```
0001: !************************************************************************
0002: !****      有限要素メッシュに関する解析を行うルーチン集              ****
0003: !************************************************************************
0004: !
0005: ! 体(Voume, 3-D), 面(Face, 2-D), 辺(Edge, 1-D), 節点(Node, 0-D) の関係を解析する.
0006: ! 一般に FEM では, V2N のみが与えられることが多いが, 他に V2E や E2N など他の関係が必要に
0007: ! なることがある. そこで本モジュールは, これらの関係を計算するルーチンを集めたものである.
0008: !
0009: ! なお, V2E(:,:) や E2N(:,:) 等, V -> F -> E -> N の方向へは, 矩形配列 (順変換)
0010: ! で記述できるが, その逆方向, すなわち E2V や N2E 等は jagged(ギザギザ)配列 (逆変換)
0011: ! の考え方が必要となる. また, 順方向は位相を持つが, 逆方向は位相は持たない.
0012: ! 本コード中では, V,F,E,Nを抽象化して A,B,C で表現し, A -> B -> C の方向を順方向とする.
0013: ! A -> B -> C は V -> E -> N でも V -> F -> N 等で良い.
0014:
0015: MODULE mesh_module
0016:
0017:    USE constants, ONLY: jag
0018:    USE buffer_class
0019:    USE sort_and_search_module
0020:    IMPLICIT none
0021:    PRIVATE
0022:
0023:    PUBLIC :: mk_col2elm_from_elm2DOFg
0024:    PUBLIC :: mk_A2B_from_A2C_and_preB2C
0025:    PUBLIC :: mk_B2C_from_A2B_A2C_and_preB2C
0026:    PUBLIC :: mk_tree_of_B_in_C
0027:
0028: CONTAINS
0029: !
```

```
0030:
0031: !*********************
0032:    SUBROUTINE mk_col2elm_from_elm2DOFg ( col2elm, elm2DOFg )
0033:
0034:       ! 要素番号から全体DOF番号を引くテーブル elm2DOFg(1:nDOFe,1:nelm) をもとに,
0035:       ! 並列処理のために有限要素を彩色 (Multi-Coloring) することで,
0036:       ! 色番号から要素番号を引くテーブル col2elm(1:ncol)%c(:) を返す.
0037:
0038:       ! （なお, 要素番号を色順で付替えれば, col2elm で変換する手間が省ける）
0039:
0040:       USE adj_mtx_class
0041:
0042:       TYPE(jag), ALLOCATABLE, INTENT(out  ) :: col2elm(:)   ! (ALLOCATABLE属性はF2003規約)
0043:       INTEGER,                INTENT(in   ) :: elm2DOFg(:,:)
0044:
0045:       TYPE(jag), ALLOCATABLE :: DOFg2elm(:)          ! elm2DOFgの逆引き(要素=頂点, 自由度=辺)
0046:       INTEGER,   ALLOCATABLE :: elm2col(:)
0047:       TYPE(adj_mtx_c)        :: dual_A               ! 逆引きDOFg2elmに基づくグラフの隣接行列
0048:       INTEGER :: nelm
0049:       INTEGER :: nDOF_n                              ! D条件の最小全体DOF番号 < 0, Nega
0050:       INTEGER :: nDOF_p                              ! N条件の最大全体DOF番号 > 0, Posi, +
0051:       INTEGER :: i
0052:
0053:       ! elm2DOFg(:,:)の逆引きDOFg2elm(:)%c(:)を作成
0054:       CALL mk_Y2X_from_X2Y_2D( DOFg2elm, elm2DOFg )
0055:
0056:       nelm   = SIZE  (elm2DOFg, dim=2)
0057:       nDOF_n = LBOUND(DOFg2elm, dim=1)
0058:       nDOF_p = UBOUND(DOFg2elm, dim=1)
0059:
0060:       ! 逆引きDOFg2elm(i)%c(:)に関する隣接行列 ― 要素を頂点とするグラフ ― を作成
0061:       CALL init( dual_A, nelm )
0062:       DO i = nDOF_n, nDOF_p
0063:          CALL add_clique( dual_A, DOFg2elm(i)%c )
0064:       END DO
0065:
0066: !CALL wrt( rev_A, 'elm_adj.dat') ! 全体DOFを介した要素に関する隣接行列を出力
0067:
0068:       ! グラフ頂点としての要素を彩色
0069:       ALLOCATE( elm2col(nelm) )
0070:       CALL do_coloring( dual_A, elm2col )            ! 要素を頂点とするグラフの頂点彩色
0071:       CALL final( dual_A )
0072:
0073: !OPEN(77,file="elm_color.dat")
0074: !WRITE(77,'(I5)') elm2col
0075: !CLOSE(77)
0076:
0077:       ! elm2col(:)から逆引きcol2elm(ic)%c(:)を作成
0078:       CALL mk_Y2X_from_X2Y_1D( col2elm, elm2col )
0079:       DEALLOCATE( elm2col )
0080:
0081:    END SUBROUTINE mk_col2elm_from_elm2DOFg
0082: !
```

```
0083:
0084: !*********************
0085:    SUBROUTINE mk_Y2X_from_X2Y_1D ( Y2X, X2Y, minY_, maxY_ )
0086:
0087:       ! 順引きテーブル X2Y(1:nX) から 逆引きテーブル Y2X(minY:maxY)%c(:) を作成 (c(:)は昇順)
0088:       ! (順引きは1次元配列だが, 逆引きはjagged配列なので注意)
0089:
0090:       TYPE(jag), ALLOCATABLE, INTENT(out  ) :: Y2X(:)  !(minY:maxY)  逆引き (AL…はF2003規約)
0091:       INTEGER,                INTENT(in   ) :: X2Y(:)  !(1:nX)       順引き
0092:       INTEGER,   OPTIONAL,    INTENT(in   ) :: minY_   ! MINVAL(X2Y)  maxY_ とセット
0093:       INTEGER,   OPTIONAL,    INTENT(in   ) :: maxY_   ! MAXVAL(X2Y)  minY_    〃
0094:
0095:       ! 補足) FUNCTION とせず SUBROUTINE とした理由
0096:       !                   ～ 仮引数へのALLOCATABLE属性(F2003規約)に関する注意点 ～
0097:       !
0098:       !    => ALLOCATABLEの割付け状態を戻すには, FUNCTION の戻り値では無理なので,
0099:       !       右側の引数とするか, 戻り値をPOINTER属性とする必要があるため.
0100:       !
0101:       !【説明】次のように関数で宣言しておいたほうが, 一見, 使い勝手がよさそうである…
0102:
0103:       !        FUNCTION rev_tbl ( X2Y ) RESULT ( Y2X )
0104:       !           TYPE(jag), ALLOCATABLE :: Y2X(:)
0105:       !           ALLOCATE( Y2X(-2:5) )
0106:       !                 :
0107:       !        END FUNCTION rev_tbl
0108:       !
0109:       !   しかし, 呼び出し側から次のような使い方はできない.
0110:       !
0111:       !        TYPE(jag), ALLOCATABLE :: Y2X(:)
0112:       !        Y2X = rev_tbl( X2Y )    ! エラー (関数内での割付け状態は Y2X に戻らない)
0113:       !
0114:       !   1) この関数を使う場合, 正しくは次の通り.
0115:       !
0116:       !        TYPE(jag), ALLOCATABLE :: Y2X(:)
0117:       !        ALLOCATE( Y2X(-5:8) )   ! まず割付ける (上下限分からない場合は大きめに確保)
0118:       !        Y2X( UBOUND(rev_tbl(X2Y)):LBOUND(rev_tbl(X2Y)) ) = rev_tbl( X2Y )
0119:       !
0120:       !      上下限が分かっていれば, 次のように簡潔になる.
0121:       !
0122:       !        ALLOCATE( Y2X(-2:5) )   ! まず割付ける
0123:       !        Y2X = rev_tbl( X2Y )
0124:       !
0125:       !      このように, 上下限が分かる場合には FUNCTION を使うメリットもあろうが, さもなくば
0126:       !      かえって混乱のもと (関数の戻り値にALLOCATABLE属性があっても, 呼び出し元の左辺
0127:       !      (受取側) には割付け状態は戻らない). なお, FUNCTION の戻り値への ALLOCATABLE
0128:       !      属性は, 主に次のような場合を想定していると考えられる.
0129:       !
0130:       !        WRITE(*,*) rev_tbl( X2Y )       ! 終了後に自動的に解放
0131:       !        b(:) = b(:) + rev_tbl( X2Y )    ! 評価後に    〃
0132:       !                                        ! 組込み関数 MATMUL と同じような使い方
0133:       !
0134:       !   2) FUNCTION の戻り値としたい場合, 次のようにALLOCATABLE を POINTER に (F90/95規約).
0135:       !
0136:       !        FUNCTION rev_tbl ( X2Y ) RESULT ( Y2X )
0137:       !           TYPE(jag), POINTER :: Y2X(:) ! POINTER 属性
0138:       !           ALLOCATE( Y2X(-2:5) )
0139:       !                 :
0140:       !        END FUNCTION rev_tbl
0141:       !
0142:       !      呼び出し側:
0143:       !
0144:       !        TYPE(jag), POINTER :: Y2X(:)
0145:       !        Y2X => func( X2Y )              ! ポインタの参照は = ではなく =>
0146:       !            :
0147:       !        DEALLOCATE( Y2X )               ! DEALLOCATE必須. さもなくばメモリリーク
0148:
0149:       !*********************************************
0150:       ! 補足) 仮引数へのALLOCATABLE属性(F2003規約)付与 は, そうせざるを得ないときのみに限定
0151:       !
0152:       !    なぜならば,
0153:       !       1) 仮引数に ALLOCATABLE が付いていると, 実引数に静的配列を渡せない.
0154:       !       3) POINTER との役割分割が直交しない.
0155:       !       2) 呼び出し側で割付けても, ルーチン内部でINTENT(out) ならばすべて無視される
0156:       !
0157:       !    仮引数へのALLOCATABLE属性(F2003規約)付与が必要となるのは, 本ルーチンのように
0158:       !    あらかじめサイズが分からない配列を割付ける場合である. すなわち次の2パスな場合.
```

```
0159:       !
0160:       !       1.  パス1：配列を動的割付けするためにその大きさを調査
0161:       !       2.          配列にメモリを動的割付け
0162:       !       3.  パス2：割付けされた配列に値を格納
0163:       !
0164:       !     これを F90/95規約 で行おうとすると，次の2択しかなかった．
0165:       !
0166:       !       A) 上記の 1. と 3. 用にそれぞれ ルーチンを用意し実現（呼び出しが煩雑）
0167:       !       B) ALLOCATABLE属性でなく POINTER属性 を使って実現
0168:       !                                       (DEALLOCATE しないとメモリリーク)
0169:
0170:       INTEGER, ALLOCATABLE :: nc(:) !(minY:maxY) 第Z行の成分数／カウンタ (Num. of Components)
0171:       INTEGER :: minY, maxY
0172:       INTEGER :: nX                 ! Num. of Y's components per X, total Num of X
0173:       INTEGER :: X, Y
0174:
0175:       nX  = SIZE( X2Y )             ! nelm や nedge
0176:
0177:       ! 0. 動的割付けのための予備処理
0178:       IF ( PRESENT( minY_) ) THEN
0179:          IF ( .NOT. PRESENT( maxY_) ) STOP "minY_ を指定したら maxY_ も必須"
0180:          minY = minY_
0181:       ELSE
0182:          minY = MINVAL( X2Y )
0183:       END IF
0184:
0185:       IF ( PRESENT( maxY_) ) THEN
0186:          IF ( .NOT. PRESENT( maxY_) ) STOP "maxY_ を指定したら minY_ も必須"
0187:          maxY = maxY_
0188:       ELSE
0189:          maxY = MAXVAL( X2Y )
0190:       END IF
0191:
0192:       ! 1. 逆引き Y2Xテーブル（jagged 配列）へメモリを動的割り付け
0193:       ALLOCATE ( nc(minY:maxY) )
0194:       nc(:) = 0                     ! 各Y の Y2X(Y)%c(:) に必要なメモリ量 nc(Y) を計算
0195:       Pass1: DO X = 1, nX
0196:          Y = X2Y(X)
0197:          nc(Y) = nc(Y) + 1
0198:       END DO Pass1
0199:
0200:       ALLOCATE ( Y2X(minY:maxY) )   ! 動的メモリ割付け
0201:       DO Y = minY, maxY
0202:          ALLOCATE ( Y2X(Y)%c(1:nc(Y)) )
0203:       END DO
0204:
0205:       ! 2. 逆引きデータの格納
0206:       nc(:) = 0                     ! 各Y の Y2X(Y)%c(:) への格納位置
0207:       Pass2: DO X = 1, nX
0208:          Y = X2Y(X)
0209:          nc(Y) = nc(Y) + 1
0210:          Y2X(Y)%c(nc(Y)) = X
0211:       END DO Pass2
0212:       DEALLOCATE ( nc )
0213:
0214:       DO Y = minY, maxY
0215:          CALL sort( Y2X(Y)%c )      ! 今後のために c(:) を昇順に並び替えておく
0216:       END DO
0217:
0218:    END SUBROUTINE mk_Y2X_from_X2Y_1D
0219:
0220:
0221:    SUBROUTINE mk_Y2X_from_X2Y_2D ( Y2X, X2Y, minY_, maxY_ )
0222:
0223:       ! 2次元順引きテーブル X2Y(1:nYx,1:nX) から 逆引きテーブル Y2X(minY:maxY)%c(:) を作成
0224:       ! （順引きは矩形配列だが，逆引きはjagged配列なので注意）
0225:       !
0226:       ! 本ルーチンの使用例
0227:       !  ・ elm2node ( 1:nnodee, 1:nelm  ) → node2elm ( 1:nnode )%c(:)    ( V2N → N2V )
0228:       !  ・ edge2node( 1:2,      1:nedge ) → node2edge( 1:nnode )%c(:)    ( E2N → N2E )
0229:       !  ・ elm2DOF  ( 1:nDOFe,  1:nelm  ) → DOF2elm  ( nDOF_n:nDOF_p )%c(:)
0230:       !
0231:       !     nnodee : Num. of NODEs per Element
0232:       !     nDOFe  : Num. of DOFs  per Element ( not only node but also edge )
0233:       !
0234:       !                              → X                              → Xy = p(Y)
```

```
0235:       !  X2Y(1:nYx, 1:nX) = ↓ [ 1, 1, 1, 3 ]  →  Y2X(1:nY)%c(:) =    [ 1, 2, 3 ]   nXy = 3
0236:       !                     Yx [ 2, 3, 5, 5 ]                      ↓ [ 1 ]             1
0237:       !                                                            Y [ 2, 4 ]           2
0238:       !    nYx: Num. of Y's components per X (全Xで同一)              [ ]                 0
0239:       !    nXy:   〃    X                    Y (各Yに依存)            [ 3, 4 ]            2
0240:
0241:       TYPE(jag), ALLOCATABLE, INTENT(out  ) :: Y2X(:)  !(minY:maxY)  逆引き (AL…はF2003規約)
0242:       INTEGER,                INTENT(in   ) :: X2Y(:,:)!(1:nYx,1:nX) 順引き
0243:       INTEGER,  OPTIONAL,     INTENT(in   ) :: minY_   ! MINVAL(X2Y) maxY_ とセット
0244:       INTEGER,  OPTIONAL,     INTENT(in   ) :: maxY_   ! MAXVAL(X2Y) minY_    〃
0245:
0246:       INTEGER, ALLOCATABLE :: nc(:) !(minY:maxY) 第Z行の成分数／カウンタ (Num. of Components)
0247:       INTEGER :: minY, maxY
0248:       INTEGER :: nYx, nX            ! Num. of Y's components per X, total Num of X
0249:       INTEGER :: X, Yx
0250:       INTEGER :: Y, Xy
0251:
0252:       nYx = SIZE( X2Y, DIM=1 )      ! nnodee や 2, nDOFe
0253:       nX  = SIZE( X2Y, DIM=2 )      ! nelm や nedge
0254:
0255:       IF ( PRESENT( minY_) ) THEN
0256:          IF ( .NOT. PRESENT( maxY_) ) STOP "minY_ を指定したら maxY_ も必須"
0257:          minY = minY_
0258:       ELSE
0259:          minY = MINVAL( X2Y )
0260:       END IF
0261:
0262:       IF ( PRESENT( maxY_) ) THEN
0263:          IF ( .NOT. PRESENT( maxY_) ) STOP "maxY_ を指定したら minY_ も必須"
0264:          maxY = maxY_
0265:       ELSE
0266:          maxY = MAXVAL( X2Y )
0267:       END IF
0268:
0269:       ALLOCATE ( nc(minY:maxY) )
0270:       nc(:) = 0
0271:       Pass1: DO X = 1, nX
0272:          DO Yx = 1, nYx
0273:             Y = X2Y(Yx,X)
0274:             nc(Y) = nc(Y) + 1
0275:          END DO
0276:       END DO Pass1
0277:
0278:       ALLOCATE ( Y2X(minY:maxY) )
0279:       DO Y = minY, maxY
0280:          ALLOCATE ( Y2X(Y)%c(1:nc(Y)) )
0281:       END DO
0282:
0283:       nc(:) = 0
0284:       Pass2: DO X = 1, nX
0285:          DO Yx = 1, nYx
0286:             Y = X2Y(Yx,X)
0287:             nc(Y) = nc(Y) + 1
0288:             Y2X(Y)%c(nc(Y)) = X
0289:          END DO
0290:       END DO Pass2
0291:       DEALLOCATE ( nc )
0292:
0293:       DO Y = minY, maxY
0294:          CALL sort( Y2X(Y)%c )
0295:       END DO
0296:
0297:    END SUBROUTINE mk_Y2X_from_X2Y_2D
0298: !
```