

Table of Contents

1. Overview of the project.
2. Pre-requisites to begin with.
3. Setting up the working directory.
4. Code execution.
5. UEL and Input file.
6. Post-processing and Results.
7. Conclusion and Outlook.
8. References and Important Links

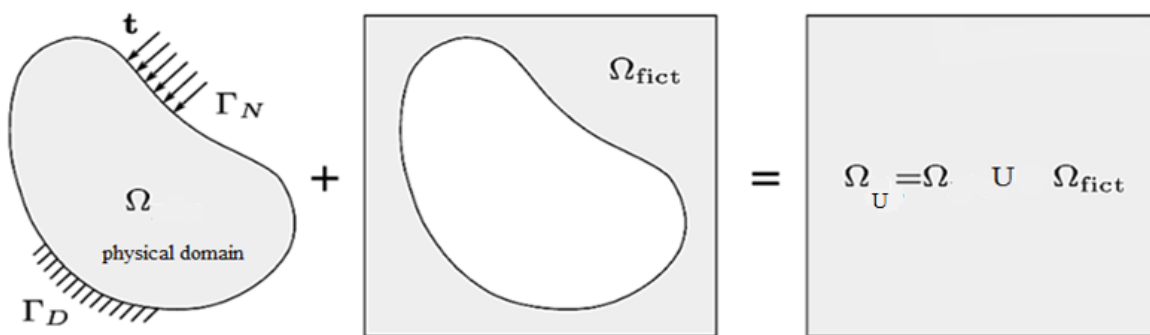
1.3 CAD Modelling

First, a CAD model is made for the geometry under consideration. It is then exported as an ‘stl’ (stereolithography) file. This file acts as a user input for the python script.

1.4 Python Script:

The Stl file (which has geometry in the form of triangulated mesh) is imported in python. Python library **Shapely** is used to recover the original geometry out of triangulated mesh and an outer bounding box is made.

This bounding box acts as the extended domain Ω_u (physical domain + fictitious domain) of FCM.



Concept of extended domain Ω_u in FCM

Then a structured mesh is generated using the extended domain and the user defined mesh parameters. Now, inside-outside test is performed to identify three sets of the cells (equivalent to elements in FEM):

1. Completely inside the physical domain (set A)
2. Completely outside the physical domain (set B)
3. Cut by the physical boundary (cut cells) (set C)

A finite stiffness value is assigned to set A which depends upon the type of material (steel, aluminum etc.). A very small stiffness value $\epsilon \approx 0$ is assigned to set B. Cells of set C need special treatment. Stiffness values for these cut cells are computed by using adaptive integration with the help of quadtree. After the execution of python script, stiffness matrices for all the cut cells are generated in the form of a text file.

1.5 UEL

UEL (user element subroutine) is written in Fortran for the cells of set C (cut cells). This UEL acts as user defined element with the stiffness value previously calculated by using adaptive integration.

1.6 Input file (.inp)

The information from python script about the element identities of cut cells, completely inside or outside cells is used in .inp file. Proper boundary conditions are applied on the respective nodes. Here actually, plane stress elements from abaqus standard library are used for set A and set B. For set C, UEL subroutine is called. This input file along with the UEL is run in abaqus for solution. After solution “.odb” file is generated which is then converted into “.vtk” file. This is then postprocessed in Paraview because abaqus cannot postprocess the UEL based simulations.

2. Prerequisites

Before beginning to implement a user defines UEL Subroutine in ABAQUS, and running Python codes to perform several tasks, it is important to install all the relevant software packages and compilers. In this section, it will be explained what software sand their versions were used in the project, and how to link the Fortran compiler to ABAQUS in order to efficiently run the user sub-routines.

The requirements are listed as follows:

1. ABAQUS

In order to run the user subroutines in ABAQUS, the “SIMULIA Academic Research Suite”, also known as the “Research License” package is required. It offers commercial level functionality for advanced research. The ABAQUS version used in this project was ABAQUS/CAE 2018.

2. Python Version

Since ABAQUS has its own Python Development Environment (PDE), the version of Python used in this project is restricted to the one used by ABAQUS. As ABAQUS/CAE 2018 uses Python version 2.7.3, one has to stick with the same version while executing the codes. On the other hand, the choice of the IDE is user dependent. For this project SPYDER IDE was used.

3. Fortran Compiler

As stated earlier, in order to execute the UEL, which is nothing but a Fortran code, ABAQUS needs a Fortran compiler linked to it. User must be careful while getting a Fortran compiler and linking it to ABAQUS, as the compatibility criteria between the versions have to be met. In this project for ABAQUS/CAE 2018 Intel® Visual Fortran Compiler (IFORT)16.0 was used, which is a part of the Intel® Parallel Studio XE 2016.

4. Fortran IDE

In order to get familiar with Fortran, an IDE was needed which in this case is the Visual Studio 2015. Again, user must be careful while linking the Fortran compiler and the IDE due to the compatibility issue. For this project IFORT 16.0 was linked to Visual Studio 2015.

2.1 Linking ABAQUS to Fortran Compiler

This is one of the most important part of the whole process of executing UEL in ABAQUS, as without this step it is not possible for ABAQUS to identify and compile the UEL written in Fortran. In order to enable ABAQUS to compile and link UEL and other user subroutines, it must be made aware of the locations of both the Intel® Fortran compiler as well as the MS Visual Studio software on the host windows machine. User must be mindful while performing this linkage, about the version compatibility of ABAQUS version with the Fortran compiler.

In this project ABAQUS/CAE 2018 was linked with Intel® Visual Fortran Compiler 16.0 in conjunction with MS Visual Studio 2015.

Another important point to note while running the ABAQUS UEL is including the file “ABA_PARAM.INC”, which is installed on the host machine by ABAQUS installation procedure. The file specifies IMPLICIT REAL *8 (A-H, O-Z) for double precision machine or IMPLICIT REAL (A-H, O-Z) for single precision machines. The ABAQUS execution procedure which compiles and links the use subroutine with the rest of ABAQUS, will include the ABA_PARAM.INC automatically. It is not necessary to find this file and copy it to any particular directory, ABAQUS will know where to find it. Every user subroutine in ABAQUS must include the statement, **INCLUDE ‘ABA_PARAM.INC’**

3. Setting up the Working Directory

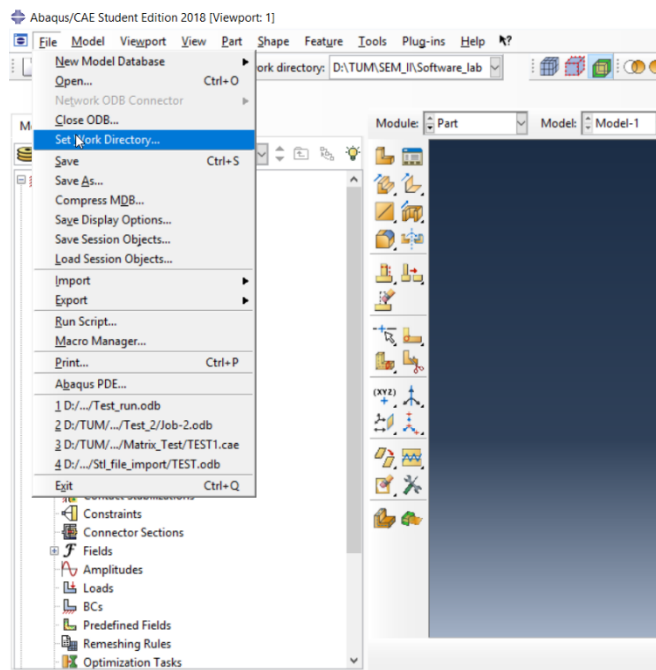
The appropriate directories need to be set up in order to run the simulation. All the files are organized in the following folders:

- Python Scripts: Contains all the python scripts
- Stl files: Contains all the .stl files of the geometry
- Data files: Contains all the data files in .txt files
- Input File and UEL: Contains the input file, Fortran script for user subroutine and other files for UEL execution in Abaqus

3.1 Setting work directory in Abaqus CAE

The working directory of Abaqus needs to be set to the folder “Python Scripts”

File-> Set Work Directory

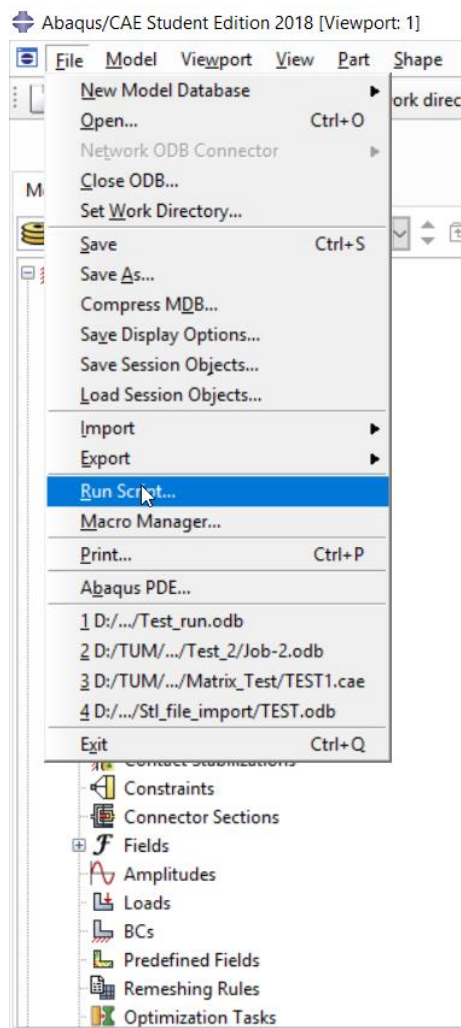


3.2 Directory requirements for running UEL

The UEL is run through the command prompt window. The command prompt window needs to be launched from the folder “*Input File and UEL*”.

3.3 Running the script

File-> Run Script -> “Abaqus_main.py”



4. Code Execution

The code is organized into two parts

1. Code to be executed in ABAQUS CAE (**Abaqus_main.py**)
2. Code to be executed in Python Standalone (**FCM_Main.py**)

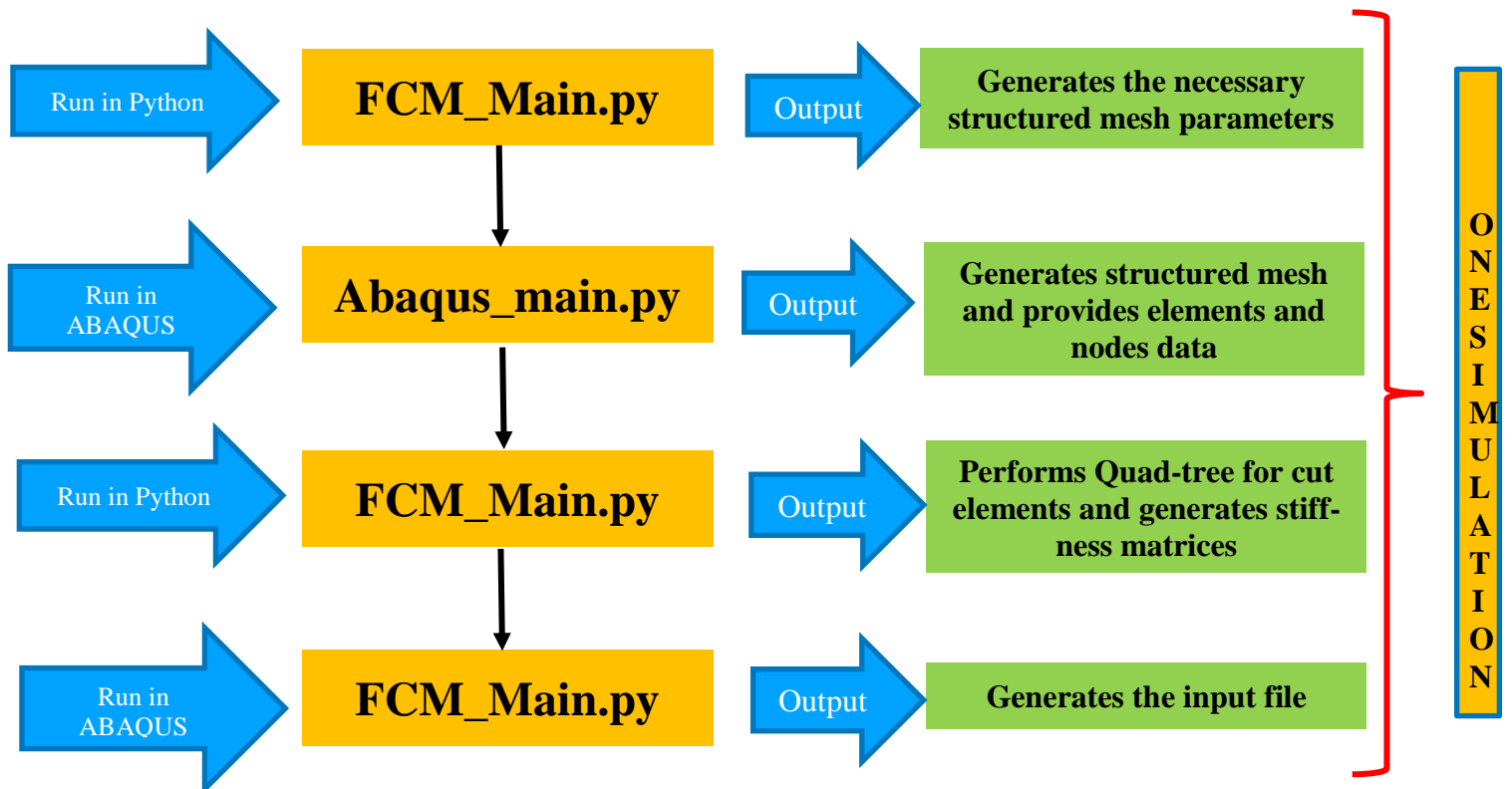
Execute the scripts in the following order:

Step-1: Execute FCM_Main.py in **Python**

Step-2: Execute Abaqus_main.py in **ABAQUS CAE**

Step-3: Execute FCM_Main.py in **Python**

Step-4: Execute Abaqus_main.py in **ABAQUS CAE**



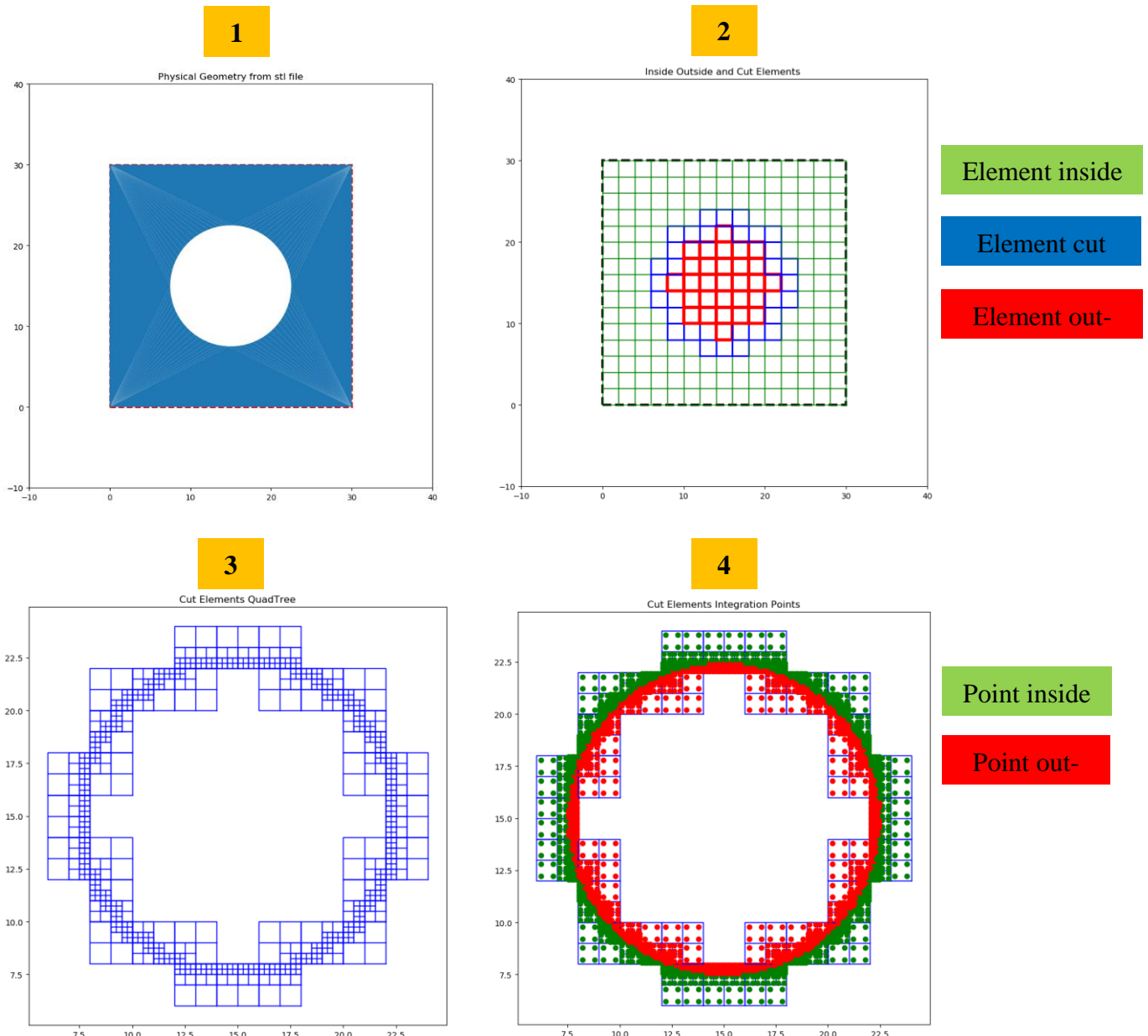
4.1 Checkpoints before starting the simulation:

- Ensure the directory in ABAQUS CAE is set to the folder containing the “*Abaqus_main.py*”
- Delete the text file “*Stiffness_matrix_Element.txt*” from the folder “Input File and UEL” prior to every simulation

4.2 Output after running scripts

In order to check the correct execution of the code, messages are displayed in the console indicating the various steps. In addition, plots are generated to indicate the correct execution. The following four plots are generated:

1. The physical geometry visualized from the provided stl file.
2. Classification of the elements in the structured mesh as inside, outside and cut.
3. Quad-tree algorithm performed for the cut cells.
4. Integration points distribution over each cut element with inside/outside test performed for each integration point.



4.3 UEL Execution

- After running the “*Abaqus_main.py*” for the second time, an input file called “*Test_run.inp*” is generated in the folder “*Input File and UEL*”.
- This is the input file, which needs to be modified. The modifications to be done are described in the section below.
- The UELsub-routine is a Fortran script and can be found in the folder “*Input File and UEL*” under the name “*User_Subroutine.f*”
- A counter file called “*counter.txt*” is automatically generated in the folder “Input File and UEL”.
- The input file along with the user subroutine is run in Abaqus using the command-prompt (cmd).

5 User Element Subroutine (UEL) and Input file

To run the UEL user has to go into the folder “Input File and UEL” and make sure the below mentioned files are present.

1. Counter.txt
2. Read Me.txt
3. Stiffness_matrix_Element.txt
4. Test_run.inp
5. User_Subroutine.f

5.1 Abaqus input file (Test_run.inp)

Abaqus solver reads the input file for the Analysis. We need it for our User element subroutine (UEL) to work. In an input file Keyword lines introduce options and often have parameters, which appear as words or phrases separated by commas on the keyword line. Parameters are used to define the behavior of an option. There are many properties that can be specified in an input file

In an input file Each keyword line starts with a star (*) followed by data lines.

5.2 Standard Input file vs the modified one

We have to make some changes in our input file to make our UEL code work. We need to specify the elements which would be using our own defined elements. We need to use *USER ELEMENT command to incorporate user defined elements. The command line that was used in our input file reads as follows:

```
*USER ELEMENT, TYPE=U100, NODE=4, PROPERTIES=2, COORDINATES=2  
  
1, 2
```

Here NODE describes the number of nodes in our element, Properties for the different characteristics that we need to give for our defined element and then we have to have specify the number of active degrees of freedom, which is only displacements in x and y directions and hence specified and 1,2.

Figure below shows the modified input file :

```

*USER ELEMENT, TYPE=U1, NODE=4, PROPERTIES=2 ,COORDINATES=2
1,2
*ELEMENT, TYPE=U1, ELSET=Cut
24 , 26, 27, 38, 37
25, 27, 28, 39, 38
26, 28, 29, 40, 39
27, 29, 30, 41, 40
33, 36, 37, 48, 47
34, 37, 38, 49, 48
37, 40, 41, 52, 51
38, 41, 42, 53, 52
43, 47, 48, 59, 58
48, 52, 53, 64, 63
53, 58, 59, 70, 69
58, 63, 64, 75, 74
63, 69, 70, 81, 80
64, 70, 71, 82, 81
67, 73, 74, 85, 84
68, 74, 75, 86, 85
74, 81, 82, 93, 92
75, 82, 83, 94, 93
76, 83, 84, 95, 94
77, 84, 85, 96, 95

```

After describing our user element, we have to specify the number of elements that will incorporate this user element by the command *ELEMENT as shown in the above figure.

5.3 USER ELEMENT SUBROUTINE (UEL)

UEL is used to create one's own element with desired material properties. Fortran script is used to write such elements. First part of it contain the arguments, inputs and outputs. The second part contains the dimensions of the variables. The third part is the user defined code to calculate different things.

```

c*****|*****
      SUBROUTINE UEL (RHS, AMATRX, SVARS, ENERGY, NDOFEL, NRHS, NSVARS,
& PROPS, NPROPS, COORDS, MCRD, NNODE, U, DU, V, A, JTYPE, TIME,
& DTIME, KSTEP, KINC, JELEM, PARAMS, NDLOAD, JDLTYP, ADLMAG,
& PREDEF, NPREDF, LFLAGS, MLVARX, DDLMAG, MDLOAD, PNEWDT, JPROPS,
& NJPRO, PERIOD)
      INCLUDE 'ABA_PARAM.INC'
      DIMENSION RHS(MLVARX,*), AMATRX(NDOFEL,NDOFEL), COORDS(MCRD, NNODE)
      integer c, ilines, temp

```

Figure above shows a part of the UEL that was written for our element in Abaqus.

For a UEL to work, one needs to specify **RHS**, **AMATRX**, **SVARS**, **ENERGY**, and **PNEWDT**.

But for static, small displacement analysis, only RHS and AMATRX need to be defined.

In our written code, we are initializing the RHS vector with the zero as shown:

```

c*****INITIALISE RHS AND LHS:-*****
      DO K1 = 1, NDOFEL
        RHS(K1,1) = 0.0
        DO K2 = 1,NDOFEL
          AMATRX(K1,K2) = 0.0
        END DO
      END DO

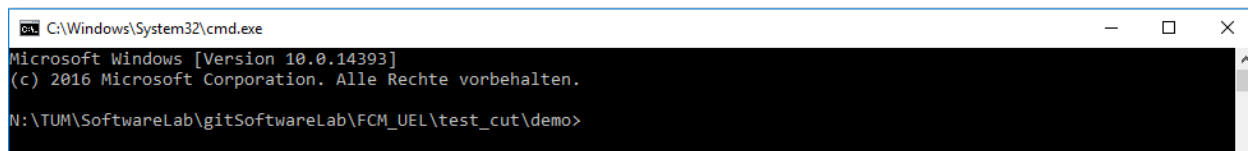
```

The value of AMATRX is read from the text file “*Stiffness_matrix_Element.txt*”

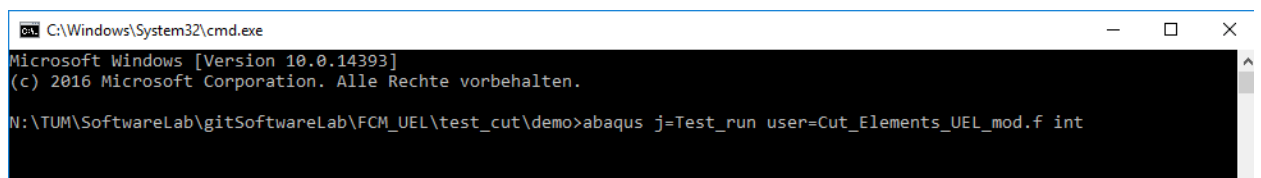
Execution of Input and UEL file.

To execute the UEL following steps are required.

1. Open the command prompt and set the working directory containing all the required files:



2. Then we have to write the command in following way. The **Cut_Elements_UEL.f** file is the Fortran code of our UEL and **Test_run** is our input file while **Abaqus j** is a command to make it work in Abaqus.



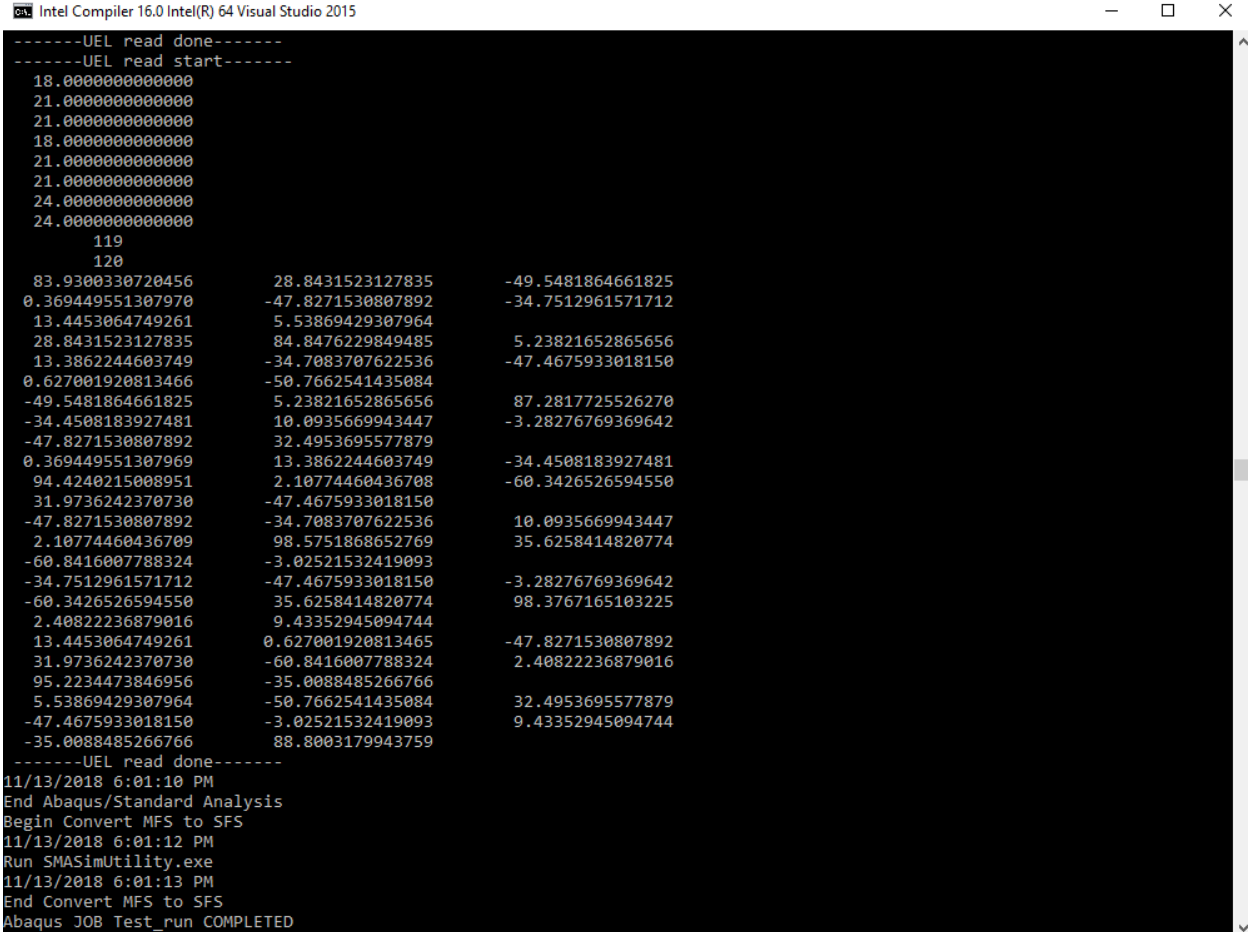
3. After the command has been executed, Abaqus will perform all the calculations and will generate a set of output files in the same folder as the input file

6 Post processing

(**Caution:** It is implemented under the branch -PythonQuadTreeShapely)

After job is submitted to Abaqus, and solved with UEL subroutines, it will generate a .odb file. It is the Abaqus output database. Besides, some log files like .dat .msg will also be generated, in which you could find the record of this submission. It is possible that it would come along with error and warning records. It is a good way to debug, if the submission fails.

In addition, in order to detect any flaws in Fortran code, we could also plug some debuggers into the code, such as printing out the coordinates and stiffness of the element being called. Or display the counter index, even as simple as like “Hello world!” would be possible.



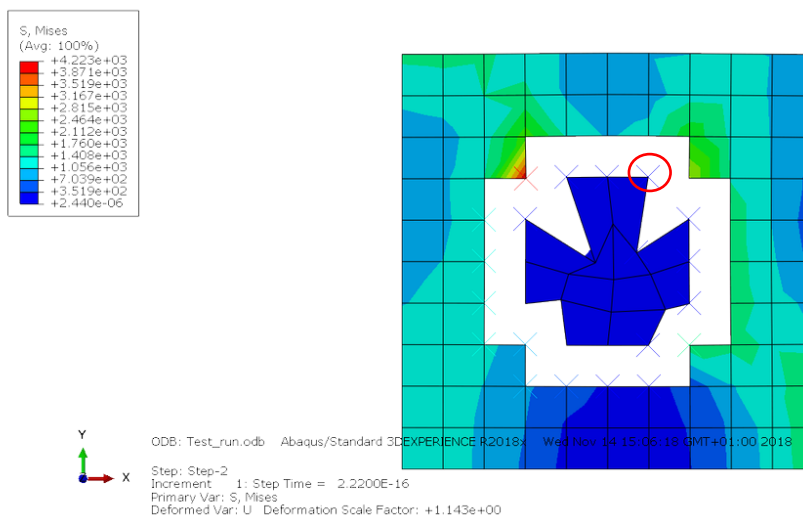
```
Intel Compiler 16.0 Intel(R) 64 Visual Studio 2015
-----UEL read done-----
-----UEL read start-----
18.000000000000
21.000000000000
21.000000000000
18.000000000000
21.000000000000
21.000000000000
24.000000000000
24.000000000000
119
120
83.9300330720456      28.8431523127835      -49.5481864661825
0.369449551307970     -47.8271530807892     -34.7512961571712
13.4453064749261      5.53869429307964
28.8431523127835      84.8476229849485      5.23821652865656
13.3862244603749      -34.7083707622536     -47.4675933018150
0.627001920813466     -50.7662541435084
-49.5481864661825      5.23821652865656      87.2817725526270
-34.4508183927481      10.0935669943447      -3.28276769369642
-47.8271530807892      32.4953695577879
0.369449551307969      13.3862244603749      -34.4508183927481
94.4240215008951      2.10774460436708      -60.3426526594550
31.9736242370730      -47.4675933018150
-47.8271530807892     -34.7083707622536      10.0935669943447
2.10774460436709      98.5751868652769      35.6258414820774
-60.8416007788324      -3.02521532419093
-34.7512961571712     -47.4675933018150     -3.28276769369642
-60.3426526594550      35.6258414820774      98.3767165103225
2.40822236879016      9.43352945094744
13.4453064749261      0.627001920813465     -47.8271530807892
31.9736242370730      -60.8416007788324      2.40822236879016
95.2234473846956      -35.0088485266766
5.53869429307964      -50.7662541435084      32.4953695577879
-47.4675933018150     -3.02521532419093      9.43352945094744
-35.0088485266766      88.8003179943759
-----UEL read done-----
11/13/2018 6:01:10 PM
End Abaqus/Standard Analysis
Begin Convert MFS to SFS
11/13/2018 6:01:12 PM
Run SMASimUtility.exe
11/13/2018 6:01:13 PM
End Convert MFS to SFS
Abaqus JOB Test_run COMPLETED
```

Abaqus

After submission, you would get,

- Result.odb : The result database. It is possible to access it through Abaqus or python.
- Result.dat : The Log file records how the job is submitted.
- Result.msg : The Log file regards the solver.

The database could be simply opened in Abaqus, however, those user defined elements, solved by means of user subroutine, would not be able to be shown in Abaqus post processor. Instead, you would only see a **cross** on only one of the nodes of the user element as indicated in the figure below.



It would be much better, if we could show the result of user elements. Therefore, we could try to show the result using other 3rd party software, such as **Paraview**. Nevertheless, it would be then necessary to convert the .odb data into the form that Paraview could handle.

In Abaqus, we need these libraries to extract and convert the .odb database

odbAccess

- `openOdb(...)` : It is a useful Abaqus standard library that you could get access to the database. It is object-oriented structure. That means the solutions could be called out in a very intuitive way, such as, “`element[i].connectivity`”. However, you could not see what is the mechanism inside and some of the attributes are read-only.

odb2vtk

- . It is a library from the internet that could convert .odb to vtk (3D element).
 - `parameter.txt` : parameter file to set directory and define what frames and steps we need

odb2vtkCell

- . This library is modified to adapt our model. It could also deal with 2D element by topology.
 - **fixConnectivity** : In this script, you would find how we fix the connectivity of user defined elements in order to convert .odb to .vtk. It is still unclear why user elements will lose connectivity during the solution of Abaqus. It is also the reason that we could only see a cross on a node of user defined elements. The element connectivity is shown below. It can be seen that certain elements have no nodes associated with them. All such elements are nothing but the user defined elements

```
(95, 96, 107, 106)
(96, 97, 108, 107)
(97, 98, 109, 108)
(98, 99, 110, 109)
(100, 101, 112, 111)
(101, 102, 113, 112)
(102, 103, 114, 113)
(103, 104, 115, 114)
(104, 105, 116, 115)
(105, 106, 117, 116)
(106, 107, 118, 117)
(107, 108, 119, 118)
(108, 109, 120, 119)
(109, 110, 121, 120)
(26,)
(27,)
(28,)
(29,)
(36,)
(37,)
(40,)
```

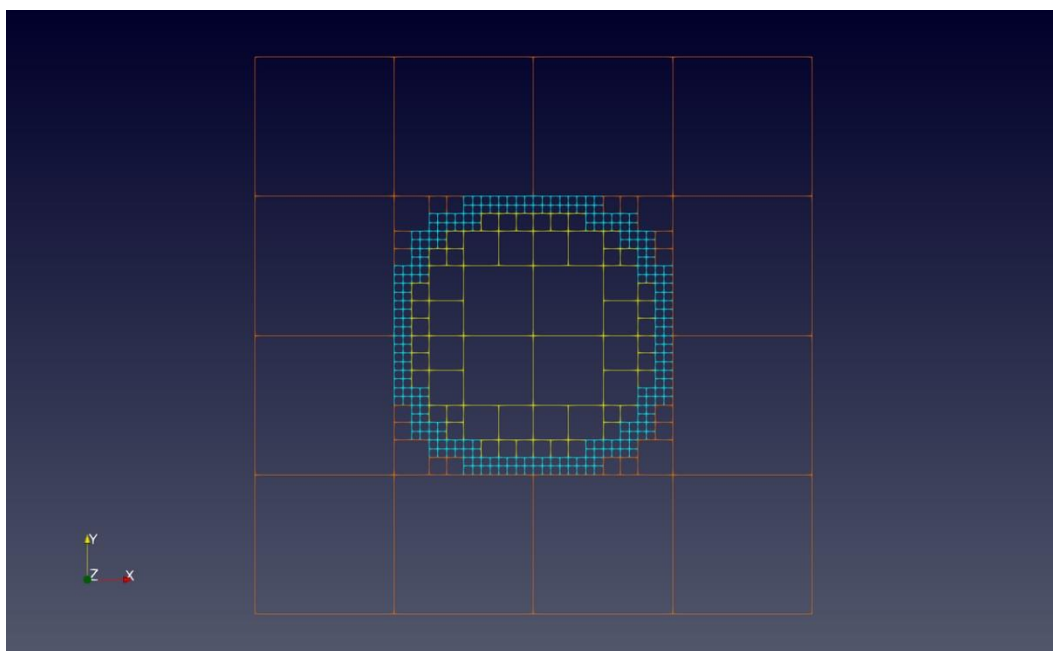


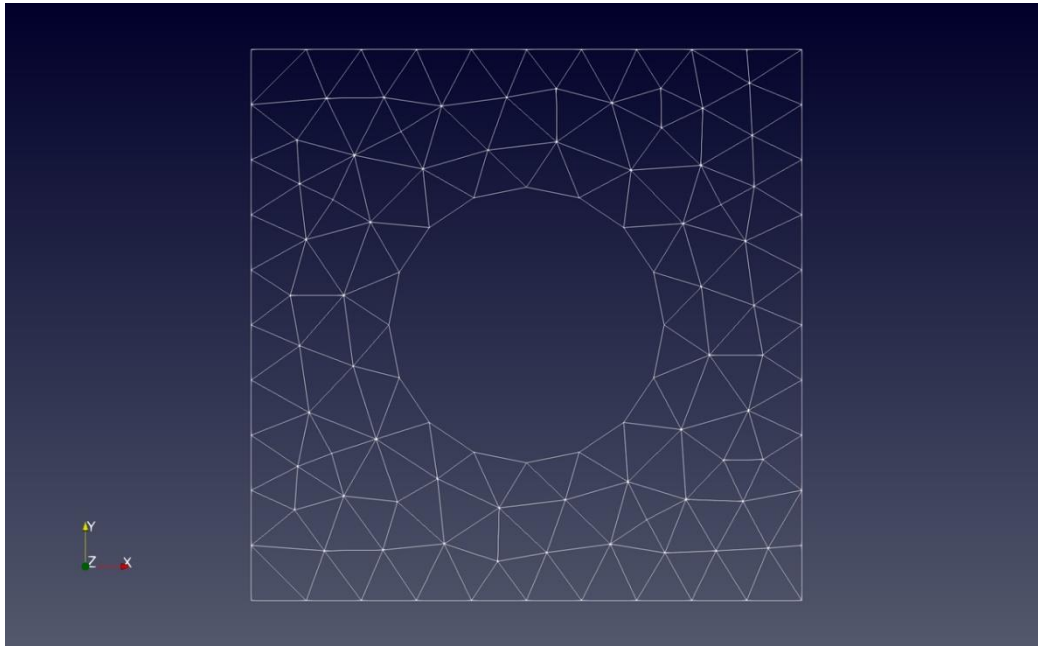
```
>>> from odb2vtk import *
>>> from odb2vtkCell import *
>>> ConvertOdb2Vtk()
C:\Users\ga53vak\Desktop\Post
Test_run
odb2vtk reading finished, time elapsed: 0.0
Basic Information:
Model: Test_run ; Mesh type: Tetra ; Number of blocks: 1
Convert frames: 0 to 1
Step & Instance : ['1'] , ['0']
ODB opened
Step: Step-2
Instance: ORPHAN_MESH_ASSEMBLY
Frame: 0
debug
Reading U, A, V, RF .....
Time elapsed: 0.00699996948242 s
Reading Stress .....
Time elapsed: 0.0220000743866 s
Reading Strain components.....
Time elapsed: 0.0199999809265 s
Partitionning model and writing vtk files .....
frame: 0; block: 0
```

One thing has to be mentioned is that it could only be done in the Python environment of Abaqus, because it would need some library that is only inside Abaqus.

Post processing - Paraview

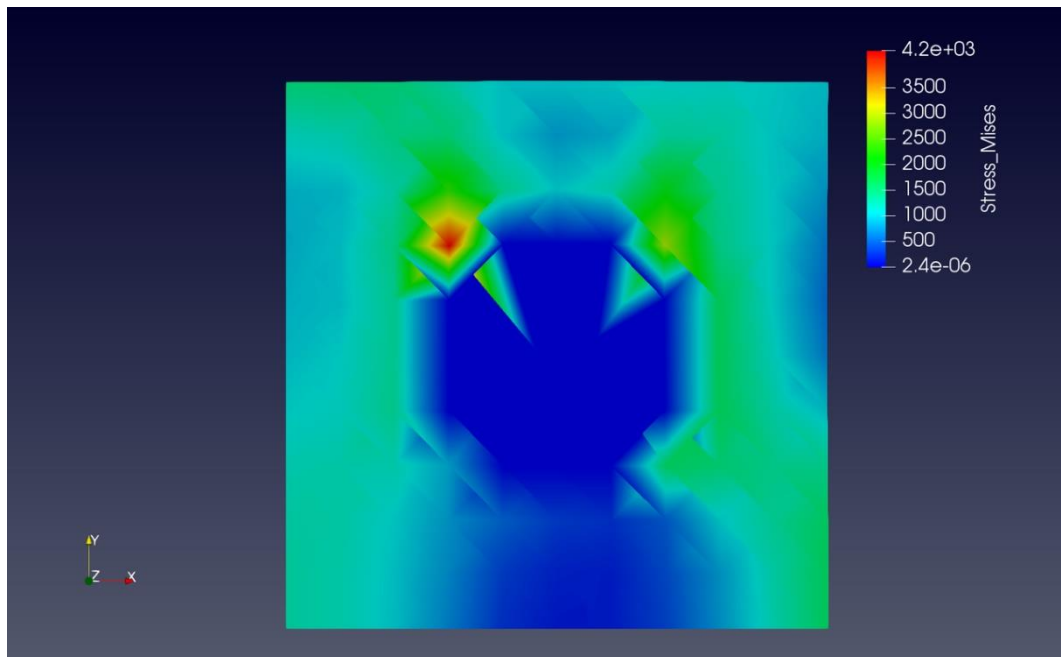
- . Under the folder “Post”, you would find files that are for post processing.
 - vtk : Here, you would find basic geometric information in vtk form.
 - stl : Here, you would find the mesh map that we use for interpolation to show result on physical domain

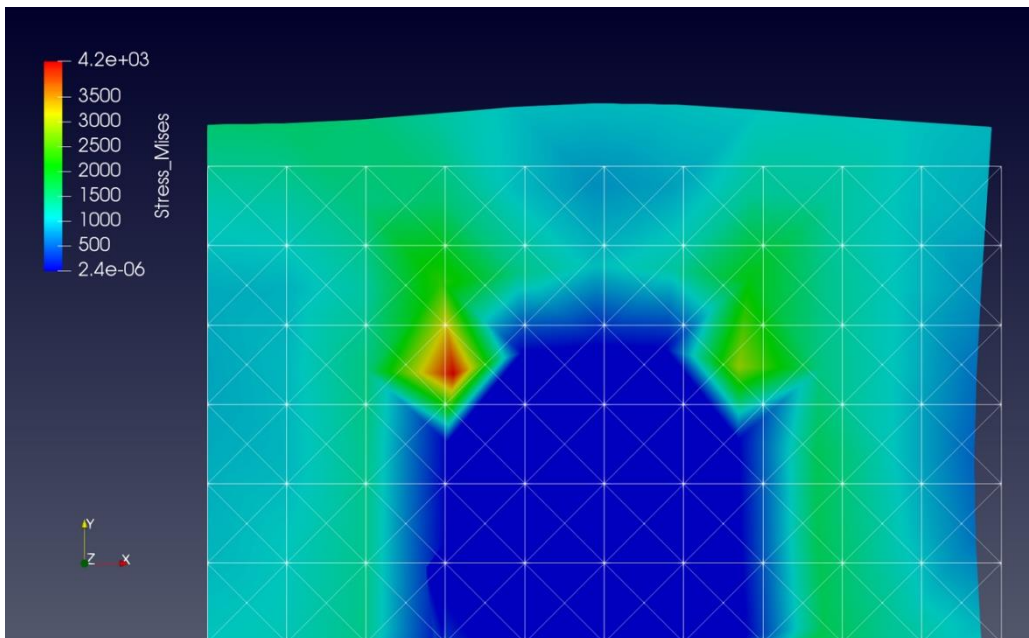
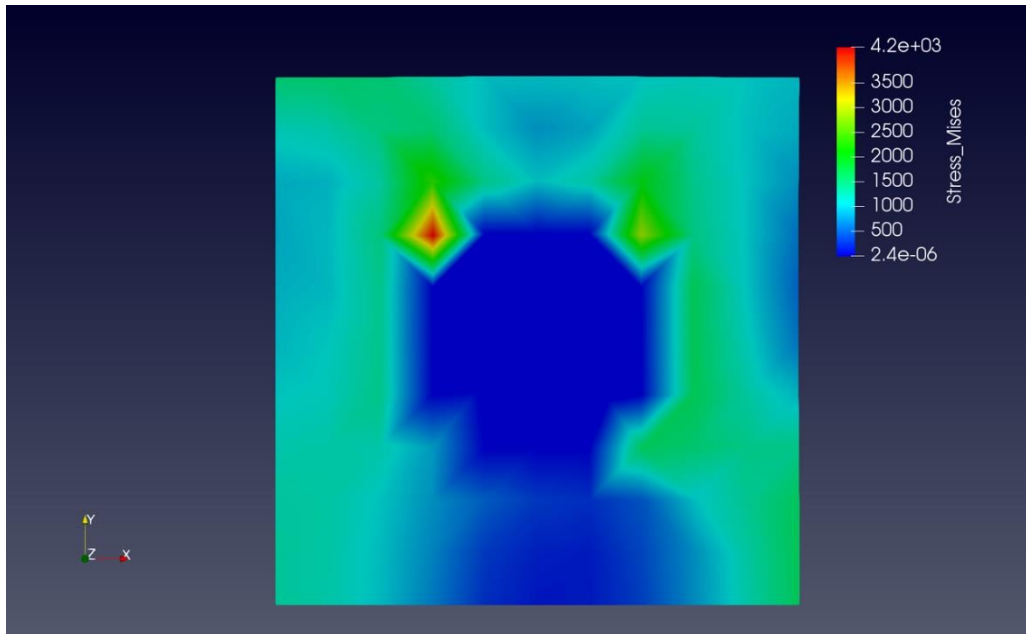




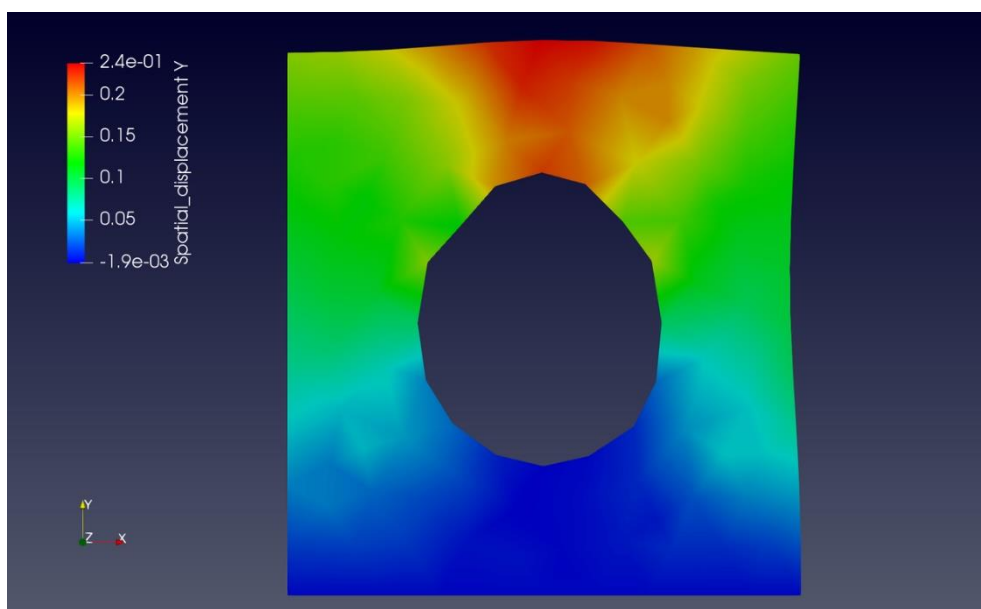
- **vtu**
 - Frame 0 : Initial geometry of the model
 - Frame 1 : static load

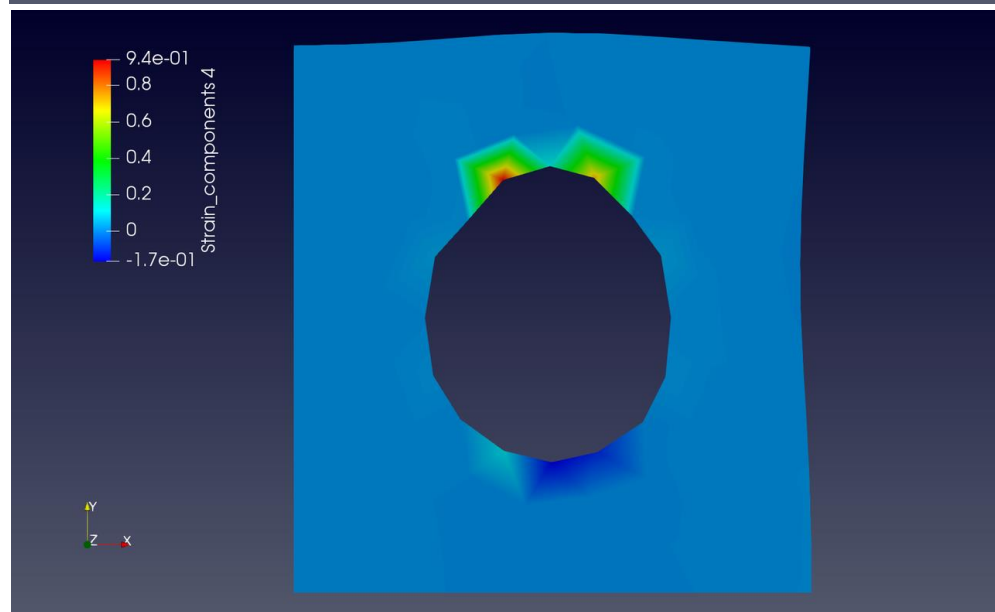
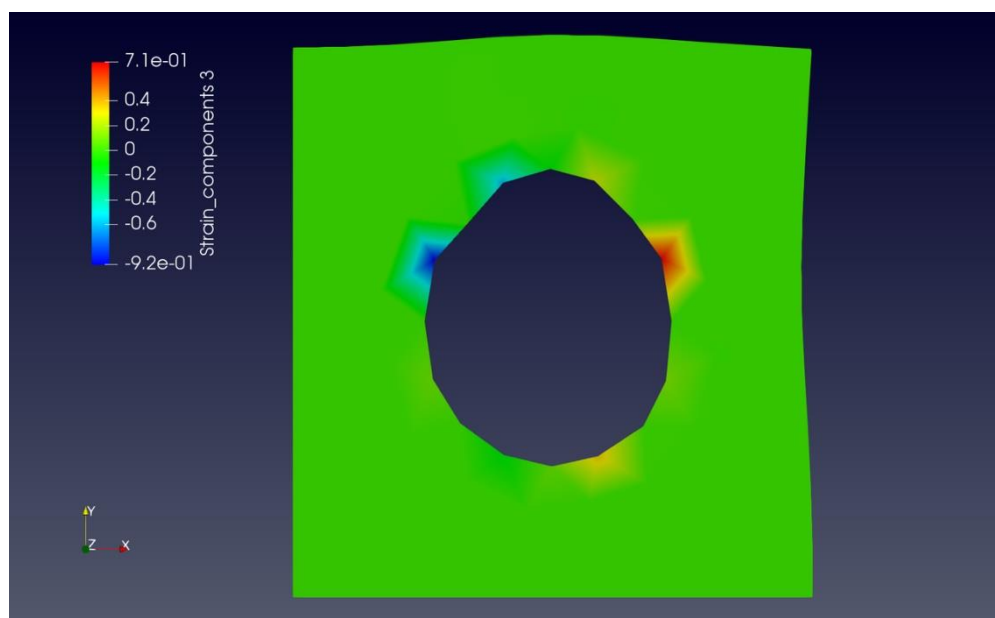
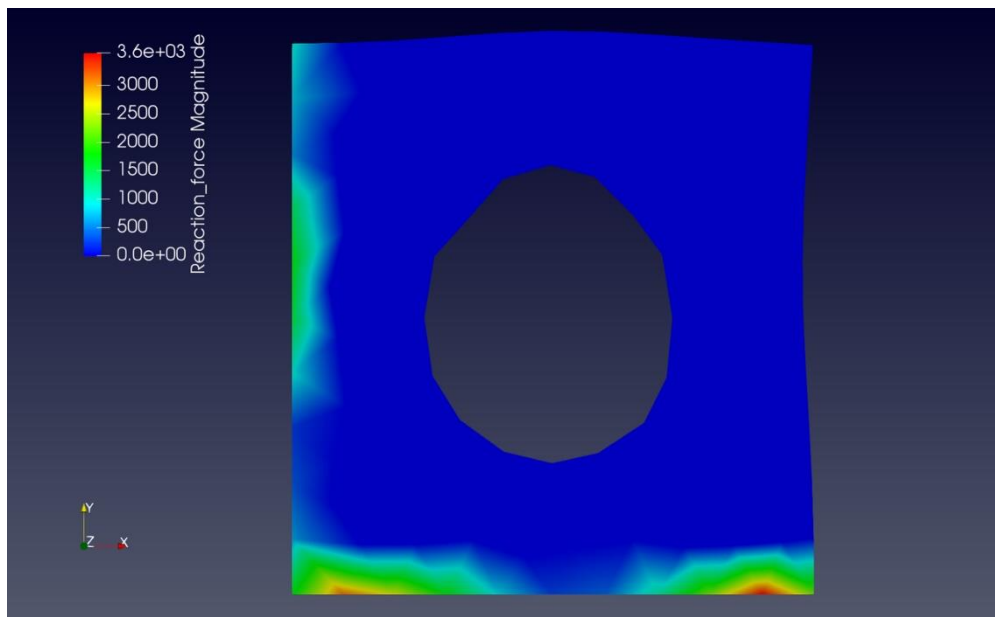
Raw data from. odb

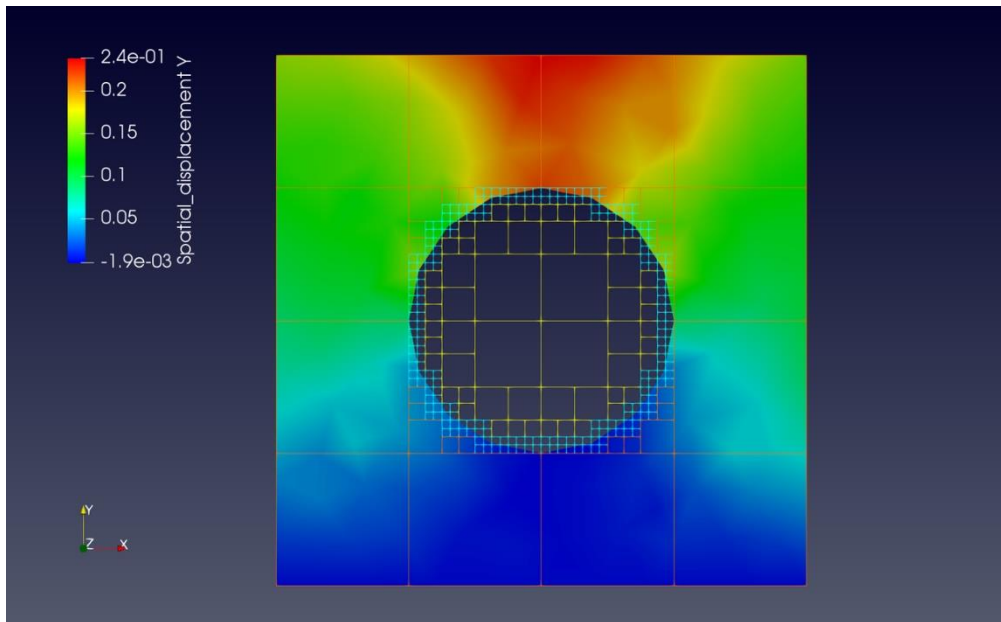




After interpolation, we can visualize different kinds of results on physical domain.







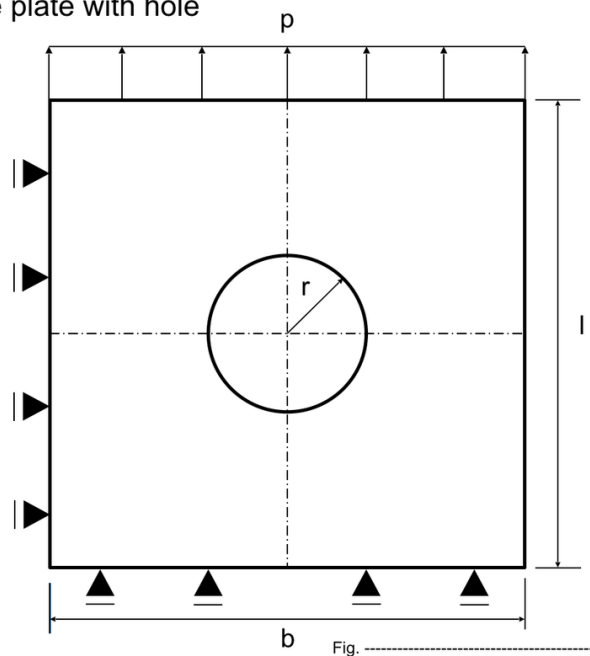
Comparison with FEM

Benchmark test

We performed a simple benchmark test in order to check that our method is on the right way.

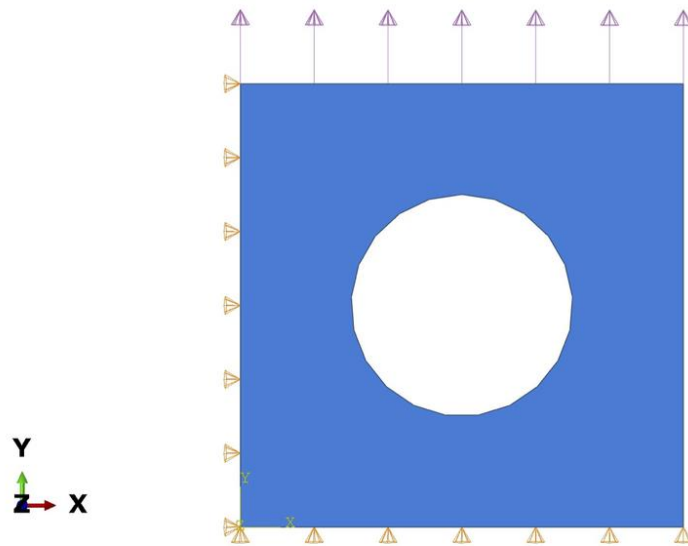
Benchmark test

Square plate with hole

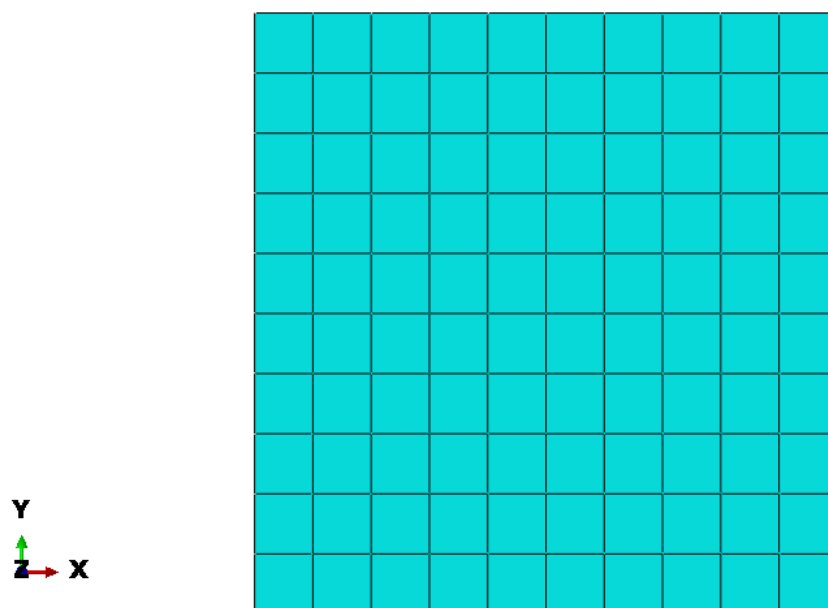
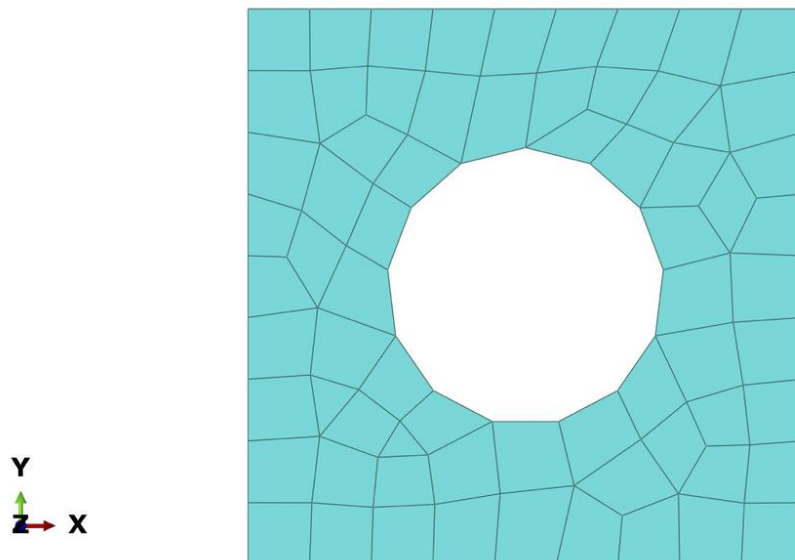


Parameters

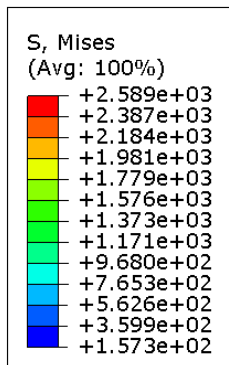
- $l = 30 \text{ mm}$
- $b = 30 \text{ mm}$
- $r = 7.5 \text{ mm}$
- $p = 100 \text{ N/mm}$
- $E = 210000 \text{ MPa}$
- $\nu = 0.3$



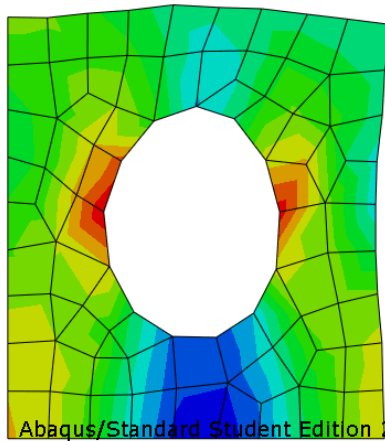
Mesh vs Embedded domain



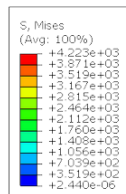
von Mises stresses



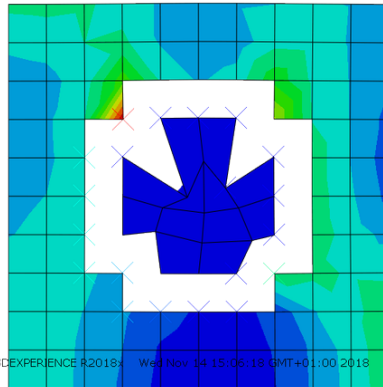
ODB: Job-1.odb
Step: static_load
Increment 1: Step Time = 2.2200E-16
Primary Var: S, Mises



Abaqus/Standard Student Edition 2018 Wed Nov 14 14:36:55 W

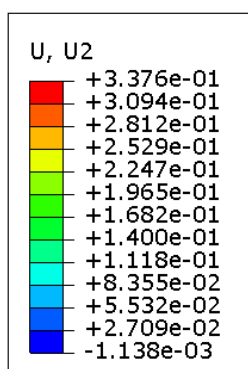


ODB: Test_run.odb
Step: Step-2
Increment 1: Step Time = 2.2200E-16
Primary Var: S, Mises
Deformed Var: U Deformation Scale Factor: +1.143e+00

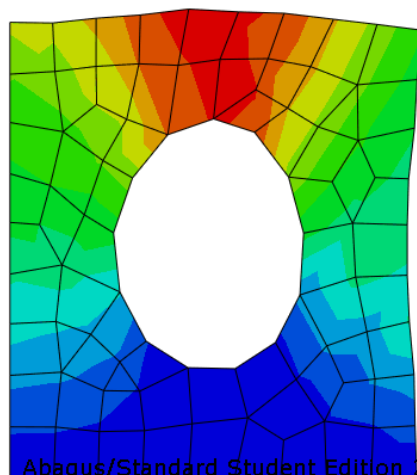


Abaqus/Standard 3DEXPERIENCE R2018x Wed Nov 14 15:06:18 GMT+01:00 2018

Y - displacement

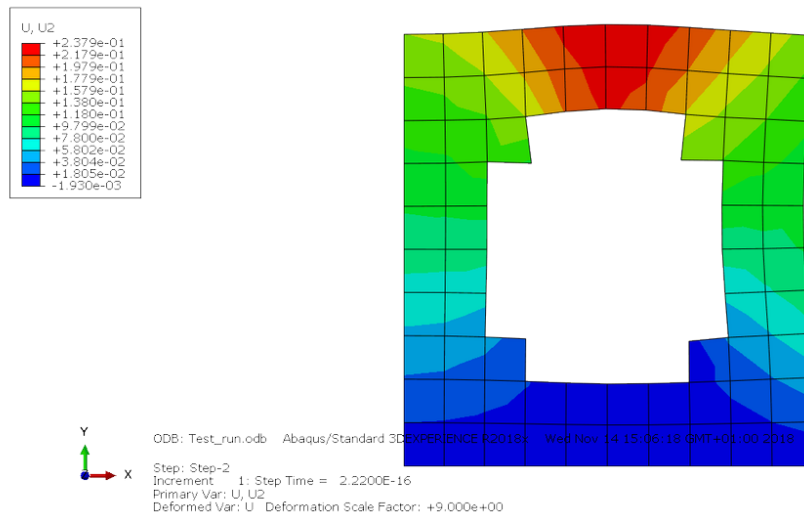


ODB: Job-1.odb
Step: static_load
Increment 1: Step Time = 2.2200E-16
Primary Var: U, U2



Abaqus/Standard Student Edition 2018 Wed Nov 14 14:36:55 W

Here we remove user defined elements and outside-domain elements, since its displacement is too large due to the fact that the stiffness of outside-domain is technically zero.



7 Conclusion and Outlook

The method developed in this project to implement FCM in ABAQUS is simple and yet it shows reasonable resemblance to the results from traditional FEM method. The area of implementing user subroutines is quite vast and provides with lots of interesting features and opportunities. The UEL option can be explored more in detail and the elements can be extended to 3D while currently they are only limited to the 2D plane stress conditions.

In this work, while writing the UEL subroutine, we have restricted ourselves mostly to Python. The results from the Python script are read by the Fortran coded UEL. In future a more efficient UEL entirely written in Fortran can be implemented, eliminating the need of reading the text files which is tedious and takes longer time to complete.

Also, the boundary conditions in this implementation has been applied directly on the nodes, where as the FCM demands that the BCs must be applied in a weak form over the actual physical domain. This can be possibly implemented via the user subroutine DLOAD along with the UEL, and then simultaneously running both the subroutines in one simulation.

One of the limiting aspects of using the UEL comes in the postprocessing. The user defined elements in the output database file (.odb) are not recognized by ABAQUS visualization tool for post processing. So, the user must either use a 3rd party visualization tool for the post-processing like PARAVIEW

8.1 References

- 1] - N. Zander a, T. Bog a, M. Elhaddad a, R. Espinoza a, H. Hua, A. Joly b, C. Wua, P. Zerbe a, A. Düster c, S. Kollmannsberger a, J. Parvizian d, M. Ruess e, D. Schillinger f, E. Rank a, FCMLab: A finite cell research toolbox for MATLAB, Advances in Engineering Software, 49-63, 2014
- [2] - Jamshid Parvizian, Alexander Düster, Ernst Rank, Finite Cell Method h- and p-extension for embedded domain problems in Solid Mechanics, Computational Mechanics manuscript
- [3] - A. Düster, E. Rank, B. Szabó, The p-Version of the Finite Element and Finite Cell Methods, Encyclopedia of Computational Mechanics Second Edition, September 2017
- [4] - A. Düster, J. Parvizian, Z. Yang and E. Rank, The finite cell method for three-dimensional problems of solid mechanics, Computer Methods in Applied Mechanics and Engineering, pp. 45-48, 15 August 2008.

8.2 Important Links

- [1] – ABAQUS user subroutine reference guide
<http://dsk.ippt.pan.pl/docs/abaqus/v6.13/books/sub/default.htm>
- [2] – ABAQUS scripting user guide
<http://dsk.ippt.pan.pl/docs/abaqus/v6.13/books/cmd/default.htm>
- [3] - ABAQUS scripting reference guide
<http://dsk.ippt.pan.pl/docs/abaqus/v6.13/books/ker/default.htm>
- [4] – Link to the video showing the steps involved in the current implementation of FCM in ABAQUS
<https://www.youtube.com/watch?v=iMdhxQwCvxw&feature=youtu.be>
