LA-UR-20-27139
Approved for public release; distribution is unlimited.

Title: Generating MCNP Input Files for Unstructured Mesh Geometries

Author(s): Armstrong, Jerawan Chudoung
Kelley, Karen Corzine

Intended for: Report

Issued: 2020-09-14

Generating MCNP Input Files for Unstructured Mesh Geometries

Jerawan Armstrong and Karen Kelley

September 2020

1 Introduction

Los Alamos National Laboratory's (LANL) Monte Carlo N-Particle (MCNP)[1] transport code version 6 has the capability for tracking particles on unstructured mesh (UM) geometry models [1–4]. The MCNP UM feature has been developed for performing calculations of complex geometry models. This capability tracks particles on hybrid geometries where finite element meshes are embedded into constructive solid geometry (CSG) cells. The MCNP UM feature was originally designed to read UM models created by Abaqus/CAE software suite [5]. MCNP versions 6.2.0 and later can process UM models read from Abaqus input files or MCNPUM files converted from Abaqus input files. Sandia National Laboratory's Cubit Toolkit [6] and other finite element analysis software packages may generate UM models and then convert these models into Abaqus input formats. A high-level workflow of MCNP UM calculation is shown in Figure 1.

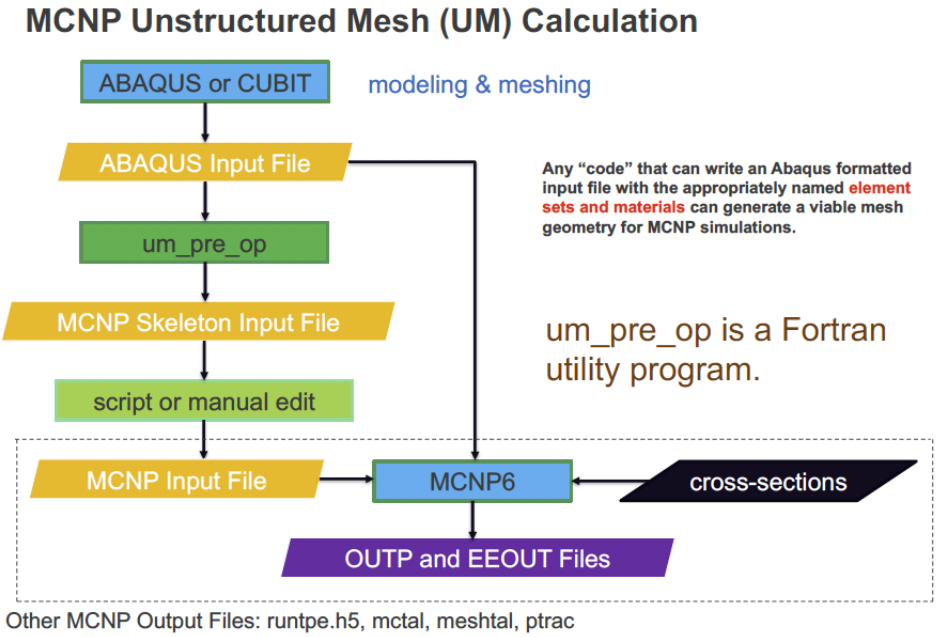

Figure 1. MCNP unstructured mesh calculation workflow

The first step of the MCNP UM calculation is creating an unstructured mesh model where each element set (elset) and material definition in the Abaqus input file is named in a specific format [3]. MCNP and Abaqus input files required for MCNP UM simulations must be related, i.e. pseudo-cells in the MCNP input file must be constructed from mesh information defined in the Abaqus input file. The *um_pre_op* (unstructured mesh pre operations) program with the `-m` option can be used to create

[1]MCNP® and Monte Carlo N-Particle® are registered trademarks owned by Triad National Security, LLC, manager and operator of Los Alamos National Laboratory for the U.S. Department of Energy under contract number 89233218CNA000001. Any third party use of such registered marks should be properly attributed to Triad National Security, LLC, including the use of the ®designation as appropriate. Any questions regarding licensing, proper use, and/or proper attribution of Triad National Security, LLC marks should be directed to trademark@lanl.gov. For the purposes of visual clarity, the registered trademark symbol is assumed for all references to MCNP within the remainder of this report.

a skeleton MCNP input file from an Abaqus input file [3]. The *um_pre_op* program was written in Fortran and was not written for performance optimization. When using *um_pre_op*, the `--datacards` option may be used to add the data cards such as source, tallies, and material cards to the MCNP skeleton input file to be a functional MCNP input file, but additional CSG cell and surface cards for hybrid geometry (i.e., mixed CSG and UM geometry) calculations have to be manually added. To improve the workflow of multiphysics calculations, a Python3 code called *write_mcnp_um_input* has been developed to generate the MCNP input file instead of using the *um_pre_op* `-m` option [7].

2 Generating an MCNP6 Input File

The intent of the *write_mcnp_um_input.py* code shown in Appendix A is to make it easier for users to set up MCNP UM input files. This program can read and process the Abaqus input file to create the appropriate pseudo-cell cards, background cell, minimal CSG world to hold the mesh universe, and the `embed` control card for the data section. A new algorithm is implemented in the *write_mcnp_um_input.py* code to create MCNP pseudo-cells. In addition, the *write_mcnp_um_input.py* code performs extensive error checking before generating the MCNP input file since Python is well suited for file parsing. The following is the signature of *write_mcnp_input* method:

```
def write_mcnp_um_input(filein, fileout=None, eeout=None, meshinfo=None,
                        cellcards=None, surfacecards=None, datacards=None,
                        bgmaterial=0, lenconv=1.0, radiusext=1.0, denunit='g/cc',
                        writemeshinfo=False, writecomments=False):
```

The command line *python write_mcnp_um_input.py –help* can be used to see the command line options:

```
usage: write_mcnp_um_input.py [-h] -i <file.inp> [-o <file.mcnp>]
                              [-e <file.eeout>] [-mi <filename.info>]
                              [-cc <cellcards.txt>] [-sc <surfacecards.txt>]
                              [-dc <datacards.txt>] [-b <material_number>]
                              [-l <len_conversion>] [-re <radius_extension>]
                              [-du <density_unit>] [-wc] [-wm]

** Write MCNP Unstructured Mesh Input file **

optional arguments:
  -h, --help            show this help message and exit
  -i <file.inp>, --input <file.inp>
                        Abaqus input for constructing MCNP input file
  -o <file.mcnp>, --output <file.mcnp>
                        output file name
  -e <file.eeout>, --eeout <file.eeout>
                        eeout file name in EMBED card
  -mi <filename.info>, --meshinfo <filename.info>
                        a mesh information file name
  -cc <cellcards.txt>, --cellcards <cellcards.txt>
                        MCNP cell cards file to include
  -sc <surfacecards.txt>, --surfacecards <surfacecards.txt>
                        MCNP surface cards file to include
  -dc <datacards.txt>, --datacards <datacards.txt>
                        MCNP data cards file to include
  -b <material_number>, --back <material_number>
                        background material for MCNP input file
  -l <len_conversion>, --length <len_conversion>
                        a multiplication conversion factor to centimeters
  -re <radius_extension>, --radext <radius_extension>
                        a radius extension of a sphere surface for a fill cell
                        (in centimeters)
  -du <density_unit>, --densityunit <density_unit>
                        density unit option: g/cm^3 [default] or atoms/barn-cm
  -wc, --writecomments  write pseudo-cell comments in MCNP input file
```

```
  -wm, --writemeshinfo  write mesh information into mesh information file
```

The required `--input` option is for entering the Abaqus input filename. The other arguments are optional. The name of the MCNP UM input file to be created can be set with the `--output`. The name of the EEOUT file on the EMBED card can be set with the `--eeout`. The name of the mesh information file can be set with the `--meshinfo`. The default filenames are `filename.out`, `filename.eeout`, and `filename.info`, respectively, where the filename is the basename of the Abaqus input file.

The degree to which a fully functional MCNP input file can be generated depends upon the completeness and correctness of the cell, surface, and data card files provided with the `--cellcards`, `--surfacecards`, and `--datacards` options. The users must make sure that these files have the proper MCNP syntax. The *write_mcnp_um_input* routine will include these files after the fill cell line in the cell card section, the surface used to define a fill cell in the surface card section, and the embed card in the data card section.

If the `--background` option is not specified, *write_mcnp_um_input* will make the background cell void. If the `--length` option is not specified, *write_mcnp_um_input* will use a length multiplier equal to 1.0. The surface of the mesh universe fill cell is spherical with its center and radius computed from nodal data read from the Abaqus input file. The parameter values defined the spherical surface are computed by the following Python code:

```
x = xmax - xmin
y = ymax - ymin
z = zmax - zmin
diagonal = np.sqrt(x*x+y*y+z*z)
center   = 0.5*np.array([xmin+xmax, ymin+ymax, zmin+zmax])
radius   = 0.5*diagonal + radiusext
```

The `radiusext` value can be specified by the `--radext` option. If this option is not specified then the `radiusext` is set to 1.0 cm. The `radiusext` value is needed so that the UM model is not clipped by the boundary of the fill cell. The `--densityunit` option can be used to specify the density unit and the default value is g/cm^3 (i.e., mass density).

The `--writecomments` option is for writing cell comments in an MCNP input file. These cell comments provide matching information between Abaqus instances, parts, and elsets and MCNP psuedo-cells. The `--writemeshinfo` option writes mesh information into the mesh information file. This file contains the matching information between mesh information and MCNP pseudo-cells as well as the node, element, elset, instance, and material data read from the Abaqus input file.

3 Testing

The Abaqus input file listed in Appendix B was used to test the *write_mcnp_input* code on ordering of pseudocells for discontiguous element sets, multiple element sets per part, and multiple instances of the same part. There is also additional information in the file (material section assignments) to verify that the code would ignore information not needed by the MCNP code.

The command *python write_mcnp_um_input -i block_hemi_v6.inp -wc* was used to form the following skeleton input file, `block_hemi_v6.mcnp` with pseudocell comments:

```
No description
C
C Abaqus Input File: block_hemi_v6.inp
C
C High-Level Mesh Information:
C   Number of Parts:     4
C   Number of Instances: 5
C   Number of Materials: 2
C
C Details of mesh information are in file: block_hemi_v6.info
C
C PSEUDO CELLS
C
C 1: HEMI-A-1, HEMI-A, statistic-11
```

```
1       1008   -1.00000      0  u=1  $ WATER_1008
C
C 2: HEMI-A-1, HEMI-A, statistic-12
2          1   -2.70000      0  u=1  $ ALUMINUM_001
C
C 3: HEMI-A-1, HEMI-A, statistic-23
3       1008   -1.00000      0  u=1  $ WATER_1008
C
C 4: HEMI-A-1, HEMI-A, statistic-34
4       1008   -1.00000      0  u=1  $ WATER_1008
C
C 5: BLOCK-B-1, BLOCK-B, statistic-1
5          1   -2.70000      0  u=1  $ ALUMINUM_001
C
C 6: BLOCK-B-1, BLOCK-B, statistic-4
6          1   -2.70000      0  u=1  $ ALUMINUM_001
C
C 7: HEX_WEDGE1-2, HEX_WEDGE1, statistic-2
7          1   -2.70000      0  u=1  $ ALUMINUM_001
C
C 8: TET_ONLY-1, TET_ONLY, statistic-1
8          1   -2.70000      0  u=1  $ ALUMINUM_001
C
C 9: HEX_WEDGE1-1, HEX_WEDGE1, statistic-2
9          1   -2.70000      0  u=1  $ ALUMINUM_001
C
10          0                0  u=1  $ background
C
C LEGACY CELLS
11          0               -999  fill=1 $ fill cell
12          0                999

C Surface Cards
C
999 sph    -7.50000E+00     1.29371E+01    -5.33512E+00     2.76519E+01

C Data Cards
C
embed1 meshgeo=abaqus
       mgeoin=block_hemi_v6.inp
       meeout=block_hemi_v6.eeout
       length=1.000000E+00
       background=10
       matcell=1 1  $ WATER_1008
               2 2  $ ALUMINUM_001
               3 3  $ WATER_1008
               4 4  $ WATER_1008
               5 5  $ ALUMINUM_001
               6 6  $ ALUMINUM_001
               7 7  $ ALUMINUM_001
               8 8  $ ALUMINUM_001
               9 9  $ ALUMINUM_001
```

References

[1] "MCNP." https://mcnp.lanl.gov.

[2] C. J. Werner, J. Armstrong, F. B. Brown, J. S. Bull, L. Casswell, L. J. Cox, D. Dixon, R. A. Forster, J. T. Goorley, H. G. Hughes, J. Favorite, R. Martz, S. G. Mashnik, M. E. Rising, C. J.

Solomon, A. Sood, J. E. Sweezy, A. Zukaitis, C. Anderson, J. S. Elson, J. W. Durkee, R. C. Johns, G. W. McKinney, G. E. McMath, J. S. Hendricks, D. B. Pelowitz, R. E. Prael, T. E. Booth, M. R. James, M. L. Fensin, T. A. Wilcox, and B. C. Kiedrowski, "MCNP User's Manual, Code Version 6.2," Tech. Rep. LA-UR-17-29981, Los Alamos National Laboratory, Los Alamos, NM, USA, Oct. 2017.

[3] R. L. Martz, "Unstructured Mesh User's Startup Guide," Tech. Rep. LA-UR-12-00795, Los Alamos National Laboratory, Los Alamos, NM, USA, Feb. 2012.

[4] R. L. Martz and D. L. Crane, "The MCNP6 Book on Unstructured Mesh Geometry: Foundations," Tech. Rep. LA-UR-12-25478, Los Alamos National Laboratory, Los Alamos, NM, USA, Nov. 2012.

[5] "Abaqus/CAE." `www.3ds.com/simulia`.

[6] "CUBIT." `https://cubit.sandia.gov`.

[7] J. Armstrong and M. Rising, "MCNP Code Modernization & Code Improvement for Microreactor Multiphysics Calculations." LA-UR-20-23568, May 2020. Los Alamos National Laboratory.

Appendix A Python Codes

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
  Code:        write_mcnp_um_input.py

  Author:      Jerawan Armstrong
               Monte Carlo Codes (XCP-3) Group
               X Computational Physics (XCP) Division
               Los Alamos National Laboratory

  Copyright (c) 2020 Triad National Security, LLC.  All rights reserved.

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in all
  copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
  SOFTWARE.
"""
import numpy as np
import os

from abaqus_input import read_abaqus
from abaqus_input import compute_global_model_extents
from abaqus_input import extract_mesh_data

from abq_part_elsets import  compute_part_elset_mat_nums
from abq_part_elsets import  compute_part_elset_data
from abq_part_elsets import  compute_part_elset_material_data

from abq_part_data_checking import check_material_elset_data

#-----------------------------------------------------------------------
def write_mcnp_um_input(filein, fileout=None, eeout=None, meshinfo=None,
                        cellcards=None, surfacecards=None, datacards=None,
                        bgmaterial=0, lenconv=1.0, radiusext=1.0, denunit='g/cc',
                        writemeshinfo=False, writecomments=False):

    print("\n***** Write MCNP skeleton input file *****")

    invalid_input = 0

    if not os.path.isfile(filein):
        print("\nfile does not exist: {:>s}\n".format(filein))
        invalid_input = 1

    if filein.lower() != filein:
        print('\nAn Abaqus input filename does not in lowercase: {:s}'.format(filein))
        print("MCNP converts filenames on EMBED cards into lowercase letters.")
        print("Abaqus input filename should be in lowercase letters to prevent")
        print("an error when running MCNP.\n")
        invalid_input = 2

    include_cell_cards = False
    if cellcards is not None:
        if not os.path.isfile(cellcards):
            print('\nThe cell card file does not exist: {:s}'.format(cellcards))
            invalid_input = 4
        else:
            include_cell_cards = True

    include_surface_cards = False
    if surfacecards is not None:
        if not os.path.isfile(surfacecards):
            print('\nThe surface card file does not exist: {:s}'.format(surfacecards))
            invalid_input = 5
        else:
            include_surface_cards = True
```

```
include_data_cards = False
if datacards is not None:
    if not os.path.isfile(datacards):
        print('\nThe data card file does not exist: {:s}'.format(datacards))
        invalid_input = 6
    else:
        include_data_cards = True

if int(bgmaterial) < 0:
    print(">>>Error: invalid background material number; it must be non-negative")
    print("          input value = {:d}".format(bgmaterial))
    invalid_input = -1

if lenconv <= 0.0:
    print(">>>Error: invalid length conversion; it must be positive")
    print("          input value = {:f}".format(lenconv))
    invalid_input =  -2

if radiusext <= 0.0:
    print(">>>Error: invalid radius extension; it must be positive")
    print("          input value = {:f}".format(radiusext))
    invalid_input =  -3

if denunit.lower() not in ['g/cc', 'g/cm^3', 'g/cm**3', 'atoms/barn-cm']:
    print(">>>Error: invalid density unit; denunit must be g/cm^3 or atoms/barn-cm")
    print("          input value = {:f}".format(radiusext))
    invalid_input =  -3
else:
    if denunit.lower() in ['atoms/barn-cm','atoms/b-cm']:
        dentype = 'atom_den'
    else:
        dentype = 'mass_den'

if invalid_input != 0: return None


name = os.path.basename(filein)
name = name.split('.')

if fileout is None: fileout = name[0]+'.mcnp'
if os.path.isfile(fileout):
    print('\nThe MCNP input file exist: {:s}'.format(fileout))
    print('Manaully remove this file or give the new MCNP input file\n')
    invalid_input = 100

if eeout is None:
    eeout = name[0]+'.eeout'
else:
    eeout_lower = eeout.lower()
    if eeout != eeout_lower:
        print("\nUser's EEOUT filename: {:s}".format(eeout))
        print("EEOUT filename is changed to: {:s}".format(eeout_lower))
        eeout = eeout_lower

if meshinfo is None:
    meshinfo = name[0]+'.info'
if os.path.isfile(meshinfo):
    print('\nThe mesh information file exist: {:s}'.format(meshinfo))
    print('Manaully remove this file or give the new mesh information file\n')
    invalid_input = 101

if invalid_input != 0: return None


#------------- Read Abaqus Input File ---------------
names, inpdata, inplines = read_abaqus(filein)

heading       = inpdata.get_heading()
part_data     = inpdata.get_parts()
instance_data = inpdata.get_instances()
material_data = inpdata.get_materials()

part_lines     = inplines.get_parts()
instance_lines = inplines.get_instances()

part_names      = names[0]
instance_names  = names[1]
material_names  = names[2]

print("   Number of parts:     {:d}".format(len(part_names)))
print("   Number of instances: {:d}".format(len(instance_names)))
print("   Number of materials: {:d}\n".format(len(material_names)))

if writemeshinfo:
    mesh_info = extract_mesh_data(names, inpdata, inplines, dentype)
del names, inpdata, inplines


#------- Process Abaqus Input File -------------
print("Processing Abaqus input file")
extents = compute_global_model_extents(part_names, part_data, part_lines,
                                       instance_names, instance_data, instance_lines)
extents = np.array(extents)*lenconv

xmin = extents[0]
xmax = extents[1]
ymin = extents[2]
ymax = extents[3]
zmin = extents[4]
zmax = extents[5]

# In um_pre_op fortran version, radius is hard coded as 0.6*diagonal.
x = xmax - xmin
y = ymax - ymin
z = zmax - zmin
diagonal = np.sqrt(x*x+y*y+z*z)
center   = 0.5*np.array([xmin+xmax, ymin+ymax, zmin+zmax])
radius   = 0.5*diagonal + radiusext

print("\n   Global Model Extents (in cm)")
print("      Min X: {:20.5E}; Max X: {:20.5E}".format(xmin, xmax))
print("      Min Y: {:20.5E}; Max Y: {:20.5E}".format(ymin, ymax))
print("      Min Z: {:20.5E}; Max Z: {:20.5E}\n".format(zmin, zmax))

text_info = "Abaqus Input File: {}".format(filein)

text_info = text_info + "\n\nHigh-Level Mesh Information:"
text_info = text_info + "\n   Number of Parts:     {:d}".format(len(part_names))
```

```
text_info = text_info + "\n   Number of Instances: {:d}".format(len(instance_names))
text_info = text_info + "\n   Number of Materials: {:d}".format(len(material_names))

text_info = text_info + "\n\nGlobal Model Extents (in cm)"
text_info = text_info + "\n   Min X: {:20.5E}; Max X: {:20.5E}".format(xmin, xmax)
text_info = text_info + "\n   Min Y: {:20.5E}; Max Y: {:20.5E}".format(ymin, ymax)
text_info = text_info + "\n   Min Z: {:20.5E}; Max Z: {:20.5E}\n".format(zmin, zmax)


print("Creating MCNP pseudo-cells")

part_elset_mat_nums = compute_part_elset_mat_nums(part_names, part_data, material_data)
part_elset_data     = compute_part_elset_data(part_names, part_data)
check_material_elset_data(part_names,part_data,material_data)

# set material elset data
part_elset_material_data = compute_part_elset_material_data(part_names,
                                                             part_lines,
                                                             part_elset_data,
                                                             material_data)

text_info = text_info + "\n\n{:>10s}".format("cell #")
text_info = text_info + " {:>10s} {:>20s} {:>50s}".format("mat #", "density", "mat_name")
text_info = text_info + " {:>50s} {:>50s} {:>50s}".format('elset_name', 'part_name', 'instance_name')
text_info = text_info + "\n"
if dentype == 'mass_den':
    text_info = text_info + " {:>20s} {:>20s}".format("", "(g/cm^3)")
elif dentype == "atom_den":
     text_info = text_info + " {:>20s} {:>20s}".format("", "(atoms/barn-cm)")
else:
    raise ValueError('>>>Error: invalid density unit')

if len(heading) > 0:
    if len(heading) <= 128:
        text = heading
    else:
        text = heading[0:127]
else:
    text = "No description"
text = text + "\nC"
text = text + "\nC Abaqus Input File: {}".format(filein)
text = text + "\nC"
text = text + "\nC High-Level Mesh Information:"
text = text + "\nC   Number of Parts:     {:d}".format(len(part_names))
text = text + "\nC   Number of Instances: {:d}".format(len(instance_names))
text = text + "\nC   Number of Materials: {:d}".format(len(material_names))
text = text + "\nC"
text = text + "\nC Details of mesh information are in file: {:s}".format(meshinfo)
text = text + "\nC"
text = text + "\nC PSEUDO CELLS"

sp = '-'
cell_num   = 0
cell_mat_names = []

for ins_name in instance_names:
    ins_data   = instance_data.get(ins_name)
    pname      = ins_data[0]

    pelsetdata       = part_elset_data.get(pname)
    order_elset_data = pelsetdata[0]
    check_key        = pelsetdata[2]

    for elset in order_elset_data:
        key_words = elset[0]
        if check_key not in key_words: continue

        elset_num   = elset[1]
        name        = '{:s}-{:d}'.format(pname,elset_num)

        pelsetmat   = part_elset_material_data.get(name)
        mat_name    = pelsetmat[0]
        mat_num     = pelsetmat[1]
        mat_density = pelsetmat[2]

        elset_name = sp.join(key_words) + '-{:d}'.format(elset_num)
        cell_num   = cell_num + 1

        text_info = text_info + "\n{:10d}".format(cell_num)
        if dentype == 'mass_den':
            text_info = text_info + " {:>10d} {:>20.6f} {:>50s}".format(mat_num, mat_density, mat_name)
        elif dentype == 'atom_den':
            text_info = text_info + " {:>10d} {:>20.6e} {:>50s}".format(mat_num, mat_density, mat_name)
        else:
            raise ValueError('>>>Error: invalid density unit')
        text_info = text_info + " {:>50s} {:>50s} {:>50s}".format(elset_name, pname, ins_name)

        if writecomments:
            text = text + '\nC\nC {:d}: {:s}, {:s}, {:s}'.format(cell_num, ins_name, pname, elset_name)

        if dentype == 'mass_den':
            text = text +'\n{:d} {:>8d} {:>12.6f}      0  u=1  $ {:s}'.format(cell_num, mat_num, -mat_density,mat_name)

        elif dentype == 'atom_den':
            text = text +'\n{:d} {:>8d} {:>12.6e}      0  u=1  $ {:s}'.format(cell_num, mat_num, mat_density,mat_name)

        else:
            raise ValueError('>>>Error: invalid density unit')

        cell_mat_names.append(mat_name)

if bgmaterial != 0:
    bgmaterial_data = material_data.get(bgmaterial)
    if bgmaterial_data is None:
        print(">>>Warning: background material {:d} is not in the list of materials".format(bgmaterial))
        print("            set background material to 0")
        bgmaterial = 0
        bgden      = None
    else:
        bgden = bgmaterial_data[1]

else:
    bgden = None

if writecomments: text = text+'\nC'

if bgden is None:
    text = text + '\n{:d} {:>8d} {:>12s}      0  u=1  $ background'.format(cell_num+1,bgmaterial,'')
else:
```

```
            if dentype == 'mass_den':
                bgden = -bgden
                text = text +'\n{:d} {:>8d} {:>12.6f}      0  u=1  $ background'.format(cell_num, bgmaterial, bgden)
            elif dentype == 'atom_den':
                text = text +'\n{:d} {:>8d} {:>12.6e}      0  u=1  $ background'.format(cell_num, bgmaterial, bgden)

        mat_num = 0
        text = text + '\nC'
        text = text + '\nC LEGACY CELLS'
        text = text + '\n{:d} {:>8d} {:>12s}     -999  fill=1 $ fill cell'.format(cell_num+2, mat_num, '')

        if include_cell_cards:
            f = open(cellcards,'r')
            lines = f.readlines()
            f.close()
            text  = text + "\nC\n"
            text = text + ''.join(lines)
        else:
             text = text + '\n{:d} {:>8d} {:>12s}      999'.format(cell_num+3,mat_num,'')

        text = text + '\n'
        text = text + '\nC Surface Cards'
        text = text + '\nC'
        text = text + '\n999 sph {:.15.6E} {:.15.6E} {:.15.6E} {:.15.6E}'.format(center[0], center[1], center[2], radius)
        if include_surface_cards:
            f = open(surfacecards,'r')
            lines = f.readlines()
            f.close()
            text  = text + "\nC\n"
            text  = text +''.join(lines)

        text = text+'\n'
        text = text+'\nC Data Cards'

        mat_name = cell_mat_names[0]
        text2= "1 1  $ {:s}".format(mat_name)
        text3= ''
        for i in range(1,cell_num):
            mat_name = cell_mat_names[i]
            text3 = text3 +  "\n               {:d} {:d}  $ {:s}".format(i+1,i+1,mat_name)

        text = text + "\nC"
        text = text + "\nembed1 meshgeo=abaqus"
        text = text + "\n       mgeoin={:s}".format(filein)
        text = text + "\n       meeout={:s}".format(eeout)
        text = text + "\n       length={:E}".format(lenconv)
        text = text + "\n       background={:d}".format(cell_num+1)
        text = text + "\n       matcell={:s}".format(text2)
        text = text + text3

        if include_data_cards:
            text = text +'\nC\n'
            f = open(datacards,'r')
            lines = f.readlines()
            f.close()
            text = text +''.join(lines)
        else:
            text = text+'\n'

        # write MCNP skeleton input file
        f = open(fileout,'w')
        f.write(text)
        f.close()
        print("\n   MCNP input file: {:s}".format(fileout))

        # write unstructured mesh information file
        if writemeshinfo:
            text_info = text_info + '\n\n' + mesh_info
        f = open(meshinfo,'w')
        f.write(text_info)
        f.close()
        if writemeshinfo:
            print("   Mesh information file: {:s}".format(meshinfo))

        print('\n')

        return text, text_info

    #-----------------------------------------------------------------------
    if __name__=='__main__':
        import argparse

        parser = argparse.ArgumentParser(description='** Write MCNP Unstructured Mesh Input file **')

        parser.add_argument("-i", "--input", required=True, metavar="<file.inp>",
                            type=str,
                            help="Abaqus input for constructing MCNP input file")
        parser.add_argument("-o", "--output", metavar="<file.mcnp>",
                            type=str,
                            help="output file name")

        parser.add_argument("-e", "--eeout",metavar="<file.eeout>",
                            type=str,
                            help="eeout file name in EMBED card")
        parser.add_argument('-mi','--meshinfo', metavar='<filename.info>',
                            type=str,
                            help='a mesh information file name')

        parser.add_argument("-cc", "--cellcards", metavar="<cellcards.txt>",
                             type=str,
                             help='MCNP cell cards file to include')

        parser.add_argument("-sc", "--surfacecards", metavar="<surfacecards.txt>",
                             type=str,
                             help='MCNP surface cards file to include')

        parser.add_argument("-dc", "--datacards", metavar="<datacards.txt>",
                             type=str,
                             help='MCNP data cards file to include')

        parser.add_argument("-b", "--back", metavar="<material_number>",
                            default=0, type=int,
                            help='background material for MCNP input file')

        parser.add_argument("-l", "--length", metavar="<len_conversion>",
                            default=1.0, type=float,
                            help="a multiplication conversion factor to centimeters")

        parser.add_argument('-re', '--radext', metavar="<radius_extension>",
```

```
                        default=1.0, type=float,
                        help="a radius extension of a sphere surface for a fill cell (in centimeters)")

    parser.add_argument("-du", "--densityunit", metavar="<density_unit>",
                         default='g/cm^3', type=str,
                         help='density unit option: g/cm^3 [default] or atoms/barn-cm')

    parser.add_argument("-wc", "--writecomments",
                        action='store_true',
                        help='write pseudo-cell comments in MCNP input file')

    parser.add_argument("-wm", "--writemeshinfo",
                        action='store_true',
                        help='write mesh information into mesh information file')


    args = parser.parse_args()

    if args.input is not None:
        write_mcnp_um_input(filein=args.input,
                            fileout=args.output,
                            eeout=args.eeout,
                            meshinfo=args.meshinfo,
                            cellcards=args.cellcards,
                            surfacecards=args.surfacecards,
                            datacards=args.datacards,
                            bgmaterial=args.back,
                            lenconv=args.length,
                            radiusext=args.radext,
                            denunit=args.densityunit,
                            writemeshinfo=args.writemeshinfo,
                            writecomments=args.writecomments)
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
  Code:         abaqus_input.py

  Author:       Jerawan Armstrong
                Monte Carlo Codes (XCP-3) Group
                X Computational Physics (XCP) Division
                Los Alamos National Laboratory

  Copyright (c) 2020 Triad National Security, LLC.  All rights reserved.

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in all
  copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
  SOFTWARE.
"""
import numpy as np
import os

element_types  = ['c3d4', 'c3d6', 'c3d8', 'c3d10', 'c3d15', 'c3d20', 'sc8']

elset_keywords = ['material', 'statistic', 'source', 'matprop']

#-----------------------------------------------------------------------
class AbaqusInp:
    """ Abaqus Input """
    def __init__(self):
        self.heading   = ''
        self.parts     = []
        self.instances = []
        self.materials = []

    def set_heading(self,head):
        self.heading = head

    def get_heading(self):
        return self.heading

    def set_parts(self,parts):
        self.parts = parts

    def get_parts(self):
        return self.parts

    def set_instances(self,instances):
        self.instances = instances

    def get_instances(self):
        return self.instances

    def set_materials(self,materials):
```

```
        self.materials = materials

    def get_materials(self):
        return self.materials


#-------------------------------------------------------------------------
def read_abaqus(filename):
    if not os.path.isfile(filename):
        raise OSError('file does not exist: {:s}'.format(filename))
    else:
        print("\nRead an Abaqus input file: {:s}\n".format(filename))
        with open(filename, "r", encoding='utf-8',errors='ignore') as f:
            names, inpdata, inplines  = _read_buffer(f)

        return names, inpdata, inplines

#-------------------------------------------------------------------------
def _read_buffer(f):

    # ----- BEGIN READING HEADING -----
    heading      = ''
    need_newline = False
    while True:
        line = f.readline()
        if not line: raise EOFError('read heading')

        if line.startswith('**'): continue

        if line.startswith('*'):
            lline = line.lower()

            if lline.startswith('*part'): break

            if lline.startswith('*heading'):
                line = f.readline()
                if not line: raise EOFError('read heading data')

                lline = line.lower()
                if lline.startswith('*part'): break

                if not line.startswith('**'): heading = line.strip()
                need_newline = True
                break
    # ----- END READING HEADING -----


    # ----- BEGIN READING PARTS -----
    part_names = []
    part_lines = {}
    part_data  = {}
    read_flag  = 0

    while True:
        if need_newline:
            line  = f.readline()
            if not line: raise EOFError('read part')
        else:
            need_newline = True

        if line.startswith('**'):
            continue
        elif line.startswith('*'):
            lline = line.lower()
        else:
            continue

        #---------------
        #      Part
        #---------------
        if read_flag == 0 and lline.startswith('*part'):
            pname = _extract_value(line[6:],'name')

            if len(pname) > 0:
                if pname in part_names:
                    raise ValueError('part name is not unique: {:s}'.format(pname))
                part_names.append(pname)
                pdata     = []
                plines    = []
                read_flag = 1
            else:
                raise ValueError('{:s} does not have a part name'.format(line))

        #-----------------
        #      Node
        #-----------------
        elif read_flag == 1 and lline.startswith('*node'):
            temp_lines = []

            # ----- [begin] read node data -----
            while True:
                line = f.readline()
                if not line: raise EOFError('read node data of part {:s}'.format(pname))

                if line.startswith('*'):
                    lline = line.lower()
                    break
                else:
                    temp_lines.append(line)
            # ----- [end] read node data -----

            n = len(temp_lines)
            if n > 0:
                # pdata[0]
                pdata.append(n)  # number of nodes
                # plines[0]
                plines.append(temp_lines)

                read_flag    = 2
                need_newline = False
            else:
                raise ValueError('no node data for part {:s}'.format(pname))

        #------------------------
        #      Element Type
        #------------------------
        elif read_flag == 2 and lline.startswith('*element'):
            n_part_elements = 0  # number of elements in this part
            element_data    = []
```

```
    temp_lines      = []

    # ----- [begin] check element types -----
    while True:
        tvalue = _extract_value(lline[9:], 'type')

        if len(tvalue) == 0 :
            raise ValueError('{:s} does not an element type'.format(line))
        else:
            etype = _extract_etype(tvalue)

            if len(etype) == 0:
                raise ValueError('invalid element type: {:s}'.format(line))
            else:

                cur_temp_lines = []

                # ----- [begin] read element data -----
                while True:
                    line = f.readline()
                    if not line: raise EOFError('read element data for part {:s}'.format(pname))

                    if line.startswith('*'):
                        lline = line.lower()
                        break
                    else:
                        cur_temp_lines.append(line)
                # ----- [end] read element data -----

                n = len(cur_temp_lines)
                if n > 0:
                    if etype == 'c3d20':
                        if n%2 == 0:
                            n = int(n/2)
                            element_data.append([etype, n])
                        else:
                            raise ValueError('bad number of C3D8 elements in part: {:}'.format(pname))
                    else:
                        element_data.append([etype, n])

                    n_part_elements += n
                    temp_lines.append(cur_temp_lines)

                else:
                    raise ValueError('no element data for element type {:s} in part {:s}'.format(etype,pname))

                if not lline.startswith('*element,'): break

                if len(element_data)==2:
                    raise ValueError('number of *element lines > 2 in part {;}'.format(pname))
    # ----- [end] check element types -----

    if len(element_data) == 0:
        raise ValueError('no element data in part {:s}'.format(pname))

    if len(element_data) == 2:
        n = [0, 0, 0, 0, 0, 0, 0]
        for d in element_data:
            etype = d[0]
            if    etype == 'c3d4'  : n[0] = n[0]+1
            elif  etype == 'c3d6'  : n[1] = n[1]+1
            elif  etype == 'c3d8'  : n[2] = n[2]+1
            elif  etype == 'c3d10' : n[3] = n[3]+1
            elif  etype == 'c3d15' : n[4] = n[4]+1
            elif  etype == 'c3d20' : n[5] = n[5]+1
            elif  etype == 'sc8'   : n[6] = n[6]+1

        for i in n:
            if i==2:
                raise ValueError('two lines of the same element type in a part is not allowed; part name {:s}'.format(pname))

        if n[0]==1:
            raise ValueError('c3d4 type cannot mix with another element type; part name {:s}'.format(pname))

        if n[1]==1:
            if n[4]==1:
                raise ValueError('c3d6 type cannot mix with c3d15 type; part name {:s}'.format(pname))
            if n[5]==1:
                raise ValueError('c3d6 type cannot mix with c3d20 type; part name {:s}'.format(pname))
            if n[6]==1:
                raise ValueError('c3d6 type cannot mix with sc8 type; part name {:s}'.format(pname))

        if n[2]==1:
            if n[4]==1:
                raise ValueError('c3d8 type cannot mixed with c3d15 type; part name {:s}'.format(pname))
            if n[5]==1:
                raise ValueError('c3d8 type cannot mixed with c3d20 type; part name {:s}'.format(pname))

        if n[3]==1:
            raise ValueError('c3d10 type cannot mixed with another element type; part name {:s}'.format(pname))

        if n[4]==1:
            if n[6]==1:
                raise ValueError('c3d15 type cannot mixed with sc8 type; part name {:s}'.format(pname))

    # pdata[1]
    pdata.append(element_data)  # [ [element_type, number_of_elements], ....]

    # plines[1]
    plines.append(temp_lines)

    read_flag    = 3
    need_newline = False


# ------------ Elset W/O Internal ---------------------
elif read_flag == 3 and lline.startswith('*elset') and lline.find('internal') == -1:
    n_part_mat_elsets     = 0  # number of material elsets in this part
    n_part_matprop_elsets = 0  # number of material property elsets in this part
    n_part_stat_elsets    = 0  # number of stat elsets in this part
    n_part_src_elsets     = 0  # number of source elsets i this part

    keyword_lines   = []
    elset_data      = []
    temp_lines      = []

    # ----- [begin] check eleset keywords -----
    while True:
        keywords = _extract_keywords(lline[7:])
```

```
    if len(keywords) == 0:
        elset_flag = False
    else:
        elset_flag = True

    #---------- begin elset_flag --------------------
    if elset_flag:
        elset_keyword_line = line
        elset_name         = _extract_value(line[7:], 'elset')
        elset_num          = _extract_number(elset_name)
        if elset_num is None:
            raise ValueError('elset name does not ending with a number: {:s}'.format(elset_name))
        elif elset_num <= 0:
            raise ValueError('elset number <= 0 for elset name : {:s}'.format(elset_name))

        generate_index = lline.rfind('generate')

        # ----- [begin] read elset data -----
        cur_temp_lines = []
        while True:
            line = f.readline()
            if not line:  raise EOFError('read elset data of part {:s}'.format(pname))

            if line.startswith('_'): break
            if line.startswith('*'): break

            cur_temp_lines.append(line)

            if generate_index > 0:
                this_elset_line   = np.int_(line.strip().split(','))
                n_this_elsets     = int((this_elset_line[1]-this_elset_line[0]+1)/this_elset_line[2])
            else:
                this_elset_line   = line.strip().split(',')
                n_this_elsets     = len(this_elset_line)

            if 'material'  in keywords: n_part_mat_elsets      += n_this_elsets
            if 'matprop'   in keywords: n_part_matprop_elsets  += n_this_elsets
            if 'statistic' in keywords: n_part_stat_elsets     += n_this_elsets
            if 'source'    in keywords: n_part_src_elsets      += n_this_elsets
            # ----- [end] read elset data -----

        n = len(cur_temp_lines)
        if n > 0:
            keyword_lines.append(elset_keyword_line.strip())
            elset_data.append([keywords, elset_num, generate_index, n])
            temp_lines.append(cur_temp_lines)

            if line.startswith('*'):
                lline = line.lower()
                if lline.startswith('*elset,') and lline.find('internal') == -1: continue
                if lline.startswith('*end part'): break
        else:
            raise ValueError('no elset data for elset line {:s} of part {:s}'.format(elset_keyword_line,pname))
    #---------- end elset_flag --------------------

    endpart_flag = False
    while True:
        line = f.readline()
        if not line:  raise EOFError('read elset data in a part')

        if line.startswith('*'):
            lline = line.lower()
            if lline.startswith('*elset,') and lline.find('internal') == -1:
                endpart_flag = False
                break
            if lline.startswith('*end part'):
                endpart_flag = True
                break

    if endpart_flag: break
# ----- [end] check eleset keywords -----

#--------------------------------
# check elset data in this part
# --------------------------------
if len(elset_data) == 0:
    raise ValueError('no elset data for part {:s}'.format(pname))

elset_mat_nums       = []
elset_matprop_nums   = []
elset_statistic_nums = []
elset_source_nums    = []

mat_elsets     = 0  # number of material elset keyword lines in this part
matprop_elsets = 0  # number of matprop elset keyword lines in this part
stat_elsets    = 0  # number of statistic elset keyword lines in this part
source_elsets  = 0  # number of source elset keyword lines in this part

for elset in elset_data:
    keywords  = elset[0]
    elset_num = elset[1]

    if 'material' in keywords:
       elset_mat_nums.append(elset_num)
       mat_elsets     += 1

    if 'matprop' in keywords:
        if elset_num in elset_matprop_nums:
            raise ValueError('part {:s} has multiplie elset matprop number {:d}'.format(pname, elset_num))
        else:
            elset_matprop_nums.append(elset_num)
            matprop_elsets += 1

    if 'statistic' in keywords:
        elset_statistic_nums.append(elset_num)
        stat_elsets += 1

    if 'source' in keywords:
        if elset_num in elset_source_nums:
            raise ValueError('part {:s} has multiple elset source number {:d}'.format(elset_num,pname))
        else:
            elset_source_nums.append(elset_num)
            source_elsets += 1


if mat_elsets == 0:
    raise ValueError('no material elset in part {:s}'.format(pname))

# all elements in each part must be assigned to a material
```

```
    if n_part_mat_elsets < n_part_elements:
        raise ValueError('not all elements in part {:s} are assigned to a material'.format(pname))
    elif n_part_mat_elsets > n_part_elements:
        raise ValueError('number of elements for material elsets > number of elements in part {:s}'.format(pname))


    if matprop_elsets > 0:
        if mat_elsets > 1:
            for elset in elset_data:
                keywords = elset[0]
                if 'matprop' in keywords and 'material' in keywords: continue
                raise ValueError('multiple material properties and multiple materials in a part is not allowed')

        if stat_elsets > 1:
            for elset in elset_data:
                keywords = elset[0]
                if 'matprop' in keywords and 'material' in keywords and 'statistic' in keywords: continue
                raise ValueError('multiple material properties and multiple statistics in a part is not allowed')

        if source_elsets > matprop_elsets:
            raise ValueError('number of source elsets > number of matprop elsets in part {:s}'.format(pname))

        if n_part_matprop_elsets < n_part_elements:
            print("number of matprop elsets = {:d}".format(n_part_matprop_elsets))
            print("number of elements in this part = {:d}".format(n_part_elements))
            raise ValueError('not all elements in part {:s} are assigned to a material property'.format(pname))
        elif n_part_matprop_elsets > n_part_elements:
            print("number of matprop elsets = {:d}".format(n_part_matprop_elsets))
            print("number of elements in this part = {:d}".format(n_part_elements))
            raise ValueError('number ot elements for material property elsets > number of elements in part {:s}'.format(pname))

    else:
        if stat_elsets == 0:
            raise ValueError('no statisitc elset in part {:s}'.format(pname))

        if mat_elsets > stat_elsets:
            raise ValueError('not all materials in part {:s} are assigned to statistics regions'.format(pname))

        if source_elsets > stat_elsets:
            raise ValueError('number of source elsets > number of statistics elsets in part {:s}'.format(pname))

        if n_part_stat_elsets < n_part_elements:
            raise ValueError('not all elements in part {:s} are assigned to a statistics'.format(pname))
        elif n_part_matprop_elsets > n_part_elements:
            raise ValueError('number ot elements for statistic elsets > number of elements in part {:s}'.format(pname))

    # pdata[2]
    pdata.append(elset_data)

    # plines[2]
    plines.append(temp_lines)

    # pdata[3]
    pdata.append([mat_elsets, matprop_elsets, stat_elsets, source_elsets])

    # pdata[4]
    pdata.append([n_part_mat_elsets, n_part_matprop_elsets, n_part_stat_elsets, n_part_src_elsets])

    # pdata[5]
    pdata.append([elset_mat_nums, elset_matprop_nums, elset_statistic_nums, elset_source_nums])

    # pdata[6]
    pdata.append(keyword_lines)

    #-----------------------------------------------------------------------------------------------------------
    # pdata is a list
    #    pdata[0] = number_of_nodes
    #
    #    pdata[1] = element_data, element_data is a list with length n where n is a number of element types in a part
    #
    #               element_data[i] = [etype, number_of_elements]
    #
    #    pdata[2] = elset_data, elset_data is a list with length n where n is a number of elsets in a part
    #
    #               elset_data[i] = [keywords,elset_number, generate_index, number_of_elset_lines]
    #
    #               keywords is a list with a length 1, 2, 3, or 4 depending on the elset name. For example,
    #
    #                  keywords = ['material']
    #                             for *Elset, elset=????material????-xxxx
    #
    #                  keywords = ['material', 'statistic']
    #                             for *Elset, elset=????material-statistic????-xxxx
    #
    #                  keywords = ['material', 'statistic', 'source']
    #                             for *Elset, elset=????material-statistic-source????-xxxx
    #
    #                  keywords = ['material', 'staitctic', 'soource', 'matprop']
    #                             for *Elset, elset=????material-statistic-source-source????-xxxx
    #
    #    pdata[3] = [mat_elsets, matprop_elsets, stat_elsets, source_elsets]
    #
    #    pdata[4] = [n_part_mat_elsets, n_part_matprop_elsets, n_part_stat_elsets, n_part_src_elsets]
    #
    #    pdata[5] = [elset_mat_nums, elset_matprop_nums, elset_statistic_nums, elset_source_nums]
    #
    #    pdata[6] = keyword_lines = [elset_keyword_line1, elset_keyword_line2, ...]
    #
    #  plines is a list of data lines in a part. There are 3 slices of data
    #         plines[0] = node data
    #         plines[1] = element data
    #         plines[2] = elset data
    #
    # -----------------------------------------------------------------------------------------------------------
    part_data[pname]  = pdata
    part_lines[pname] = plines
    read_flag         = 0


# ------------ END PART -------------
elif lline.startswith('*end part'):
    if read_flag==0:
        raise ValueError('mismatch *part and *end part')
    elif read_flag==1:
        raise ValueError('no node data in part {:s}'.format(pname))
    elif read_flag==2:
        raise ValueError('no element data in part {:s}'.format(pname))
    elif read_flag==3:
        raise ValueError('no elset data in part {:s}'.format(pname))
```

```
    # --------- ASSEMBLY ------------------
    elif lline.startswith('*assembly'):
        if read_flag == 0:
            need_newline = False
            break
        else:
            raise ValueError('bad Abaqus input format when reading *assembly')

    # ------------ MATERIAL ----------------
    elif lline.startswith('*material,'):
        if read_flag == 0:
            need_newline = False
            break
        else:
            raise ValueError('bad Abaqus input format when reading *material')
#---------------- END READING PARTS --------------------

temp_line = []

if len(part_names)==0: raise ValueError('no part data in this file')
if read_flag != 0: raise ValueError('mismatch *part and *end part')


#----------- BEGIN READING ASSEMBLY & MATERIALS ----------
num_assembly     = 0

instance_names   = []
instance_lines   = {}
instance_data    = {}

material_names   = []
material_nums    = []
material_data    = {}

assembly_flag    = 0
material_flag    = 0

while True:
    if need_newline:
        line  = f.readline()
        if not line: break
    else:
        need_newline = True

    if line.startswith('**'):
        continue
    elif line.startswith('*'):
        lline = line.lower()
    else:
        continue

    # --------- *ASSEMBLY -----------
    if lline.startswith('*assembly'):
        if num_assembly == 0:
            num_assembly  = num_assembly + 1
            need_newline  = True
            assembly_flag = 1
            material_flag = 0
        else:
            raise ValueError('file has more than 1 assembly')

    # ---------- * INSTANCE -----------
    elif assembly_flag == 1 and lline.startswith('*instance,'):

        ins_name = _extract_value(line[10:],'name')
        if len(ins_name) == 0 :
            raise ValueError('no instance name {:s}'.format(line))
        else:
            if ins_name in instance_names:
                raise ValueError('instance name is not unique {:s}'.format(line))
            else:
                pname = _extract_value(line[10:],'part')
                if pname not in part_names:
                    raise ValueError("instance's partname does not match with any partname. \n{:s}".format(line))

                else:
                    inslines = []
                    ndata    = 0
                    while True:
                        line  = f.readline()
                        if not line:  raise EOFError('read instance data {:s}'.format(ins_name))

                        if line.startswith('*'):
                            if line.lower().startswith('*end instance'):
                                break
                            else:
                                raise ValueError('bad instance data; see {:s}'.format(line))

                        else:
                            if ndata >= 2:
                                raise ValueError('bad instance data; see {:s}'.format(line))
                            else:
                                inslines.append(line)
                                ndata = ndata + 1

                    instance_names.append(ins_name)
                    instance_data[ins_name]  = [pname, ndata]
                    instance_lines[ins_name] = inslines
                    need_newline             = True

    # -------- *END ASSEMBLY --------------
    elif lline.startswith('*end assembly'):
        assembly_flag = 0
        material_flag = 0
        need_newline  = True

    # ---------- *MATERIAL ----------------
    elif lline.startswith('*material,'):
        mat_name = _extract_value(line[10:],'name')
        if len(mat_name) == 0 :
            raise ValueError('no material name {:s}'.format(line))
        else:
            if mat_name in material_names:
                raise ValueError('material name is not unique {:s}'.format(mat_name))
            else:
                mat_num = _extract_number(mat_name)
                if mat_num is None:
```

```
                    raise ValueError('material name is not ending with number')
                else:
                    if mat_num is material_nums:
                       raise ValueError('material number is not unique {:s}'.format(mat_name))
                    material_names.append(mat_name)
                    material_nums.append(mat_num)
                    material_data[mat_num] = [mat_name, 0.0]
                    material_flag          = 1
                    need_newline           = True

        elif lline.startswith('*density'):
            if material_flag == 0:
                raise ValueError('mismatch material and its density')
            else:
                line  = f.readline()
                if not line:  raise EOFError('read density data')

                densities = line.split(',')

                if len(densities) == 0:
                    raise ValueError('bad density value: {:s}'.format(line))
                else:
                    density = densities[0].strip()
                    try:
                        density                = float(density)
                        material_data[mat_num] = [mat_name, density]
                        material_flag          = 0
                    except:
                        raise ValueError('density value is not a number: {:f}'.format(line))

    #---------- END READING ASSEMBLY & MATERIALS ----------
    if len(instance_names) == 0:
        raise  ValueError('no instance in this file')
    if len(material_names) == 0:
        raise ValueError('no material in this file')

    inpdata  = AbaqusInp()
    inplines = AbaqusInp()

    inpdata.set_heading(heading)
    inpdata.set_parts(part_data)
    inpdata.set_instances(instance_data)
    inpdata.set_materials(material_data)

    inplines.set_heading('')
    inplines.set_parts(part_lines)
    inplines.set_instances(instance_lines)
    inplines.set_materials('')

    names = [part_names, instance_names, material_names]

    return [names, inpdata, inplines]

#-----------------------------------------------------------------------
def _extract_value(line,option):
    """
    Extract value of option from line.

    Example: line   = '*Part, name=" part-name "'
             option = Name
             value  = 'part-name'
    """
    value = ''

    line_list = line.split(',')

    for l in line_list:
        i = l.lower().find(option.lower())
        if i >= 0:
            j = l[i:].find('=')
            if j > 0:
                value = l[i+j+1:].strip().strip('"')
                break

    return value

#-----------------------------------------------------------------------
def _extract_etype(value):
    """
    Extract etype from value

    Example: value = 'C3D8R'
             etype = 'c3d8'
    """
    etype = ''
    for et in element_types:
        i = value.lower().find(et)
        if i >= 0 :
            etype = et
            break

    return etype

#-----------------------------------------------------------------------
def _extract_keywords(value):
    """
    Extract keywords from value

    Example: value = 'set-material-statistic-001'
             keywords = [material, stattistic]
    """
    keywords = []
    for kw in elset_keywords:
        i = value.lower().find(kw)
        if i >= 0: keywords.append(kw)

    if 'statistic' not in keywords:
        i = value.lower().find('tally')
        if i >= 0: keywords.append('statistic')

    return keywords

#-----------------------------------------------------------------------
def _extract_number(value):
    """
    Extract number from value.

    Example: value ='Set-material-001'
             number   = 1
```

```
    """
    number = None
    i = value.rfind('-')
    j = value.rfind('_')
    k = max(i,j)
    if k >= 0:
        if value[k+1:].isnumeric():
            number = int(value[k+1:])
        else:
            raise ValueError('{:s} does not end with a number'.format(value))

    return number

#--------------------------------------------------------------------------
def extract_node_data(line):
    line = line.strip().split(',')
    return int(line[0]), list(np.float_(line[1:]))

#--------------------------------------------------------------------------
def extract_element_data(line):
    line = line.strip().split(',')
    if '' in line: line.remove('')
    return list(np.int_(line))

#--------------------------------------------------------------------------
def extract_elset_data_generate(line):
    line = line.strip().split(',')
    return list(np.int_(line))

#--------------------------------------------------------------------------
def extract_elset_data(line):
    line = line.strip().split(',')
    if '' in line: line.remove('')
    return list(np.int_(line))

#--------------------------------------------------------------------------
def extract_instance_data(line):
    line = line.strip().split(',')
    return list(np.float_(line))

#--------------------------------------------------------------------------
def extract_mesh_data(names, inpdata, inplines, dentype):
    """
    Extract mesh data
    """
    part_names      = names[0]
    instance_names  = names[1]
    material_names  = names[2]

    heading       = inpdata.get_heading()
    part_data     = inpdata.get_parts()
    instance_data = inpdata.get_instances()
    material_data = inpdata.get_materials()

    part_lines     = inplines.get_parts()
    instance_lines = inplines.get_instances()


    mesh_info = ''
    if len(heading) > 0:
        mesh_info = mesh_info + '\n   heading: {:s}\n'.format(heading)

    mesh_info = mesh_info + '\n   number of parts: {:d}'.format(len(part_names))
    mesh_info = mesh_info + '\n   number of instances: {:d}'.format(len(instance_names))
    mesh_info = mesh_info + '\n   number of materials: {:d}\n'.format(len(material_names))

    total_nodes    = 0
    total_elements = []
    for i in range(len(element_types)): total_elements.append(0)

    for pname in part_names:
        pdata = part_data.get(pname)
        total_nodes += pdata[0]

        element_data = pdata[1]
        for el in element_data:
            etype = el[0]
            nels  = el[1]
            if   etype == 'c3d4':  total_elements[0] += nels
            elif etype == 'c3d6':  total_elements[1] += nels
            elif etype == 'c3d8':  total_elements[2] += nels
            elif etype == 'c3d10': total_elements[3] += nels
            elif etype == 'c3d15': total_elements[4] += nels
            elif etype == 'c3d20': total_elements[5] += nels
            elif etype == 'sc8':   total_elements[6] += nels
            else:
                raise ValueError('invalid element type :{:s}'.format(etype))

    sum_elements = np.sum(total_elements)
    mesh_info = mesh_info + '\n   total  number of nodes: {:d}'.format(total_nodes)
    mesh_info = mesh_info + '\n   total  number of elements: {:d}\n'.format(sum_elements)
    if total_elements[0] > 0:
        mesh_info = mesh_info + '\n   total  number of linear tet elements: {:d}'.format(total_elements[0])
    if total_elements[1] > 0:
        mesh_info = mesh_info + '\n   total  number of linear pent elements: {:d}'.format(total_elements[1])
    if total_elements[2] > 0:
        mesh_info = mesh_info + '\n   total  number of linear hex elements: {:d}'.format(total_elements[2])
    if total_elements[3] > 0:
        mesh_info = mesh_info + '\n   total  number of quad tet elements: {:d}'.format(total_elements[3])
    if total_elements[4] > 0:
        mesh_info = mesh_info + '\n   total  number of quad pent elements: {:d}'.format(total_elements[4])
    if total_elements[5] > 0:
        mesh_info = mesh_info + '\n   total  number of quad hex elements: {:d}'.format(total_elements[5])
    if total_elements[6] > 0:
        mesh_info = mesh_info + '\n   total  number of sc8 elements: {:d}'.format(total_elements[6])
    mesh_info = mesh_info + '\n'

    for pname in part_names:
        pdata    = part_data.get(pname)
        plines   = part_lines.get(pname)

        n_nodes   = pdata[0]
        mesh_info = mesh_info + '\n\n   *** part name: {:s}'.format(pname)
        mesh_info = mesh_info + '\n   *** number of nodes: {:d}'.format(n_nodes)

        element_data = pdata[1]
        for el in element_data:
            etype = el[0]
            nels  = el[1]
```

```
            mesh_info = mesh_info + '\n   *** element_type: {:s}'.format(etype)
            mesh_info = mesh_info + '\n       number of elements: {:d}'.format(nels)

        sp = '-'
        elset_data  = pdata[2]
        elset_lines = plines[2]

        for elset, lines  in zip(elset_data,elset_lines):
            keywords              = elset[0]
            elset_num             = elset[1]
            generate_index        = elset[2]

            if generate_index >= 1:
                line      = lines[0]
                line_data = extract_elset_data_generate(line)
                nelsets   = int((line_data[1] - line_data[0] + 1)/ line_data[2])

            else:
                nelsets = 0
                for line in lines:
                    line_data = extract_elset_data(line)
                    nelsets  += len(line_data)

            elset_keywords = sp.join(keywords)
            mesh_info = mesh_info + '\n   *** elset keywords: {:s}'.format(elset_keywords)
            mesh_info = mesh_info + '\n       elset number: {:d}'.format(elset_num)
            mesh_info = mesh_info + '\n       number of elements in this set: {:d}'.format(nelsets)

        mesh_info = mesh_info + '\n'

    for ins_name in instance_names:
        ins_data  = instance_data.get(ins_name)
        ins_lines = instance_lines.get(ins_name)

        mesh_info = mesh_info + '\n\n   *** instance name: {:s}'.format(ins_name)
        mesh_info = mesh_info + '\n       part name: {:s}'.format(ins_data[0])

        if ins_data[1]==0:
            mesh_info = mesh_info + '\n       translation: no'
            mesh_info = mesh_info + '\n       rotation: no'

        elif ins_data[1]==1:
            mesh_info = mesh_info + '\n       translation: yes'
            mesh_info = mesh_info + '\n       rotation: no'
            mesh_info = mesh_info + '\n       {:s}'.format(ins_lines[0])

        elif ins_data[1]==2:
            mesh_info = mesh_info + '\n       translation: yes'
            mesh_info = mesh_info + '\n       rotation: yes'
            mesh_info = mesh_info + '\n       {:s}'.format(ins_lines[0])
            mesh_info = mesh_info + '\n       {:s}'.format(ins_lines[1])
    mesh_info = mesh_info + '\n'

    material_nums = material_data.keys()

    for m in material_names:
        for n in material_nums:
            mdata = material_data.get(n)
            if mdata[0]==m: break
        mesh_info = mesh_info + '\n\n   *** material name: {:s}'.format(m)
        mesh_info = mesh_info + '\n       material number: {:d}'.format(n)
        if dentype == 'mass_den':
            mesh_info = mesh_info + '\n       material density: {:10.5f}'.format(mdata[1])
        elif dentype == 'atom_den':
            mesh_info = mesh_info + '\n       material density: {:10.5e}'.format(mdata[1])

    mesh_info = mesh_info + '\n'

    return mesh_info

#-----------------------------------------------------------------------
def compute_translated_points(begin_points,end_points):
    begin_points = np.array(begin_points)
    end_points   = np.array(end_points)
    return  begin_points + end_points

#-----------------------------------------------------------------------
def compute_rotated_points(points, begin_points, end_points, angle):
    points       = np.array(points)
    begin_points = np.array(begin_points)
    end_points   = np.array(end_points)

    angle    = angle*np.pi/180.0 # convert degree to radian
    cos_ang  = np.cos(angle)
    sin_ang  = np.sin(angle)
    a        = end_points - begin_points

    a2               = a*a
    a_time_sin_ang   = a*sin_ang
    a01              = a[0]*a[1]
    a02              = a[0]*a[2]
    a12              = a[1]*a[2]
    one_minus_cos_ang = 1.0 - cos_ang

    m = np.matrix('0 0 0; 0 0 0; 0 0 0')

    m[0,0] = cos_ang + one_minus_cos_ang * a2[0]
    m[0,1] = one_minus_cos_ang * a01 - a_time_sin_ang[2]
    m[0,2] = one_minus_cos_ang * a02 + a_time_sin_ang[1]

    m[1,0] = one_minus_cos_ang * a02  + a_time_sin_ang[2]
    m[1,1] = cos_ang + one_minus_cos_ang * a2[1]
    m[1,2] = one_minus_cos_ang * a12  - a_time_sin_ang[0]

    m[2,0] = one_minus_cos_ang * a02 - a_time_sin_ang[1]
    m[2,1] = one_minus_cos_ang * a12 + a_time_sin_ang[0]
    m[2,2] = cos_ang + one_minus_cos_ang * a2[2]

    x = points - begin_points
    x = x.reshape((3,1))

    return np.array(m*x).reshape((3,))+ begin_points

#-----------------------------------------------------------------------
def compute_global_model_extents(part_names, part_data, part_lines,
                                 instance_names, instance_data, instance_lines):
    """
    Compute global model extents
    """
```

```
    # process part data
    part_xpoints = {}
    part_ypoints = {}
    part_zpoints = {}

    for pname in part_names:
        plines = part_lines.get(pname)

        xpoints = []
        ypoints = []
        zpoints = []

        node_lines = plines[0]
        for line in node_lines:
            id, xyz = extract_node_data(line)
            xpoints.append(xyz[0])
            ypoints.append(xyz[1])
            zpoints.append(xyz[2])

        part_xpoints[pname] = [np.min(xpoints), np.max(xpoints)]
        part_ypoints[pname] = [np.min(ypoints), np.max(ypoints)]
        part_zpoints[pname] = [np.min(zpoints), np.max(zpoints)]


    # process instance data
    xmin_points = []
    xmax_points = []

    ymin_points = []
    ymax_points = []

    zmin_points = []
    zmax_points = []

    for ins_name in instance_names:
        ins_data  = instance_data.get(ins_name)
        ins_lines = instance_lines.get(ins_name)

        pname  = ins_data[0]
        nlines = ins_data[1]

        xmin, xmax = part_xpoints[pname]
        ymin, ymax = part_ypoints[pname]
        zmin, zmax = part_zpoints[pname]

        if nlines==1: # translation
            t  = extract_instance_data(ins_lines[0])
            if np.any(t != 0.0):
                pmin = [xmin,ymin,zmin]
                pmax = [xmax,ymax,zmax]
                xmin, ymin, zmin  = compute_translated_points(pmin,t)
                xmax, ymax, zmax  = compute_translated_points(pmax,t)

        elif nlines==2: # translation and rotation
            t  = extract_instance_data(ins_lines[0])
            if np.any(t != 0.0):
                pmin = [xmin,ymin,zmin]
                pmax = [xmax,ymax,zmax]
                xmin, ymin, zmin  = compute_translated_points(pmin,t)
                xmax, ymax, zmax  = compute_translated_points(pmax,t)

            r = extract_instance_data(ins_lines[1])
            if np.any(r !=0.0):
                p1    = r[0:3]
                p2    = r[3:6]
                angle = r[-1]

                pmin = [xmin,ymin,zmin]
                pmax = [xmax,ymax,zmax]
                xmin, ymin, zmin = compute_rotated_points(pmin, p1, p2, angle)
                xmax, ymax, zmax = compute_rotated_points(pmax, p1, p2, angle)

        xmin_points.append(xmin)
        xmax_points.append(xmax)

        ymin_points.append(ymin)
        ymax_points.append(ymax)

        zmin_points.append(zmin)
        zmax_points.append(zmax)

    xmax = np.max(xmax_points)
    xmin = np.min(xmin_points)

    ymax = np.max(ymax_points)
    ymin = np.min(ymin_points)

    zmax = np.max(zmax_points)
    zmin = np.min(zmin_points)

    return [xmin,xmax,ymin,ymax,zmin,zmax]

#-----------------------------------------------------------------------


#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
  Code:        abq_part_elsets.py

  Author:      Jerawan Armstrong
               Monte Carlo Codes (XCP-3) Group
               X Computational Physics (XCP) Division
               Los Alamos National Laboratory

  Copyright (c) 2020 Triad National Security, LLC.  All rights reserved.

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in all
  copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
```

```
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
  SOFTWARE.
"""
import numpy as np
from abaqus_input import extract_elset_data_generate
from abaqus_input import extract_elset_data

#-----------------------------------------------------------------------
def compute_part_elset_mat_nums(part_names, part_data, material_data):

    material_nums = material_data.keys()

    part_elset_mat_nums = {}

    for pname in part_names:
        elset_data = part_data.get(pname)[2]

        mat_nums = []

        for elset in elset_data:
            if 'material' not in elset[0]: continue

            elset_num = elset[1]

            if elset_num in material_nums:

                if elset_num not in mat_nums: mat_nums.append(elset_num)

            else:
                raise ValueError('elset material num {:d} in part {:s} \
is not in a list of materials'.format(elset[1],pname))

        if len(mat_nums) > 0:
            part_elset_mat_nums[pname] = mat_nums
        else:
            raise ValueError('no material for part {:s}'.format(pname))

    return part_elset_mat_nums

#-----------------------------------------------------------------------
def compute_part_elset_data(part_names, part_data):
    part_elset_data = {}

    for pname in part_names:
        pdata          = part_data.get(pname)
        elset_data     = pdata[2]
        mat_elsets     = pdata[3][0]
        matprop_elsets = pdata[3][1]
        stat_elsets    = pdata[3][2]

        if matprop_elsets >= stat_elsets:
            check_key    = 'matprop'
            n_check_keys = matprop_elsets
        else:
            check_key    = 'statistic'
            n_check_keys = stat_elsets

        if mat_elsets == 1 and n_check_keys == 1:
            complex_part_elset = False
        else:
            complex_part_elset = True

        order_elset_data     = []
        order_elset_numbers  = []

        for eldata in elset_data:
            add_data     = False
            elset_number = eldata[1]

            for i in range(len(order_elset_numbers)):
                if elset_number <= order_elset_numbers[i]:
                    order_elset_data.insert(i,eldata)
                    order_elset_numbers.insert(i,elset_number)
                    add_data = True
                    break

            if not add_data:
                order_elset_data.append(eldata)
                order_elset_numbers.append(elset_number)

        if len(order_elset_data) != len(elset_data):
            raise ValueError('bad elset number for part {:>s}'.format(pname))

        else:
            part_elset_data[pname] = [order_elset_data, elset_data, check_key, mat_elsets, complex_part_elset]

    return part_elset_data


#-----------------------------------------------------------------------
def compute_part_elset_material_data(part_names, part_lines, part_elset_data, material_data):
    part_elset_material_data = {}
    names = []

    for pname in part_names:
        pelsetdata          = part_elset_data.get(pname)
        elset_data          = pelsetdata[1]
        check_key           = pelsetdata[2]
        complex_part_elset  = pelsetdata[4]

        # -------------------------------------------------------------
        # check for materials in a part that has complex elset lines
        if complex_part_elset:
            elset_lines  = part_lines.get(pname)[2]

            elset_mat_data = []
            for elset, lines in zip(elset_data,elset_lines):
                # elset[0] = list of elset keywords
                # elset[1] = elset num
                # elset[2] = genetate flag, =0 for w/o generate;  >0 for generate
                if 'material' in elset[0]:
                    if elset[2] >= 1:
                        line       = lines[0]
                        line_data  = extract_elset_data_generate(line)
                        this_elset_mat_data = set(np.arange(line_data[0],line_data[1]+1,line_data[2]))

                    else:
```

```
                        this_elset_mat_data = []
                        for line in lines:
                            line_data  = extract_elset_data(line)
                            for n in line_data: this_elset_mat_data.append(n)
                        this_elset_mat_data = set(this_elset_mat_data)

                    if len(this_elset_mat_data) == 0:
                        raise ValueError('no material elset data for elset number {:d} \
in part {:s}'.format(elset[1],pname))
                    else:
                        elset_mat_data.append([elset[1],this_elset_mat_data])
        # ------------------ end checking material --------------------------

        for elset in elset_data:
            key_words = elset[0]
            if check_key not in key_words: continue

            elset_num = elset[1]

            name = '{:s}-{:d}'.format(pname,elset_num)
            if name not in names:
                names.append(name)
            else:
                raise ValueError('part {:s} has multiple {:s}-{:d} \
elset'.format(pname,check_key,elset_num))

            found_mat = False

            if not complex_part_elset:
                for t_elset in elset_data:
                    if 'material' not in t_elset[0]: continue
                    mat_num     = t_elset[1]
                    mat_data    = material_data.get(mat_num)
                    mat_name    = mat_data[0]
                    mat_density = mat_data[1]
                    found_mat   = True
                    break

                if not found_mat:
                    raise ValueError('elements in {:s}-{:d} elset of part {:s} \
is not assigned to a material'.format(check_key,elset_num,pname))


            else:
                elset_element_data = []

                for t_elset, lines in zip(elset_data,elset_lines):
                    # t_elset[0] = list of elset keywords
                    # t_elset[1] = elset num
                    # T_elset[2] = genetate flag, =0 for w/o generate;  >0 for generate
                    if check_key in t_elset[0] and elset_num == t_elset[1]:
                        if t_elset[2] >= 1:
                            line       = lines[0]
                            line_data  = extract_elset_data_generate(line)
                            elset_element_data = set(np.arange(line_data[0],line_data[1]+1,line_data[2]))
                        else:
                            for line in lines:
                                line_data  = extract_elset_data(line)
                                for n in line_data: elset_element_data.append(n)
                            elset_element_data = set(elset_element_data)
                        break

                if len(elset_element_data) == 0:
                    raise ValueError('no elset element data for {:s}-{:d} elset in part \
{:s}'.format(check_key,elset_num,pname))

                else:

                    for md  in elset_mat_data:
                        this_mat_num        = md[0]
                        this_elset_mat_data = md[1]

                        if this_elset_mat_data.intersection(elset_element_data) == elset_element_data:
                            mat_num     = this_mat_num
                            mat_data    = material_data.get(mat_num)
                            mat_name    = mat_data[0]
                            mat_density = mat_data[1]
                            found_mat   = True
                            break

                    if not found_mat:
                        raise ValueError('elements in {:s}-{:d} elset of part {:s} \
has an incorrect material assignment'.format(check_key,elset_num,pname))

            part_elset_material_data[name] = [mat_name, mat_num, mat_density]

    return part_elset_material_data

#-----------------------------------------------------------------------------
def compute_part_elset_element_data(part_names, part_lines, part_elset_data,
                                    elset_type='unknown', n_source_keywords={}):

    part_elset_element_data = {}

    if elset_type=='source' and len(n_source_keywords) != len(part_names):
        raise ValueError('mismatch length for source elset')

    for pname in part_names:
        if elset_type=='source':
            if n_source_keywords.get(pname) == 0:
                name = pname+'0'
                part_elset_element_data[name] = [0, [], pname]
                continue

        pelsetdata        = part_elset_data.get(pname)
        elset_data        = pelsetdata[1]

        if elset_type == 'source':
            check_key = 'source'
        else:
            check_key = pelsetdata[2]

        elset_lines = part_lines.get(pname)[2]

        for elset in elset_data:
            key_words = elset[0]
            if check_key not in key_words: continue

            elset_num   = elset[1]
```

```
            name        = pname+'-'+str(elset_num)

            elset_element_data = []

            for this_elset, this_line in zip(elset_data, elset_lines):
                if check_key in this_elset[0] and elset_num == this_elset[1]:
                    if this_elset[2] >= 1:
                        line       = this_line[0]
                        line_data  = extract_elset_data_generate(line)
                        elset_element_data = list(np.arange(line_data[0],line_data[1]+1,line_data[2]))
                    else:
                        for line in this_line:
                            line_data  = extract_elset_data(line)
                            for n in line_data: elset_element_data.append(n)
                    break

            n_elements = len(elset_element_data)
            part_elset_element_data[name] = [n_elements, elset_element_data, pname]

    return part_elset_element_data

#-----------------------------------------------------------------------------


#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
  Code:        abq_part_data_checking.py

  Author:      Jerawan Armstrong
               Monte Carlo Codes (XCP-3) Group
               X Computational Physics (XCP) Division
               Los Alamos National Laboratory

  Copyright (c) 2020 Triad National Security, LLC.  All rights reserved.

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in all
  copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
  SOFTWARE.
"""
import numpy as np

#-----------------------------------------------------------------------------
def check_node_element_data(part_names, part_node_data, part_element_data):
    """
    Check whether node ids in element connectivity in each part are matched
    with node ids from node data
    """
    for pname in part_names:
        node_data    = part_node_data.get(pname)
        element_data = part_element_data.get(pname)

        if node_data[1]:
            node_ids = node_data[2]
        else:
            node_ids = np.arange(1, node_data[0]+1, 1)
        node_ids = set(node_ids)

        n_els    = element_data[0]
        cons     = element_data[3]

        element_connects = set()

        for i in range(len(n_els)):
            if n_els[i] == 0: continue
            element_connects = element_connects.union(cons[i])

        if node_ids.intersection(element_connects) != element_connects:
            print("Part Name: {:s}".format(pname))
            print("List of bad node ids in element connectivity")
            #for n in element_connects:
            #    if n not in node_ids: print(n)
            raise ValueError('bad element connectivities in part {:s}'.format(pname))

#-----------------------------------------------------------------------------
def check_material_elset_data(part_names,part_data,material_data):
    material_nums = material_data.keys()
    for pname in part_names:
        pdata = part_data.get(pname)
        elset_mat_nums = pdata[5][0]
        for nm in elset_mat_nums:
            if nm not in material_nums:
                raise ValueError('material number {:d} in part {:s} is not in a material list'.format(nm,pname))

#-----------------------------------------------------------------------------
def check_element_elset_data(part_element_ids, part_elset_element_data):

    names = part_elset_element_data.keys()

    for name in names:
        elset_element_data = part_elset_element_data.get(name)
        n_elements         = elset_element_data[0]
        if n_elements == 0: continue

        element_ids        = set(elset_element_data[1])
        pname              = elset_element_data[2]

        temp_element_ids   = part_element_ids.get(pname)

        if temp_element_ids.intersection(element_ids) != element_ids:
            print('list of bad element ids:')
            for e in element_ids:
                if e not in temp_element_ids: print(e)
            raise ValueError('bad element elset in part {:s}'.format(pname))
```

Appendix B Example Abaqus Input File

```
*Heading
** Job name: block_hemi_v6 Model name: block_hemi_v6
** Generated by: Abaqus/CAE 2019
*Preprint, echo=NO, model=NO, history=NO, contact=NO
**
** PARTS
**
*Part, name=BLOCK-B
*Node
      1,          -5.,           0.,          10.
      4,         -20.,           0.,          10.
      9,          -5.,          10.,          10.
     10,          -5.,          10.,           0.
     11,         -20.,          10.,           0.
     12,         -20.,          10.,          10.
     13,         -20.,           0.,          20.
     18,          -5.,          10.,          20.
     19,          -5.,           0.,           5.
     23,         -15.,           0.,          10.
     24,         -10.,           0.,          10.
     35,          -5.,          10.,           5.
     36,         -10.,          10.,           0.
     37,         -15.,          10.,           0.
     38,         -20.,          10.,           5.
     39,         -15.,          10.,          10.
     40,         -10.,          10.,          10.
     41,         -20.,           5.,          10.
     42,          -5.,           5.,          10.
     43,         -20.,           5.,           0.
     44,          -5.,           5.,           0.
     45,         -20.,           0.,          15.
     54,         -15.,           0.,          20.
     55,         -20.,           5.,          20.
     56,         -20.,          10.,          15.
     57,          -5.,           5.,          20.
     58,          -5.,          10.,          15.
     59,         -10.,          10.,          20.
     60,         -15.,          10.,          20.
     61,         -10.,           0.,           5.
     62,         -15.,           0.,           5.
     71,         -10.,          10.,           5.
     72,         -15.,          10.,           5.
     73,         -10.,           5.,          10.
     74,         -15.,           5.,          10.
     75,         -10.,           5.,           0.
     76,         -15.,           5.,           0.
     77,          -5.,           5.,           5.
     78,         -20.,           5.,           5.
     83,         -15.,           0.,          15.
     84,         -10.,           0.,          15.
     87,         -20.,           5.,          15.
     88,          -5.,           5.,          15.
     89,         -15.,           5.,          20.
     90,         -10.,           5.,          20.
     91,         -15.,          10.,          15.
     92,         -10.,          10.,          15.
     95,         -10.,           5.,           5.
     96,         -15.,           5.,           5.
     99,         -15.,           5.,          15.
    100,         -10.,           5.,          15.
*Element, type=C3D8R
13,   1,  19,  61,  24,  42,  77,  95,  73
15,  24,  61,  62,  23,  73,  95,  96,  74
20,  77,  44,  75,  95,  35,  10,  36,  71
21,  73,  95,  96,  74,  40,  71,  72,  39
24,  96,  76,  43,  78,  72,  37,  11,  38
37,  23,  83,  99,  74,   4,  45,  87,  41
38,  83,  54,  89,  99,  45,  13,  55,  87
39,  74,  99,  91,  39,  41,  87,  56,  12
41,  24,  84, 100,  73,  23,  83,  99,  74
44, 100,  90,  59,  92,  99,  89,  60,  91
47,  42,  88,  58,   9,  73, 100,  92,  40
48,  88,  57,  18,  58, 100,  90,  59,  92
*Elset, elset=SET_STATISTIC_004
 13, 15, 20, 21, 24
*Elset, elset=SET_STATISTIC_001
 37, 38, 39, 41, 44, 47, 48
*Elset, elset=SET_MATERIAL_001
 13, 15, 20, 21, 24, 37, 38, 39, 41, 44, 47, 48
** Section: Section-1-SET_MATERIAL_001
*Solid Section, elset=SET_MATERIAL_001, material=ALUMINUM_001
,
*End Part
**
*Part, name=HEMI-A
*Node
      9,         -15.,           0.,           0.
     10,         -20.,           0.,           0.
     65,  -14.2658482,           0.,   4.63525486
     66,  -12.1352549,           0.,   8.81677914
     67,  -8.81677914,           0.,   12.1352549
     69,        -17.5,           0.,           0.
     70,  -19.0211296,           0.,   6.18033981
     71,  -16.1803398,           0.,   11.7557049
     78,  -14.4888878,   3.88228559,           0.
     79,  -12.9903812,          7.5,           0.
     83,  -19.3185158,   5.17638111,           0.
    191,   -16.651104,           0.,   5.41421127
    192,  -14.1600876,           0.,   10.2832737
    193,  -10.2832737,           0.,   14.1600876
    218,  -9.44908714,   4.87010145,   10.5828571
    220,  -11.0603456,   7.77711439,   6.49501753
    221,  -12.1539965,   4.35719395,   7.63513088
    222,  -13.8446913,   4.12033272,    4.0431881
    223,  -12.4165096,   7.64765692,   3.51334953
    227,  -16.9026642,   4.53362846,           0.
    228,  -15.1577482,   8.74622917,           0.
    232,  -15.9588022,    5.7864871,   10.5751219
    233,  -18.3967533,   5.52291918,   5.57286406
```

```
    234,  -14.4478245,   10.4146786,   9.09916687
    235,  -16.4513702,   10.2669563,   4.89305735
    239,  -12.3548412,   6.48197842,   14.3297539
    241,  -11.2937136,   11.4542313,   11.8849745
    294,  -13.3110504,   4.72851801,   8.81317425
    295,    -12.15802,   8.63381863,    7.4945159
    301,   -9.3094511,   9.32660103,   9.97525501
    302,  -10.0414009,   5.15912437,   12.0751715
    303,  -15.4972162,   4.57867861,   4.65281343
    304,  -13.8886414,   8.59171391,   4.04228067
*Element, type=C3D8R
147, 302, 193, 192, 294, 218,  67,  66, 221
149, 191,  69, 227, 303,  65,   9,  78, 222
150, 294, 303, 304, 295, 221, 222, 223, 220
151, 303, 227, 228, 304, 222,  78,  79, 223
160, 232, 233, 303, 294,  71,  70, 191, 192
161, 233,  83, 227, 303,  70,  10,  69, 191
162, 234, 235, 304, 295, 232, 233, 303, 294
171, 301, 241, 234, 295, 302, 239, 232, 294
*Elset, elset=SET_STATISTIC_012
 147, 149
*Elset, elset=SET_STATISTIC_023
 150, 171
*Elset, elset=SET_STATISTIC_034
 151, 162
*Elset, elset=SET_STATISTIC_011
 160, 161
*Elset, elset=SET_MATERIAL_001
 147, 149
*Elset, elset=SET_MATERIAL_1008
 150, 151, 160, 161, 162, 171
** Section: Section-2-SET_MATERIAL_001
*Solid Section, elset=SET_MATERIAL_001, material=ALUMINUM_001
,
** Section: Section-3-SET_MATERIAL_1008
*Solid Section, elset=SET_MATERIAL_1008, material=WATER_1008
,
*End Part
**
*Part, name=HEX_WEDGE1
*Node
      1,           0.,   23.8814831,           0.
      2,           0.,   25.8742371,           0.
      3,   1.09208882,   23.5859795,    4.0757308
      4,  0.549050272,   23.7657948,   2.04908371
      5,   1.09813011,   25.5555992,   4.09827709
      6,  0.552914023,   25.7522068,   2.06350327
      7,  0.823185027,   23.5859795,    4.1384306
      8,  0.413858265,   23.7657948,   2.08060598
      9,  0.827738762,   25.5555992,   4.16132355
     10,  0.416770637,   25.7522068,   2.09524751
     11,  0.550756216,   23.5859795,   4.18340874
     12,  0.276894003,   23.7657948,   2.10321879
     13,  0.553802907,   25.5555992,    4.2065506
     14,  0.278842568,   25.7522068,   2.11801958
     15,  0.275968969,   23.5859795,   4.21047306
     16,  0.138744071,   23.7657948,   2.11682534
     17,  0.277495593,   25.5555992,   4.23376465
     18,  0.139720425,   25.7522068,   2.13172174
     19,           0.,   23.5859795,   4.21950722
     20,           0.,   23.7657948,   2.12136745
     21,           0.,   25.5555992,   4.24284887
     22,           0.,   25.7522068,    2.1362958
*Element, type=C3D8R
1,  7,  8, 10,  9,  3,  4,  6,  5
2, 11, 12, 14, 13,  7,  8, 10,  9
3, 15, 16, 18, 17, 11, 12, 14, 13
4, 19, 20, 22, 21, 15, 16, 18, 17
*Element, type=C3D6
5,  2,  6, 10,  1,  4,  8
6,  2, 10, 14,  1,  8, 12
7,  2, 14, 18,  1, 12, 16
8,  2, 18, 22,  1, 16, 20
*Elset, elset=SET_MATERIAL_001, generate
 1,  8,  1
*Elset, elset=SET_STATISTIC_002, generate
 1,  8,  1
** Section: Section-4-SET_MATERIAL_001
*Solid Section, elset=SET_MATERIAL_001, material=ALUMINUM_001
,
*End Part
**
*Part, name=TET_ONLY
*Node
      1,           0.,           5.,          10.
      2,          -3.,           5.,          10.
      3,           0.,           5.,           5.
      4,          -3.,           5.,           5.
      5,          -6.,           5.,           5.
      6,          -6.,          2.5,          10.
      7,          -3.,          2.5,          10.
      8,           0.,           5.,          7.5
      9,          -3.,           5.,          7.5
     10,           0.,          2.5,           5.
     11,          -3.,          2.5,           5.
     12,          -6.,          2.5,           5.
     13,   1.25195348,    2.3700695,   7.90644789
     14,  -1.88319409,   1.72202432,   7.78357697
     15,  -4.05866098,    1.7532028,   7.15756607
*Element, type=C3D4
1, 15,  2, 14,  9
2, 14, 10,  4,  3
3, 14, 11,  4, 10
4, 15, 11, 12,  5
5,  9,  4, 14, 15
6,  8, 13,  1, 14
7,  8,  3, 13, 14
8, 15,  6,  7,  2
*Elset, elset=SET_STATISTIC_001, generate
 1,  8,  1
*Elset, elset=SET_MATERIAL_001, generate
 1,  8,  1
** Section: Section-5-SET_MATERIAL_001
*Solid Section, elset=SET_MATERIAL_001, material=ALUMINUM_001
,
*End Part
**
**
** ASSEMBLY
```

```
**
*Assembly, name=Assembly
**
*Instance, name=HEMI-A-1, part=HEMI-A
*End Instance
**
*Instance, name=BLOCK-B-1, part=BLOCK-B
          0.,           0.,         -25.
*End Instance
**
*Instance, name=HEX_WEDGE1-2, part=HEX_WEDGE1
         -5.,          -5.,           0.
*End Instance
**
*Instance, name=TET_ONLY-1, part=TET_ONLY
         10.,          15.,         -15.
         10.,          15.,         -15.,          10.,          15., -13.9999999873119, 89.9999992730282
*End Instance
**
*Instance, name=HEX_WEDGE1-1, part=HEX_WEDGE1
          0.,           0.,         -22.
          0.,           0.,         -22.,           0., 1.00000001268805,         -22., 89.9999992730282
*End Instance
**
*End Assembly
**
** MATERIALS
**
*Material, name=ALUMINUM_001
*Density
 2.7,
*Material, name=WATER_1008
*Density
1.,
```