九州大学田中先生向けメッシュデータ変換プログラム開発について

# 概要

九州大学田中先生が開発した結晶塑性解析ソルバー(CLP)の入力とするために、Ansysで作成したメッシュデータファイルを変換するプログラムを開発する。メッシュデータの変換では、単にファイル形式の変換だけではなく、要素内の節点順序の整理と、要素ごとのフラグの設定も行う。

# 開発言語

Python3系

# 入出力

## 入力

Ansysが出力するメッシュデータファイル

## 出力

結晶塑性解析ソルバーの入力ファイル

節点番号ファイル (節点マージ前後の節点番号の対応関係を記述したファイル)

要素ごとの領域番号ファイル

# クラス定義

ソースファイルは、1クラス1ファイルとし、クラス名と同じファイル名にする。

## main関数

プログラム実行時に呼ばれる関数である。

以下の順序で処理を行う

1. Ansysファイルの読み込み
2. 節点のマージ
3. 要素の隣接関係の計算
4. 要素ごとのフラグの設定と、要素内の節点順序の整理
5. メッシュデータの整合性チェック
6. メッシュデータの出力

## MeshData

メッシュデータを保持するクラスである。

以下のメンバー変数と、その変数のgetter, setterを定義すること。

|  |  |
| --- | --- |
| 変数名 | 説明 |
| num\_nodes | 整数。節点の個数である。 |
| nodes\_df | 節点のデータフレームである。列名は、節点IDの'NodeId'と、XYZ軸の各座標値である'x', 'y', 'z'である。 |
| max\_coords | 節点座標値の最大値である。 |
| min\_coords | 節点座標値の最小値である。 |
| tolerance | 節点マージのときに用いる許容誤差である。 |
| dictionary\_node\_id\_to\_index | 節点Id(ファイルに書かれた値)から、節点インデックス(0始まり)への対応関係を表した辞書である。 |
| num\_elements | 整数。要素の個数である。 |
| num\_domains | 整数。領域の個数である。 |
| elements\_df | 要素のデータフレームである。列名は、  要素IDの'ElementId'と、領域番号の'DomainId'、8節点の'Node0', 'Node1', 'Node2', 'Node3', 'Node4', 'Node5', 'Node6', 'Node7'である。 |
| map\_merge\_nodes | 節点マージでの節点番号の変化を表したマップである。 |
| nodes\_df\_old | 節点マージ前の節点データである。 |
| num\_nodes\_old | 節点マージ前の節点個数である。 |
| neighbor\_element | 整数の配列。第i(i=0,1,2,...)要素の、要素内の第j(i=0,1,2,...,5 )面について、\_neighbor\_element[i][j]=この面を共有している隣接要素の番号である。  -1で埋めて初期化する |
| neighbor\_face | 整数の配列。第i(i=0,1,2,...)要素の、要素内の第j(i=0,1,2,...,5 )面について、\_neighbor\_ face[i][j]=この面を共有している隣接要素でのこの面を表す番号である。  -1で埋めて初期化する |
| num\_elements\_of\_node | 各節点が属する要素の個数の配列, 第i(i=0, 1, 2…)節点において\_num\_elements\_of\_node[i]=第i節点が属する要素の個数 |
| element\_id\_of\_node | 各節点が属する要素番号を格納した2次元配列, 第i(i=0, 1, 2…)節点が属する要素のうち第j番目について\_element\_id\_of\_node[i][j]=その要素の要素番号 |
| element\_flag | 整数の配列。第i(i=0,1,2,...)要素について、\_element\_flag [i]=要素に設定されたフラグである。初期値は、すべて0とする。 |

また、プログラムを簡便にするために、以下の定数を定義する。

|  |  |
| --- | --- |
| 定数名 | 説明 |
| ORDERED\_NODES | 節点の並び替えとしてあり得るものの配列である。全部で24通りある。 |
| ORDERED\_FACE\_NUMBER | 節点を並び替えた場合の、面番号の変わり方を表した配列である。全部で24通りある  a番目の並び変えで、並び替え前に第i面の位置に,並び替え後に第j面がくるとき、i= ORDERED\_FACE\_NUMBER[a][j]である。 |
| NODES\_OF\_FACE | 面を構成する節点のローカルな番号である。面の番号は0, 1, ... , 5とする。  {  {0, 2, 3, 1}, {0, 1, 5, 4}, {1, 3, 7, 5}, {2, 6, 7, 3}, {0, 4, 6, 2}, {4, 5, 7, 6}  }  点の順番は、外部からみて反時計周りである。 |
| NODES\_OF\_FACE\_REVERSE | NODES\_OF\_FACEに対して、面内の点の順番を、外部からみて時計回りにしたものである。面内の最初の点は同じにしている。  面の番号は0, 1, ... , 5とする。  {  {0, 1, 3, 2}, {0, 4, 5, 1}, {1, 5, 7, 3}, {2, 3, 7, 6}, {0, 2, 6, 4}, {4, 6, 7, 5}  }  点の順番は、外部からみて反時計周りである。 |
| FACE\_RELATION\_ARRAY | 節点並び替えで使用する、ペアとなるべき面の番号を表している。  面0と面5、面1と面3、面2と面4がそれぞれペアとなっている。  [5, 3, 4, 1, 2, 0] |

## AnsysFileReader

Ansysファイルの読み込みを行うクラスである。

図 4‑1は、Ansysでの要素内の節点の番号付けを表したものである。

図 4‑2は、CLPでの要素内の節点の番号付けを表したものである。

表 1にAnsysファイルの概要を示す。

Ansysファイルを読み込む際に、節点番号の付け替えを行い、CLPでの番号付けルールに変換する。変換は、node2とnode3、node6とnode7をそれぞれ入れ替えるものである。(ここで行うのは、要素内の番号付けルールをCLP流に変えただけで、要素間の整合性はこの後処理する必要がある)

Ansysファイルでは、節点番号は1始まりになっており、連番でない可能性も考慮し、プログラム内部で扱う0始まりの連番の節点番号に変換するMapを作成する。ファイルから読み込んだ要素コネクティビティに対して、このMapを適用する。

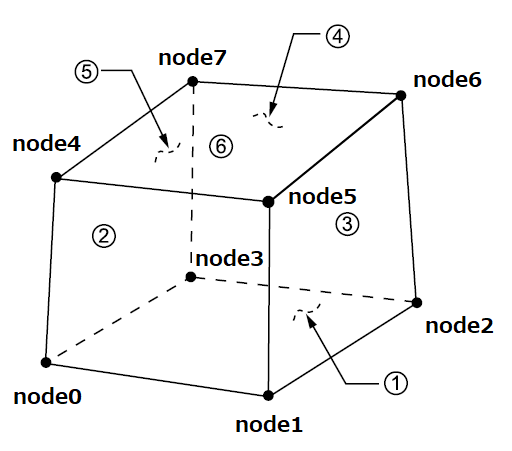


図 4‑1 Ansysでの要素内のローカルな節点番号

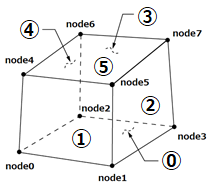


図 4‑2 CLPでの要素内のローカルな節点番号および面番号

表 1 Ansysファイルの概要

|  |
| --- |
| (略)  /com,\*\*\*\*\*\*\*\*\*\*\* Nodes for the whole assembly \*\*\*\*\*\*\*\*\*\*\* #節点座標値ブロックのはじまり  nblock,3,,29 #末尾は、このブロックの行数で、最初と最後の2行は使わない  (1i9,3e20.9e3)  1 5.000000000E-001 5.000000000E-001 5.000000000E-001 #節点番号、X,Y,Z  2 5.000000000E-001 0.000000000E+000 5.000000000E-001  (略)  26 0.000000000E+000 5.000000000E-001 1.000000000E+000  27 5.000000000E-001 5.000000000E-001 1.000000000E+000  -1  (略)  /com,\*\*\*\*\*\*\*\*\*\*\* Elements for Body 1 "SYS\Solid" \*\*\*\*\*\*\*\*\*\*\* #要素コネクティビティブロックのはじまり  et,1,185 #領域番号と要素種類番号  eblock,19,solid,,8 #末尾は要素個数  (19i9)  # 節点数　　　　　　　要素番号　節点番号1,2,3,4,5,6,7,8  1 1 1 1 0 0 0 0 8 0 1 1 11 19 23 25 15 16 24  1 1 1 1 0 0 0 0 8 0 2 1 11 17 20 23 19 18 21  (略)  1 1 1 1 0 0 0 0 8 0 7 1 2 8 20 27 4 3 22  1 1 1 1 0 0 0 0 8 0 8 1 25 6 2 27 26 5 4  -1  (略) |

## MeshFileWriter

メッシュファイルの出力を行うクラスである。

メッシュデータや、CLPフォーマットで出力する。CLPでは、節点番号・要素番号は1始まりであるが、プログラム内部ではどちらも0始まりで扱っているため、節点番号・要素番号を出力する際は1を加算する。

|  |
| --- |
| coment1 #はじめ10行はコメント  coment2  (略)  coment9  coment10  total\_nodal\_point\_number= 27　#総節点数(行の頭は空白なしの文字列)  n x y z #項目の意味を表す文字列  1 5.000000000E-001 5.000000000E-001 5.000000000E-001 #節点番号、xyz座標値  2 5.000000000E-001 0.000000000E+000 5.000000000E-001  (略)  26 0.000000000E+000 5.000000000E-001 1.000000000E+000  27 5.000000000E-001 5.000000000E-001 1.000000000E+000  data　#「data」の文字列が必要  8　#要素数  1 24 25 23 1 16 15 19 11 1　#要素番号、8節点の節点番号、四面体の分割方法を表すフラグ  2 23 1 21 20 19 11 18 17 -1  (略)  7 2 4 8 3 1 27 20 22 -1  8 6 5 2 4 25 26 1 27 1 |

## MergeNodes

節点のマージを行うクラスである。

探索の効率化のために、バケットを利用する。

節点マージ処理の概要

1. 節点のXYZ座標値が最小の点p\_min(min\_x, min\_y, min\_z)と、最大の点p\_max(max\_x, max\_y, max\_z)を計算し、バウンディングボックスを計算する。  
   また、バウンディングボックスの大きさから、誤差判定の許容値εを計算する
2. d^3=(バウンディングボックスの体積) / (節点数)となるようなdを計算する。
3. バウンディングボックスを、XYZの各方向について、p\_minを起点として、一辺dの立方体を積み上げ、バウンディングボックスを覆う。このときの1個の立方体をバケットと呼ぶ。
4. i\_before番目の節点p(x, y, z)が与えられたときに、その節点のマージ後の節点番号i\_afterを計算する。  
   xInt = (int)((x – min\_x) / d), yInt = (int)((y – min\_y) / d), zInt = (int)((z – min\_z) / d)として、座標値を整数化する。  
   位置(xInt, yInt, zInt)にあるバケットに、pとの距離がε未満の節点が存在しなければ、i\_after=(処理時点のマージ後節点数)とする。そのような節点が存在すれば、その節点のマージ後の節点番号をi\_afterに代入する。(pと近接する節点の探索を効率かするためにバケットを用いる)

## NeighborElementChecker

要素の隣接関係を計算するクラスである。

要素の隣接関係計算の概要

1. 各節点について、その節点が属する要素のリストを作成する
2. 第i要素の第j面について、この面を構成する4点を、p\_ij0, p\_ij1, p\_ij2, p\_ij3とする(NODES\_OF\_FACEを使用する)。  
   p\_ij0, p\_ij1, p\_ij2, p\_ij3の各節点が属する要素のリストに共通するi以外の要素番号を抽出し、その要素番号をkとする。(共通するi以外の要素番号kが複数存在する場合は、警告メッセージを出す。共通するi以外の要素番号が存在しない場合は、エラーではない)  
   第k要素について、第m面の節点p\_km0, p\_km1, p\_km2, p\_km3(NODES\_OF\_FACE\_REVERSEを使用する)が、p\_ij0, p\_ij1, p\_ij2, p\_ij3と並び順を含めて一致するようなmを計算する。  
   このとき、第i要素の第j面と、第k要素の第m面が共有されていることになるので、  
   \_neighbor\_element[i][j]=k  
   \_neighbor\_face[i][j]=m  
   \_neighbor\_element[k][m]=i  
   \_neighbor\_face[k][m]=j  
   と代入を行う。(代入するべき値が既に入っている場合はエラーとはみなさずに処理を続ける。)  
   ただし、\_neighbor\_element[i][j]と\_neighbor\_face[i][j]、\_neighbor\_element[k][m]と\_neighbor\_face[k][m]のいずれかでも-1でない場合はエラーかどうかをチェックする。これら四つの配列の値が、代入しようとしている値と同じ場合はエラーとはみなさない。そうでない場合は第k要素の第m面がすでに他の要素と共有されていることになるので、エラーメッセージを出力する。このエラーが出た場合は、隣接関係の計算は継続するが、隣接関係の計算が終了したらプログラムを終了する。

## ReorderElementConnectivity

要素の節点の並び順の整理と、要素のフラグの設定を行うクラスである。

処理の概要

1. 各要素が並び替え処理済みであるかを示すboolの配列isOrderedを作り、初期値としてFalseで埋める。  
   処理対象の要素番号を覚えて置くキュー(同じ番号は重複させない)を作成する。
2. 第0要素には、フラグ1を設定し、要素番号0をキューに追加する。
3. キューに要素番号が存在している限り、以下を繰り返す
4. キューから取り出した要素番号をnとする。isOrderd[n]==Trueの場合,第n要素は処理済みであるためC)に戻る。isOrderd[n]==Falseの場合,第n要素は並び替え処理済みであるとして、isOrdered[n]=Trueとする。
5. 第n要素の面のうち、隣接要素で並び替え済みであるものが存在する最小の番号をmとする。第m面で接する要素をn’とする。第n要素と第n’要素のフラグが同じ値である場合は、エラーメッセージを出力して、第n要素の処理を終了する。(プログラムは継続する)  
   第n要素と第n’要素のフラグが異なる値の場合は、第n’要素を基準として、第n要素の節点を並び替える(詳細は後述)。
6. 第n要素の第0～5面について、隣接要素の番号をn’’として、第n’’要素が並び替え処理済みでない場合は、第n’’要素に、第n要素とは異なる値のフラグ(1 or -1)を設定し、n’’をキューに追加する。

1つの要素の節点の並び変えについて

第n要素と第n’要素が隣接している場合に、第n’要素を基準にして第n要素を並び替える手順を以下に示す。

1. 隣接関係を調べて、共有されている面が、第n要素では第m面、第n’要素では第m’面であるとする。
2. 並び替え方を表す番号をaとして、a=0, ..., 23の範囲で次のチェックを行う。
3. a番目の並び替えを第n要素に適用したと仮定する。  
   m’と対になるべき面番号をm\_newとする。(m\_new =FACE\_RELATION\_ARRAY[m’])  
   第n要素に、a番目の並び替えを適用した状態で、第m\_new面の節点をp\_a\_nm0, p\_a\_nm1, p\_a\_nm2, p\_a\_nm3(ORDERED\_NODES とNODES\_OF\_FACEを使用する)とする。  
   第n’要素の第m’面の節点(こちらは並び替えない)をp\_n’m’0, p\_n’m’1, p\_n’m’2, p\_n’m’3(NODES\_OF\_FACE\_REVERSEを使用する)とする。  
   p\_a\_nm0, p\_a\_nm1, p\_a\_nm2, p\_a\_nm3とp\_n’m’0, p\_n’m’1, p\_n’m’2, p\_n’m’3が、順番も含めて一致するかを調べる。  
   一致した場合は、並び替え方aを確定する。
4. 並び替え方aが確定したので、隣接関係での面番号の書き換えを行う。  
   第n要素の第m面と、第n’要素の第m’面が共有されていた場合は、以下のように値を設定する。  
   第n要素については、j=0, ... , 5, j’= ORDERED\_FACE\_NUMBER[a][j]として、  
   \_neighbor\_element[n][j’]= \_neighbor\_element[n][j]  
   \_neighbor\_face[n][j’]= \_neighbor\_face[n][j]  
   とする。(配列の値の並び替えになるので、変更前の値を退避させておくなどの必要がある)  
   また、第n要素と隣接する要素について、\_neighbor\_faceの値を書き換える。(\_neighbor\_elementは変更なし)