AD

AD-E404 388

Technical Report ARMET-TR-20053

MATLAB SCRIPT TO CONVERT TETRAHEDRALS TO HEXAHEDRALS IN LS-DYNA

Kenneth H. Hohnecker

June 2022

U.S. ARMY COMBAT CAPABILITIES DEVELOPMENT
COMMAND ARMAMENTS CENTER

Munitions Engineering Technology Center

Picatinny Arsenal, New Jersey

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
June 2022	Final		

4. TITLE AND SUBTITLE
MATLAB Script to Convert Tetrahedrals to Hexahedrals in LS-DYNA

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
Kenneth H. Hohnecker

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
U.S. Army DEVCOM AC, METC Munition Systems Directorate (FCDD-ACM-MI) Picatinny Arsenal, NJ 07806-5000	N\A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
U.S. Army DEVCOM AC, ESIC Knowledge & Process Management Office (FCDD-ACE-K) Picatinny Arsenal, NJ 07806-5000		Technical Report ARMET-TR-20053

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This paper documents the methodology and MATLAB code used to convert linear tetrahedral elements to linear hexahedral elements for solid model components that are meshed in the LS-DYNA finite element analysis software package.

15. SUBJECT TERMS
LS-DYNA Tetrahedrals Hexahedrals MATLAB Finite element analysis

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	c. THIS PAGE		
U	U	U	SAR	29

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER (Include area code)
Kenneth Hohnecker	(973) 724-7633

Standard Form 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

CONTENTS

ACKNOWLEDGMENTS

The author would like to thank the U.S. Army Combat Capabilities Development Command (DEVCOM) Armaments Center (AC), Picatinny Arsenal, NJ, Small Caliber Munitions Modeling & Simulation team for providing valuable feedback in testing this script.

INTRODUCTION

When analyzing an object using finite element analysis (FEA) methods, it is necessary to subdivide (mesh) the object into smaller regions called elements, in order to evaluate the governing physical equations. If the object has complex three-dimensional (3D) geometry, most commercially available FEA preprocessor software can only mesh the object using four-pointed pyramids called tetrahedrals. However, if one is limited to using linear tetrahedrals, that can lead to numerical artifacts that adversely affect the results of the model; rather, linear hexahedral elements are preferred (ref. 1). This paper documents a MATLAB script that converts tetrahedrals to hexahedrals in the FEA software LS-DYNA, in order to both mesh complex parts yet have the computational advantages of hexahedrals.

MATLAB CODE

Description of Code

The code *tet_to_hex.m* first reads in a part from an LS-DYNA input file (also known as a .k file) that has been meshed with tetrahedrals (see fig. 1). The code then converts each tetrahedral to four hexahedral elements by adding a node to the midpoint of each edge, the centroid of each face, and the volumetric centroid of the tetrahedral (see fig. 2). The node-ordering scheme for LY-DYNA's tetrahedral and hexahedral elements are found in reference 2. The node-naming convention used in *tet_to_hex.m* can be found in appendix A.

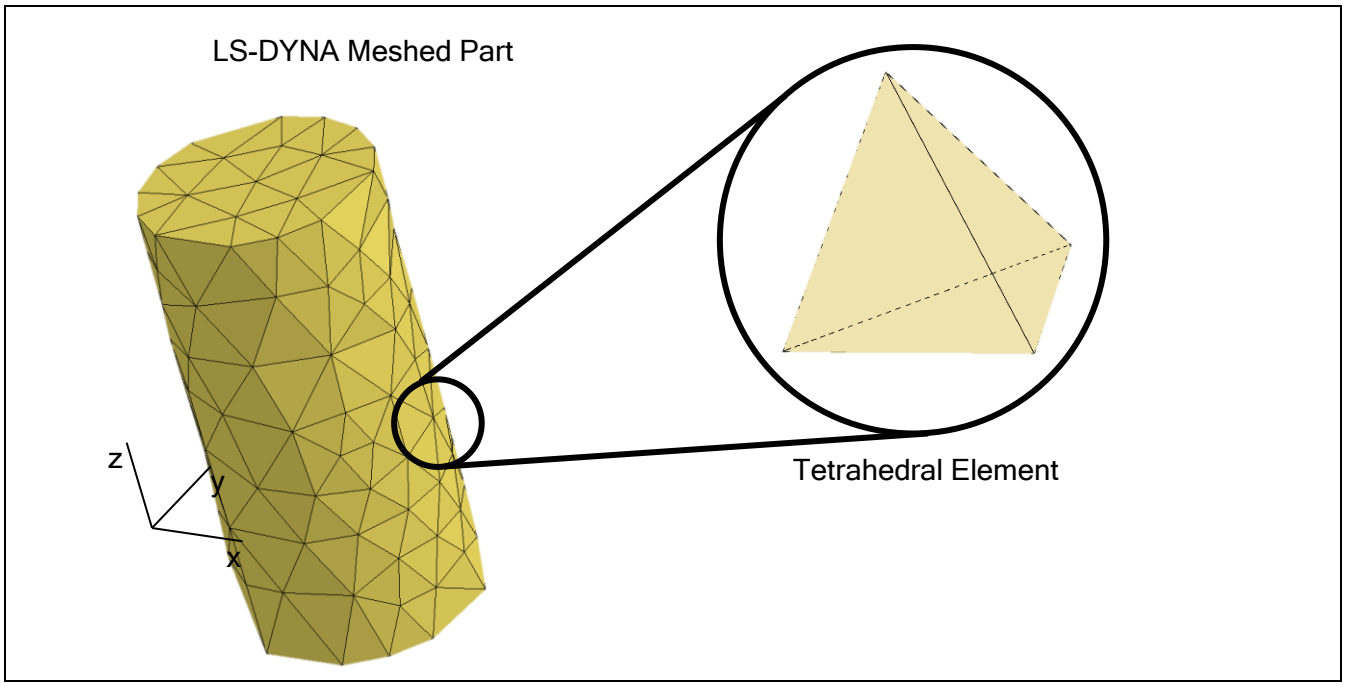


Figure 1
LS-DYNA part meshed with tetrahedrals

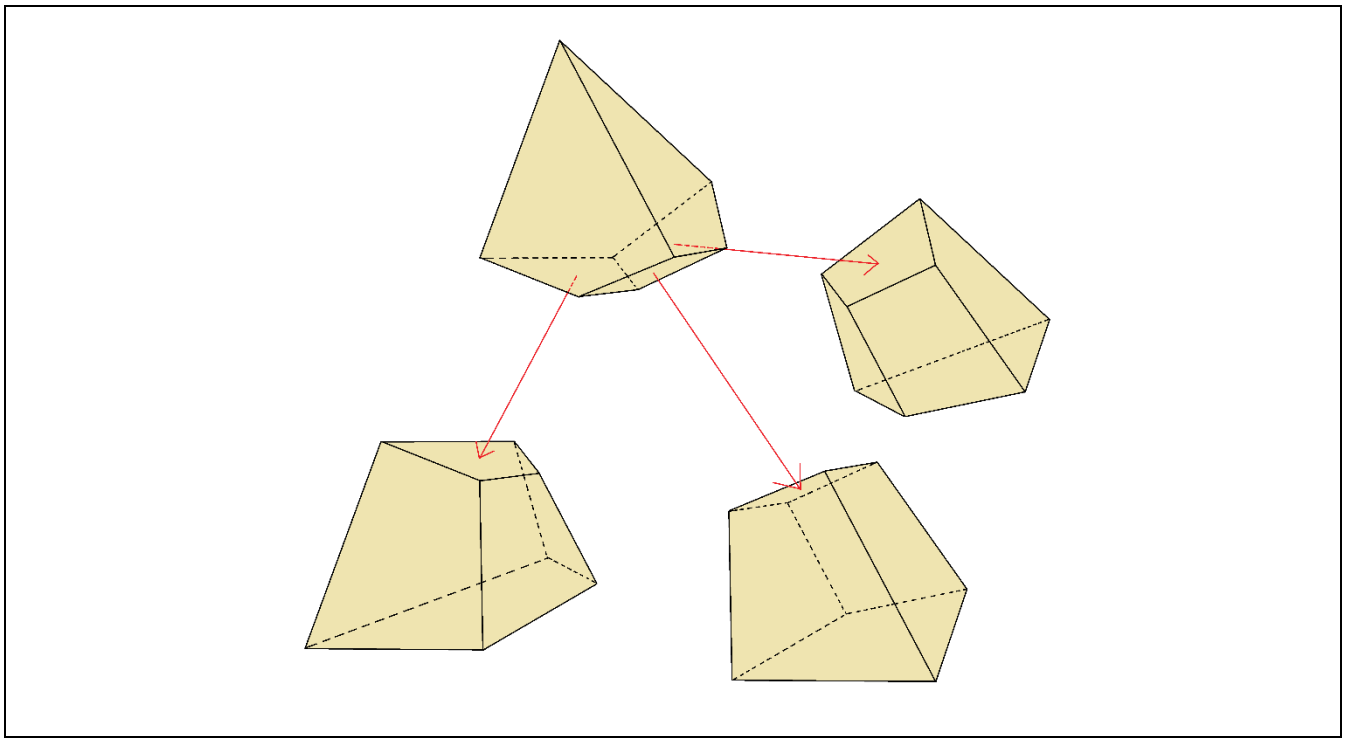


Figure 2
Tetrahedral element decomposed into four hexahedral elements

Using the Code

The MATLAB code *tet_to_hex.m* can be found in appendix B. It was written for version 2018a, but it should be compatible with earlier versions as well. After *tet_to_hex.m* is run, the dialogue box in figure 3 appears. Enter the name of the .k file (no extension needed) into the first text field.[1] Enter the part identification (ID) of the part that needs to be converted from tetrahedrals to hexahedrals into the second text field. The remainder of the inputs are only required if the part that is being converted has a symmetry plane. For example, if the top surface of the cylinder in figure 1 is a symmetry plane, the components of the normal vector to the symmetry plane would be (0, 0, 1). Then, the coordinates of a point on that symmetry plane would also need to be inputted. The next text field is the maximum perpendicular distance a node can be from the symmetry plane and still be considered part of that plane. Finally, the last text field tells the code the node set ID for the new nodes on the symmetry plane (note that a symmetry boundary condition must be created for this new node set). Once all inputs are entered, click “OK”. A new .k file will be created in the same directory as the original .k file, with “hex” appended to the end of the original file name.

[1] If the original .k file is not in the same directory as *tet_to_hex.m*, then the full path of the .k file must be entered.

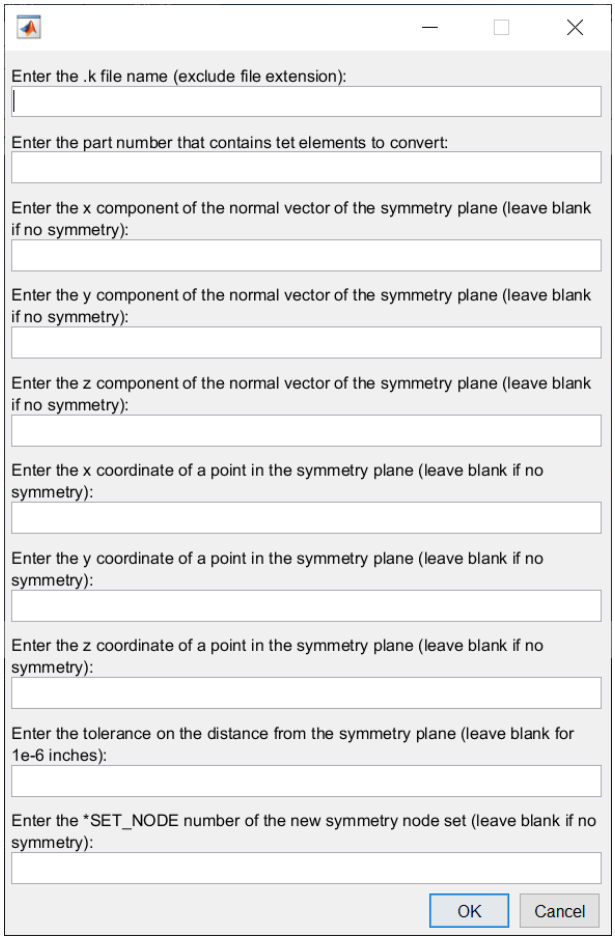


Figure 3
User-input dialogue box

There are two additional steps that need to be taken after running the code:

1) The code assumes all element cards in the original .k file are *ELEMENT_SOLID. Therefore, if the original .k file contained element cards other than *ELEMENT_SOLID, those cards in the new .k file must be changed back to their original types.

2) The *SECTION_SOLID card must be changed to a hexahedral type for the part that was converted.

SAMPLE RESULTS

The cylinder in
Figure 1figure 1 was converted using the code *tet_to_hex.m*, and the resulting part is shown in figure 4.

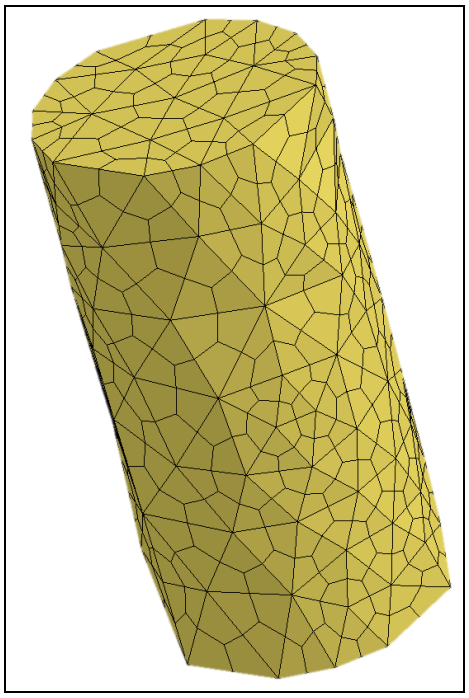


Figure 4
Converted to hexahedral part using *tet_to_hex.m*

CONCLUSIONS

This paper documented the process and use of a MATLAB script, which readily converts tetrahedrals to hexahedrals in the finite element analysis (FEA) software LS-DYNA. An example of converting a cylinder component is included to demonstrate the code. This method has the potential to greatly improve FEA simulation quality for complex parts that are traditionally difficult to model with linear tetrahedral elements.

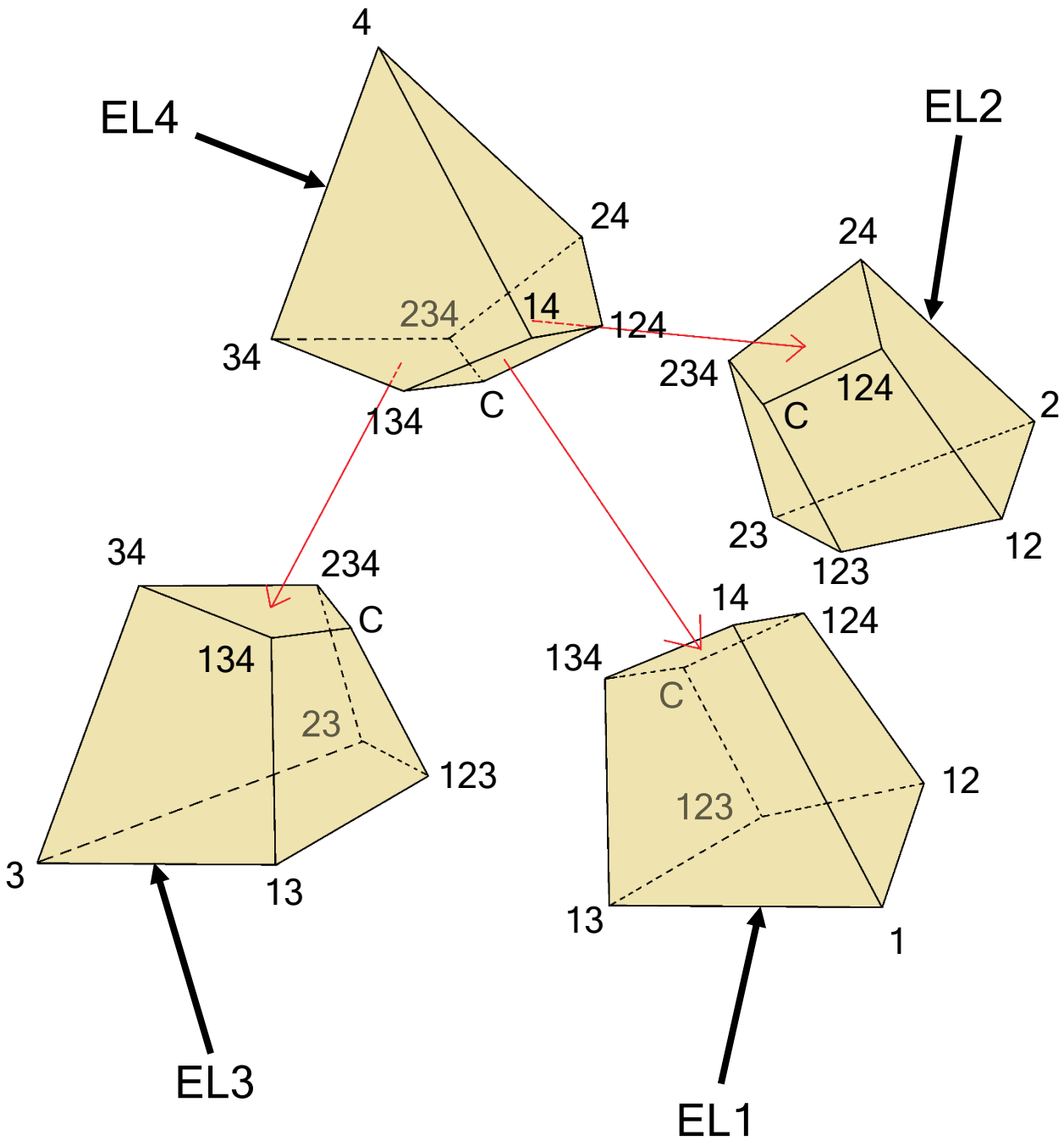
REFERENCES

1. Wang, E., Nelson, T., and Rauch, R., “Back to Elements – Tetrahedra vs. Hexahedra,” Proceedings of the 2004 international ANSYS conference, ANSYS, Pennsylvania, 2004.

2. Livermore Software Technology Corporation (LSTC), “LS-DYNA Keyword User’s Manual, Volume I,” Revision 2356, 2012.

APPENDIX A
NODE-NAMING CONVENTION USED IN *TET_TO_HEX.M*

Node-naming convention used in *tet_to_hex.m*



APPENDIX B
MATLAB CODE

MATLAB code

```
%% tet_to_hex.m

close all;
clear all;

% User input:
userInput = inputdlg({'Enter the .k file name (exclude file extension):',...
   'Enter the part number that contains tet elements to convert:',...
   ['Enter the x component of the normal vector of the symmetry plane',...
   ' (leave blank if no symmetry):'],...
   ['Enter the y component of the normal vector of the symmetry plane',...
   ' (leave blank if no symmetry):'],...
   ['Enter the z component of the normal vector of the symmetry plane',...
   ' (leave blank if no symmetry):'],...
   ['Enter the x coordinate of a point in the symmetry plane',...
   ' (leave blank if no symmetry):'],...
   ['Enter the y coordinate of a point in the symmetry plane',...
   ' (leave blank if no symmetry):'],...
   ['Enter the z coordinate of a point in the symmetry plane',...
   ' (leave blank if no symmetry):'],...
   ['Enter the tolerance on the distance from the symmetry plane',...
   ' (leave blank for 1e-6 inches):'],...
   ['Enter the *SET_NODE number of the new symmetry node set',...
   ' (leave blank if no symmetry):']});

fileName = userInput{1};

% number of part to change from tet to hex
part = str2double(userInput{2});

% if user inputted a symmetry plane
if ~isempty(userInput{3})
   xn = str2double(userInput{3});
   yn = str2double(userInput{4});
   zn = str2double(userInput{5});

   % magnitude of normal vector of symmetry plane
   mag = sqrt(xn^2 + yn^2 + zn^2);

   % normalize normal vector
   xn = xn/mag;
   yn = yn/mag;
   zn = zn/mag;

   % point in symmetry plane
   x0 = str2double(userInput{6});
   y0 = str2double(userInput{7});
   z0 = str2double(userInput{8});
```

```
    % new set number for nodes on new symmetry plane
    setNum = str2double(userInput{10});

    if isempty(userInput{9})
        tol = 1E-6; % default tolerance for distance from symmetry plane
    else
        tol = str2double(userInput{9}); % user-defined tolerance
    end
end

%%
% Read in .k file.
kFile = fopen(char(strcat(fileName, '.k')));
fileData = textscan(kFile, '%s', 'delimiter', '\n', 'whitespace', '');
fclose(kFile);

%%
% Find start and end of *ELEMENT_SOLID and *NODE cards
[elemStart, elemEnd] = card_bookends(fileData{1}, '*ELEMENT_SOLID');
[nodeStart, nodeEnd] = card_bookends(fileData{1}, '*NODE');

%%
% Create array [nodes] whose rows contain the node coordinates of the node
% number corresponding to the index of that row of [nodes]

% temporary array
tempNodes = parse_numbers(nodeStart, nodeEnd, fileData{1}, 1, 4);
tempNodes = sortrows(tempNodes); % sort temporary array

% original max node number
maxNode = max(tempNodes(:, 1));

% original min node number
minNode = min(tempNodes(:, 1));

% shift node numbers so that minimum node number is 1
tempNodes(:, 1) = tempNodes(:, 1) - minNode + 1;
maxNode = maxNode - minNode + 1;
maxNodeOriginal = maxNode;

originalNodeNums = tempNodes(:, 1);

% number of rows in [nodes] is equal to the greatest node number in
% [tempNodes]
nodes = zeros(length(maxNode), 3);

% [nodes] is structured such that the row index represents the node number
nodes(originalNodeNums, 1:3) = tempNodes(:, 2:4);

clear tempNodes % free memory

%%
% Create a (. x 4) array [elemNodes] containing nodes associated with
```

```
% each element in the desired part.

% store all element lines
remainderElemNodes...
   = parse_numbers(elemStart, elemEnd, fileData{1}, 1, 10);

% shift node numbers
remainderElemNodes = remainderElemNodes - minNode + 1;

% max element number
maxElem = max(remainderElemNodes(:, 1));

% only keep nodes of elements belonging to the user-inputted part number
elemNodes = remainderElemNodes(remainderElemNodes(:, 2) == part, :);

% [remainderElemNodes] contains the elemets not belonging to the
% user-inputted part number
remainderElemNodes = setdiff(remainderElemNodes, elemNodes,...
   'rows', 'stable');

% only care about the first four node entries of the tet elements
elemNodes = elemNodes(:, 3:6);

lenElem = length(elemNodes(:, 1)); % length of [elemNodes]

%%
% From [elemNodes], create a (. x 2) array [lines], where each row
% contains pairs of nodes that are connected with each other. The columns
% of the matrix formed in the reshape command are chosen such that each
% column of [elemNodes] pairs with each other column of [elemNodes] once
% [COL1, COL3;
%  COL2, COL4;
%  COL3, COL2;
%  COL4, COL3;
%  COL1, COL4;
%  COL2, COL1].

lines = reshape([elemNodes, elemNodes, elemNodes(:, 2:4),...
   elemNodes(:, 1)], 6*lenElem, 2);

% Sort each row of [lines] in ascending order (i.e., left entry is less
% than right entry).
lines = sort(lines, 2);

%%
% Create a matrix [uniqueLines] that contains only the unique rows of
% [lines]. The matrix [lines] can be recreated by setting lines =
% uniqueLines(uniqueLinesIndex, :). In other words, the ith row of
% [lines] is the row in [uniqueLines] that equals the ith element of
% [uniqueLinesIndex].

[uniqueLines, ~, uniqueLinesIndex] = unique(lines, 'rows', 'stable');
```

```
clear lines

% Append a column to [uniqueLines] whose ith element contains the node
% number corresponding to the new node that will bisect the pair of nodes
% in the ith row of [uniqueLines].
uniqueLines = [uniqueLines,...
   (maxNode+1 : 1 : length(uniqueLines) + maxNode)'];

% current max node number
maxNode = uniqueLines(end);

% Reshape [uniqueLinesIndex] so that the ith element of the first column
% corresponds to the index of [uniqueLines] that gives the 13 node (see
% APPENDIX A; first row of [lines] connects the 1 and 3 nodes) of the ith
% original element in its 3rd column. The ith element of the second column
% corresponds to the index of [uniqueLines] that gives the 24 node (see
% APPENDIX A; second row of [lines] connects the 2 and 4 nodes) of the ith
% original element, etc.
uniqueLinesIndex = reshape(uniqueLinesIndex, lenElem, 6);

%%
% From [elemNodes], create a (. x 3) array [faces], where each row
% contains triplets of nodes that form faces of tets. The columns of the
% matrix formed in the reshape command are chosen such that the combination
% of columns of elemNodes occurs once
% [COL1, COL2, COL3;
%  COL2, COL3, COL4;
%  COL3, COL4, COL1;
%  COL4, COL2, COL1].

faces = reshape([elemNodes, elemNodes(:, 2:4), elemNodes(:, 2:4),...
   elemNodes(:, 1), elemNodes(:, 1)], 4*lenElem, 3);

% Sort each row of [faces] in ascending order (i.e., left entry is less
% than right entry).
faces = sort(faces, 2);

%%
% Create a matrix [uniqueFaces] that contains only the unique rows of
% [faces]. The matrix [faces] can be recreated by setting faces =
% uniqueFaces(uniqueFacesIndex, :). In other words, the ith row of
% [faces] is the row in [uniqueFaces] that equals the ith element of
% [uniqueFacesIndex].

[uniqueFaces, ~, uniqueFacesIndex] = unique(faces, 'rows', 'stable');

clear faces

% Append a column to [uniqueFaces] whose ith element contains the node
% number corresponding to the new node that will be coincident with the
% centroid of the triplet of nodes in the ith row of [uniqueFaces].

uniqueFaces = [uniqueFaces, (maxNode+1 : 1 : length(uniqueFaces) + maxNode)'];
```

```
maxNode = uniqueFaces(end); % update maxNode

% Reshape [uniqueFacesIndex] so that the ith element of the first column
% corresponds to the row of [uniqueFaces] that gives the 123 node (see
% APPENDIX A; first row of [faces] lies on the 1-2-3 face) of the ith
% original element in its 4th column. The ith element of the second column
% corresponds to the index of [uniqueFaces] that gives the 234 node (see
% APPENDIX A; second row of [faces] lies on the 2-3-4 face) of the ith
% original element, etc.
uniqueFacesIndex = reshape(uniqueFacesIndex, lenElem, 4);

%%
% Append a column to [elemNodes] whose ith element contains the node
% number corresponding to the new node that will be coincident with the
% volumetric centroid of the quadruplet of nodes in the ith row of
% [elemNodes].

elemNodes = [elemNodes, (maxNode+1 : 1 : length(elemNodes) + maxNode)'];

maxNode = lenElem + maxNode; % update maxNode

%%
% Generate new nodes

% Append zeroes to [nodes] to allow for new nodes to be written
nodes = [nodes; zeros(maxNode - length(nodes(:, 1)), 3)];

% Use the node numbers in the third column of [uniqueLines] to specify the
% new node numbers to be created in [nodes]. The values of these new
% coordinates are the average values of the coordinates of the nodes in the
% first and second columns of [uniqueLines].
nodes(uniqueLines(:, 3), :)...
   = (nodes(uniqueLines(:, 1), :) + nodes(uniqueLines(:, 2), :))/2;

% Use the node numbers in the fourth column of [uniqueFaces] to specify
% the new node numbers to be created in [nodes]. The values of these new
% coordinates are the average values of the coordinates of the nodes in the
% first, second, and third columns of [uniqueFaces].
nodes(uniqueFaces(:, 4), :) = (nodes(uniqueFaces(:, 1), :)...
   + nodes(uniqueFaces(:, 2), :) + nodes(uniqueFaces(:, 3), :))/3;

% Use the node numbers in the fifth column of [elemNodes] to specify
% the new node numbers to be created in [nodes]. The values of these new
% coordinates are the average values of the coordinates of the nodes in the
% first, second, third, and fourth columns of [elemNodes].
nodes(elemNodes(:, 5), :) = (nodes(elemNodes(:, 1), :)...
   + nodes(elemNodes(:, 2), :) + nodes(elemNodes(:, 3), :)...
   + nodes(elemNodes(:, 4), :))/4;

%%
% Create new elements. First cell represents the first element created from
% the quadruplet of elements in [elemNodes]; i.e., the element that
```

```
% contains the node in column 1 of [elemNodes]. Etc. for the other cells.
newElem = cell(1, 4);

% Initialize cells
for j = 1:4
   newElem{j} = zeros(lenElem, 8);
end

% see APPENDIX A for node descriptions
node12 = uniqueLines(uniqueLinesIndex(:, 6), 3);
node13 = uniqueLines(uniqueLinesIndex(:, 1), 3);
node14 = uniqueLines(uniqueLinesIndex(:, 5), 3);
node23 = uniqueLines(uniqueLinesIndex(:, 3), 3);
node24 = uniqueLines(uniqueLinesIndex(:, 2), 3);
node34 = uniqueLines(uniqueLinesIndex(:, 4), 3);
node123 = uniqueFaces(uniqueFacesIndex(:, 1), 4);
node124 = uniqueFaces(uniqueFacesIndex(:, 4), 4);
node134 = uniqueFaces(uniqueFacesIndex(:, 3), 4);
node234 = uniqueFaces(uniqueFacesIndex(:, 2), 4);
nodeC = elemNodes(:, 5);

% EL1
newElem{1}(:, 1) = elemNodes(:, 1); % first node is node 1
newElem{1}(:, 2) = node12; % second node is node 12
newElem{1}(:, 3) = node123; % third node is node 123
newElem{1}(:, 4) = node13; % fourth node is node 13
newElem{1}(:, 5) = node14; % fifth node is node 14
newElem{1}(:, 6) = node124; % sixth node is node 124
newElem{1}(:, 7) = nodeC; % seventh node is Center
newElem{1}(:, 8) = node134; % eighth node is node 134

% EL2
newElem{2}(:, 1) = elemNodes(:, 2); % first node is node 2
newElem{2}(:, 2) = node23; % second node is node 23
newElem{2}(:, 3) = node123; % third node is node 123
newElem{2}(:, 4) = node12; % fourth node is node 12
newElem{2}(:, 5) = node24; % fifth node is node 24
newElem{2}(:, 6) = node234; % sixth node is node 234
newElem{2}(:, 7) = nodeC; % seventh node is Center
newElem{2}(:, 8) = node124; % eighth node is node 124

% EL3
newElem{3}(:, 1) = elemNodes(:, 3); % first node is node 3
newElem{3}(:, 2) = node34; % second node is node 34
newElem{3}(:, 3) = node134; % third node is node 134
newElem{3}(:, 4) = node13; % fourth node is node 13
newElem{3}(:, 5) = node23; % fifth node is node 23
newElem{3}(:, 6) = node234; % sixth node is node 234
newElem{3}(:, 7) = nodeC; % seventh node is Center
newElem{3}(:, 8) = node123; % eighth node is node 123

% EL4
newElem{4}(:, 1) = elemNodes(:, 4); % first node is node 4
```

```
newElem{4}(:, 2) = node14; % second node is node 14
newElem{4}(:, 3) = node134; % third node is node 134
newElem{4}(:, 4) = node34; % fourth node is node 34
newElem{4}(:, 5) = node24; % fifth node is node 24
newElem{4}(:, 6) = node124; % sixth node is node 124
newElem{4}(:, 7) = nodeC; % seventh node is Center
newElem{4}(:, 8) = node234; % eighth node is node 234

clear elemNodes

% form matrix of new elements
newElem = [newElem{1}; newElem{2}; newElem{3}; newElem{4}];
numNewElem = length(newElem(:, 1));

%%
% Find new nodes that are on the symmetry plane.
if ~isempty(userInput{3})
   % perpendicular distance from symmetry plane
   d = abs(xn*(nodes(maxNodeOriginal+1:maxNode, 1) - x0)...
      + yn*(nodes(maxNodeOriginal+1:maxNode, 2) - y0)...
      + zn*(nodes(maxNodeOriginal+1:maxNode, 3) - z0));

   % [symNodes] contains nodes that are within [tol] of the symmetry
   % plane
   symNodes = find(d <= tol);
end

%%
% Create and open new file to write hexoganalized .k file
fileID = fopen(char(strcat(fileName, '_hex', '.k')), 'w');

% print all lines up to the first *ELEMENT or *NODE card
if elemStart(1) < nodeStart(1)
   nextCard = elemStart(1);
   lastCard = elemEnd(1);
   elemCount = 2;
   nodeCount = 1;
else
   nextCard = nodeStart(1);
   lastCard = nodeEnd(1);
   elemCount = 1;
   nodeCount = 2;
end
fprintf(fileID, '%s\n', fileData{1}{1:nextCard-1});

% print all lines except *ELEMENT and *NODE cards
numElemCards = length(elemStart);
numNodeCards = length(nodeStart);
while elemCount <= numElemCards && nodeCount <= numNodeCards
   if elemStart(elemCount) < nodeStart(nodeCount)
      nextCard = elemStart(elemCount);
      lastCard = elemEnd(elemCount);
      elemCount = elemCount + 1;
```

```
    else
        nextCard = nodeStart(nodeCount);
        lastCard = nodeEnd(nodeCount);
        nodeCount = nodeCount + 1;
    end
    fprintf(fileID, '%s\n', fileData{1}{lastCard+1:nextCard-1});
end
if nodeCount > numNodeCards
    for k = elemCount:numElemCards-1
        fprintf(fileID, '%s\n', fileData{1}{elemEnd(k)+1:elemStart(k+1)-1});
    end
else
    for k = nodeCount:numNodeCards-1
        fprintf(fileID, '%s\n', fileData{1}{nodeEnd(k)+1:nodeStart(k+1)-1});
    end
end

% print all lines after the last *ELEMENT card
fprintf(fileID, '%s\n', fileData{1}{elemEnd(end)+1:end-1});

% Print cards for elements that were not affected. NOTE THAT THIS ASSUMES
% ALL ELEMENTS IN MODEL ARE *ELEMENT_SOLID. IF OTHER TYPES ARE USED, NEED
% TO MANUALLY CHANGE IN .K FILE.
if ~isempty(remainderElemNodes)
    fprintf(fileID, '*ELEMENT_SOLID\n');
    fprintf(fileID, ['$   1EID    2PID      N1      N2      N3      ',...
        'N4      N5      N6      N7      N8\n']);
    fprintf(fileID, '%8d%8d%8d%8d%8d%8d%8d%8d%8d%8d\n',...
        remainderElemNodes');
end

% print new *NODE cards
fprintf(fileID, '*NODE\n');
fprintf(fileID, ['$   1NID              2X              3Y              ',...
    '4Z     5TC     6RC\n']);
fprintf(fileID, '%8d%16.8e%16.8e%16.8e\n',...
    [(1:maxNode)', nodes]');

% print new *ELEMENT cards
fprintf(fileID, '*ELEMENT_SOLID\n');
fprintf(fileID, ['$   1EID    2PID      N1      N2      N3      N4      ',...
    'N5      N6      N7      N8\n']);
fprintf(fileID, '%8d%8d%8d%8d%8d%8d%8d%8d%8d%8d\n',...
    [(maxElem+1 : 1 : maxElem+numNewElem)',...
    repmat(part, numNewElem, 1), newElem]');

% print new *SET_NODE card
if ~isempty(userInput{3})
    fprintf(fileID, '*SET_NODE_LIST\n');
    fprintf(fileID, '$     1SID      2DA1      3DA2      4DA3      5DA4\n');
    fprintf(fileID, '%10d\n', setNum);
    fprintf(fileID, ['$    1NID1     2NID2     3NID3     4NID4     ',...
        '5NID5     6NID6     7NID7     8NID8\n']);
```

```
    fprintf(fileID, '%10d%10d%10d%10d%10d%10d%10d%10d\n',...
        maxNodeOriginal+symNodes(1:end-1));
    fprintf(fileID, '%10d\n', maxNodeOriginal+symNodes(end));
end

fprintf(fileID, '*END');

fclose(fileID);

% warning messages
msgbox(['Warning: If non-*ELEMENT_SOLID element cards were in the original ',...
    'model, the card headers will need to be manually inserted into the .k file.']);
msgbox('Warning: Change the *SECTION card of the part converted from tet to hex.');
if ~isempty(userInput{3})
    msgbox({strcat(['Warning: A boundary condition card must be written ',...
        'for *SET_NODE #'], num2str(setNum), '.')});
end



% This function parses lines of a .k file into floating point numbers
% 'card start' is a vector containing the starting rows of the cards of
% interest
% 'card end' is a vector containing the ending rows of the cards of
% interest
% 'file_data' is a vector of strings where each element of the vector is a
% line of the .k file
% 'start_col' is the starting column to parse numbers
% 'end_col' is the ending column to parse numbers
function ret = parse_numbers(card_start, card_end, file_data, start_col, end_col)

% Allocate temporary array assuming all lines of the cards are meaningful
% (i.e., not comments)
temp_array = zeros(sum(card_end - card_start + 1), end_col - start_col + 1);
index = 1; % index to move along the rows of temp_array

for i = 1:length(card_start) % repeat this for each individual card card

    row = card_start(i) + 1;

    % Skip over dollar signs (comments).
    while strcmp(file_data{row}(1:1), '$')
        row = row + 1;
    end

    % until you reach the end of the card, do the below
    while row <= card_end(i)

        % parse line and assign to temp_array
        temp_string = textscan(file_data{row},'%f');
        temp_array(index, :) = temp_string{1}(start_col:end_col)';

        index = index + 1;
```

```
      row = row + 1;

      % Skip over dollar signs (comments).
      while strcmp(file_data{row}(1:1), '$')
         row = row + 1;
      end

   end
end

% return only the rows of temp_array used
ret = temp_array(1:(index - 1), :);

end



% This function finds the start and end lines of cards in .k files
% 'file_data' is a vector of strings where each element of the vector is a
% line of the .k file
% 'card_name' is a string that contains the name of the card you wish to
% find the stat and end lines for
function [start_cards, end_cards] = card_bookends(file_data, card_name)

% find start of cards
start_cards = find(strncmp(file_data, card_name, length(card_name)));
num_cards = length(start_cards); % number of cards

% Allocate array containing end rows of each card
end_cards = zeros(num_cards, 1);

% Populate array containing end rows of each card
for i = 1:num_cards

   row = start_cards(i) + 1;

   % Search until the next card is found
   while strcmp(file_data{row}(1:1), '*') == 0
      row = row + 1;
   end

   end_cards(i) = row - 1;

end

end
```

DISTRIBUTION LIST

U.S. Army DEVCOM AC
ATTN: FCDD-ACE-K
FCDD-ACM-MI, K. Hohnecker
Picatinny Arsenal, NJ 07806-5000

Defense Technical Information Center (DTIC)
ATTN: Accessions Division
8725 John J. Kingman Road, Ste. 0944
Fort Belvoir, VA 22060-6218

GIDEP Operations Center
P.O. Box 8000
Corona, CA 91718-8000
gidep@gidep.org

REVIEW AND APPROVAL OF ARDEC TECHNICAL REPORTS

MATLAB Script to Convert Tetrahedrals to Hexahedrals in LS-DYNA
Title

_____
Date received by LCSD

___Ken Hohnecker_____
Author/Project Engineer

_____
Report number (to be assigned by LCSD)

_x7633_____ 65N_____ ____FCDD-ACM-MI_____
Extension Building Author's/Project Engineers Office
(Division, Laboratory, Symbol)

PART 1. **Must be signed before the report can be edited.**

a. The draft copy of this report has been reviewed for technical accuracy and is approved for editing.

b. Use Distribution Statement A__X__, B____, C____, D____, E____, F____ or X ____ for the reason checked on the continuation of this form.

1. If Statement A is selected, the report will be released to the National Technical Information Service (NTIS) for sale to the general public. Only unclassified reports whose distribution is not limited or controlled in any way are released to NTIS.

2. If Statement B, C, D, E, F, or X is selected, the report will be released to the Defense Technical Information Center (DTIC) which will limit distribution according to the conditions indicated in the statement.

c. The distribution list for this report has been reviewed for accuracy and completeness.

4/29/2020

X Mark Nicolich

Mark Nicolich
Competency Manager, Small Caliber Munito...

_____
Division Chief (Date)

PART 2. To be signed either when draft report is submitted or after review of reproduction copy.

This report is approved for publication.

Mark Nicolich 6/8/22

_____
Division Chief (Date)