431:     TYPE(struct_elements3d), INTENT(INOUT) :: es3d
432:
433: !-----
434:
435:     IF( es3d%n .EQ. 0 ) THEN
436:
437:         RETURN
438:
439:     END IF
440:
441: !-----
442:
443:     NULLIFY( es3d%ns3d )
444:     NULLIFY( es3d%le3d )
445:
446: !-----
447:
448:     es3d%n = 0
449:
450:     !-----
451:

```

```

452:      DEALLOCATE( es3d%connectivity )
453:
454:      !-----
455:
456:      DEALLOCATE( es3d%volume )
457:
458:      !-----
459:
460:      es3d%max_volume = 0.0D0
461:      es3d%ie_max_volume = 0
462:
463:      es3d%min_volume = 0.0D0
464:      es3d%ie_min_volume = 0
465:
466:      !-----
467:
468:      es3d%sum_volume = 0.0D0
469:
470:      !-----
471:
472:      RETURN
473:
474:      !#####
475:      END SUBROUTINE del_elements3d
476:      !#####
477:
478:
479:      !#####
480:      END MODULE mod_elements3d

```

10. アプリケーションモジュール mod_appli.f90

```

1:      MODULE mod_appli
2:      !#####
3:
4:      USE mod_nodes3d

```

```

5:      USE mod_localelement3d
6:      USE mod_elements3d
7:      USE mod_rectmesher3d
8:
9: !-----
10:
11:      IMPLICIT NONE
12:
13: !-----
14:
15:      TYPE(struct_nodes3d), POINTER      :: ns3d
16:      TYPE(struct_localelement3d), POINTER :: le3d
17:      TYPE(struct_elements3d), POINTER   :: es3d
18:      TYPE(struct_rectmesher3d), POINTER :: rm3d
19:
20: !-----
21:
22:      CONTAINS
23:
24:
25:      ! Start appli
26: !#####
27:      SUBROUTINE start_appli()
28: !#####
29:
30:      INTEGER :: ns3d_n
31:      INTEGER :: le3d_nboundaries
32:      INTEGER :: le3d_nnodes
33:      INTEGER :: le3d_nqps
34:      INTEGER :: es3d_n
35:      INTEGER :: rm3d_n_x(3)
36:
37:      REAL(8) :: rm3d_x_start(3)
38:      REAL(8) :: rm3d_x_end(3)
39:
40:      CHARACTER(1) :: dataname

```

```

41:
42: !-----
43:
44:     ALLOCATE( ns3d )
45:     ALLOCATE( le3d )
46:     ALLOCATE( es3d )
47:     ALLOCATE( rm3d )
48:
49: !-----
50:
51:     OPEN(13, FILE = 'param_meshing.dat')
52:
53:     READ(13, *) dataname
54:     READ(13, *) rm3d_n_x(1), rm3d_n_x(2), rm3d_n_x(3)
55:     READ(13, *) dataname
56:     READ(13, *) rm3d_x_start(1), rm3d_x_start(2), rm3d_x_start(3)
57:     READ(13, *) dataname
58:     READ(13, *) rm3d_x_end(1), rm3d_x_end(2), rm3d_x_end(3)
59:     READ(13, *) dataname
60:
61:     CLOSE(13)
62:
63: !-----
64:
65:     CALL init_rectmesher3d                &
66:         (rm3d, ns3d, le3d, es3d,          &
67:          rm3d_n_x, rm3d_x_start, rm3d_x_end)
68:
69: !-----
70:
71:     CALL get_nodes3d_n(ns3d, ns3d_n)
72:
73:     CALL get_localelement3d_nboundaries(le3d, le3d_nboundaries)
74:     CALL get_localelement3d_nnodes(le3d, le3d_nnodes)
75:     CALL get_localelement3d_nqps(le3d, le3d_nqps)
76:

```

```

77:      CALL get_elements3d_n(es3d, es3d_n)
78:
79:      !-----
80:
81:      CALL init_nodes3d(ns3d, ns3d_n)
82:
83:      CALL init_localelement3d                                &
84:          (le3d, le3d_nboundaries, le3d_nnodes, le3d_nqps)
85:
86:      CALL init_elements3d(es3d, ns3d, le3d, es3d_n)
87:
88:      !-----
89:
90:      RETURN
91:
92:      !#####
93:      END SUBROUTINE start_appli
94:      !#####
95:
96:
97:      ! Run appli
98:      !#####
99:      SUBROUTINE run_appli()
100:      !#####
101:
102:      INTEGER :: ns3d_n
103:      INTEGER, ALLOCATABLE :: ns3d_bc(:)
104:      INTEGER :: le3d_nboundaries
105:      INTEGER :: le3d_nnodes
106:      INTEGER :: es3d_n
107:      INTEGER, ALLOCATABLE :: es3d_connectivity(:, :)
108:      INTEGER :: es3d_ie_max_volume
109:      INTEGER :: es3d_ie_min_volume
110:      INTEGER :: i
111:      INTEGER :: id
112:      INTEGER :: na

```

```

113:    INTEGER :: ie
114:
115:    REAL(8), ALLOCATABLE :: ns3d_x(:, :)
116:    REAL(8), ALLOCATABLE :: ns3d_u(:)
117:    REAL(8), ALLOCATABLE :: es3d_volume(:)
118:    REAL(8) :: es3d_max_volume
119:    REAL(8) :: es3d_min_volume
120:    REAL(8) :: es3d_sum_volume
121:    REAL(8) :: rm3d_x_start(3)
122:    REAL(8) :: rm3d_x_end(3)
123:    REAL(8) :: rm3d_x_center(3)
124:    REAL(8) :: rm3d_length_x(3)
125:
126: !-----
127:
128:    CALL cal_rectmesher3d(rm3d)
129:
130:    CALL cal_elements3d(es3d)
131:
132: !-----
133:
134:    CALL get_nodes3d_n(ns3d, ns3d_n)
135:
136:    ALLOCATE( ns3d_x(3, ns3d_n) )
137:    CALL get_nodes3d_x(ns3d, ns3d_x)
138:    ALLOCATE( ns3d_u(3*ns3d_n) )
139:    ns3d_u = 0.0D0
140:    ALLOCATE( ns3d_bc(3*ns3d_n) )
141:    ns3d_bc = 0
142:
143:    CALL get_localelement3d_nboundaries(le3d, le3d_nboundaries)
144:    CALL get_localelement3d_nnodes(le3d, le3d_nnodes)
145:
146:    CALL get_elements3d_n(es3d, es3d_n)
147:    ALLOCATE( es3d_volume(es3d_n) )
148:    CALL get_elements3d_volume           &

```

```

149:      (es3d, es3d_volume,      &
150:      es3d_max_volume, es3d_ie_max_volume, &
151:      es3d_min_volume, es3d_ie_min_volume, &
152:      es3d_sum_volume)
153:      ALLOCATE( es3d_connectivity(le3d_nnodes, es3d_n) )
154:      CALL get_elements3d_connectivity(es3d, es3d_connectivity)
155:      CALL get_rectmesher3d_x_start_x_end  &
156:      (rm3d, rm3d_x_start, rm3d_x_end, &
157:      rm3d_x_center, rm3d_length_x)
158:
159: !-----
160:
161:      OPEN(10, FILE = 'mesh.dat')
162:
163:      WRITE(10, ' (A)') '!NODE'
164:
165:      DO id = 1, ns3d_n
166:
167:          WRITE( 10, ' ( I8, 3(A, E17.8) )' )      &
168:          id, ( ', ', ns3d_x(i, id), i = 1, 3 )
169:
170:      END DO
171:
172:      WRITE( 10, ' (A, 3(A, I3) )' )      &
173:      '!ELEMENT', ', ', le3d_nboundaries, &
174:      ', ', le3d_nnodes, ', ', 2
175:
176:
177:      DO ie = 1, es3d_n
178:
179:          WRITE( 10, ' ( I8, 27(A, I8) )' )      &
180:          ie, ( ', ', es3d_connectivity(na, ie), &
181:          na = 1, le3d_nnodes )
182:
183:      END DO
184:

```

```

185:      WRITE(10, '(A)') '!END'
186:
187:      CLOSE(10)
188:
189: !-----
190:
191:      OPEN(11, FILE = 'ic.dat')
192:
193:      WRITE(11, '(A)') '!DISPLACEMENT'
194:
195:      DO id = 1, ns3d_n
196:
197:          WRITE( 11, '(I8, 3(A, E17.8) )' )           &
198:              id, ( ' ', ' ', ns3d_u( 3*(id-1)+i ), i = 1, 3 )
199:
200:      END DO
201:
202:      WRITE(11, '(A)') '!END'
203:
204:      CLOSE(11)
205:
206: !-----
207:
208:      OPEN(12, FILE = 'bc.dat')
209:
210:      WRITE(12, '(A)') '!DISPLACEMENT'
211:
212:      DO id = 1, ns3d_n
213:
214:          WRITE( 12, '(I8, 3(A, I8) )' )           &
215:              id, ( ' ', ' ', ns3d_bc( 3*(id-1)+i ), i = 1, 3 )
216:
217:      END DO
218:
219:      WRITE(12, '(A)') '!END'
220:

```



```

221:      CLOSE(12)
222:
223: !-----
224:
225:      OPEN(14, FILE = 'mesh.inp')
226:
227:      WRITE(14, '( 5(I8, 1X) )') ns3d_n, es3d_n, 3, 13, 0
228:
229:      DO id = 1, ns3d_n
230:
231:          WRITE( 14, '( (I8, 1X), 3(E17.8, 1X) )' ) &
232:              id, ( ns3d_x(i, id), i = 1, 3 )
233:
234:      END DO
235:
236:      DO ie = 1, es3d_n
237:
238:          WRITE( 14, '( 2(I8, 1X), (A5, 1X), 27(I8, 1X) )' )      &
239:              ie, 1, ' hex',                                     &
240:              ( es3d_connectivity(na, ie), na = 1, le3d_nnodes )
241:
242:      END DO
243:
244:      WRITE(14, '( 4(I8, 1X) )') 1, 3
245:      WRITE(14, '(A)') 'DISPLACEMENT, m'
246:
247:      DO id = 1, ns3d_n
248:
249:          WRITE( 14, '( (I8, 1X), 3(E17.8, 1X) )' )      &
250:              id, ( ns3d_u( 3*(id-1)+i ), i = 1, 3 )
251:
252:      END DO
253:
254:      WRITE(14, '( 14I8 )') 1, 1
255:      WRITE(14, '( (A, 1X) )') 'VOLUME, m3'
256:

```

```

257:      DO ie = 1, es3d_n
258:
259:        WRITE( 14, '( (I8, 1X), (E17.8, 1X) )' ) &
260:          ie, es3d_volume(ie)
261:
262:      END DO
263:
264:      CLOSE(14)
265:
266: !-----
267:
268:      DEALLOCATE( ns3d_x )
269:      DEALLOCATE( ns3d_u )
270:
271:      DEALLOCATE( es3d_volume )
272:      DEALLOCATE( es3d_connectivity )
273:
274: !-----
275:
276:      RETURN
277:
278: !#####
279:      END SUBROUTINE run_appli
280: !#####
281:
282:
283: !#####
284:      SUBROUTINE finish_appli()
285: !#####
286:
287:      CALL del_nodes3d(ns3d)
288:      CALL del_localelement3d(le3d)
289:      CALL del_elements3d(es3d)
290:      CALL del_rectmesher3d(rm3d)
291:
292: !-----

```

```
293:
294:     DEALLOCATE( ns3d )
295:     DEALLOCATE( le3d )
296:     DEALLOCATE( es3d )
297:     DEALLOCATE( rm3d )
298:
299: !-----
300:
301:     RETURN
302:
303: !#####
304:     END SUBROUTINE finish_appli
305: !#####
306:
307:
308: !#####
309:     END MODULE mod_appli
```