NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY NTNU
FACULTY OF ENGINEERING
DEPARTMENT OF STRUCTURAL ENGINEERING

TKT4142 Finite Element Methods in Structural Engineering

CASE STUDY 4

SOLUTION PROPOSAL

Task 1

a) The cantilever beam should be modeled using 8-node brick elements with reduced integration (C3D8R in Abaqus) and 50 mm characteristic size. Report your model in Abaqus by generating a figure of the model.

Solution:

Figure 1 shows a figure of the beam model with 50.0 mm 8-node brick elements (C3D8R).

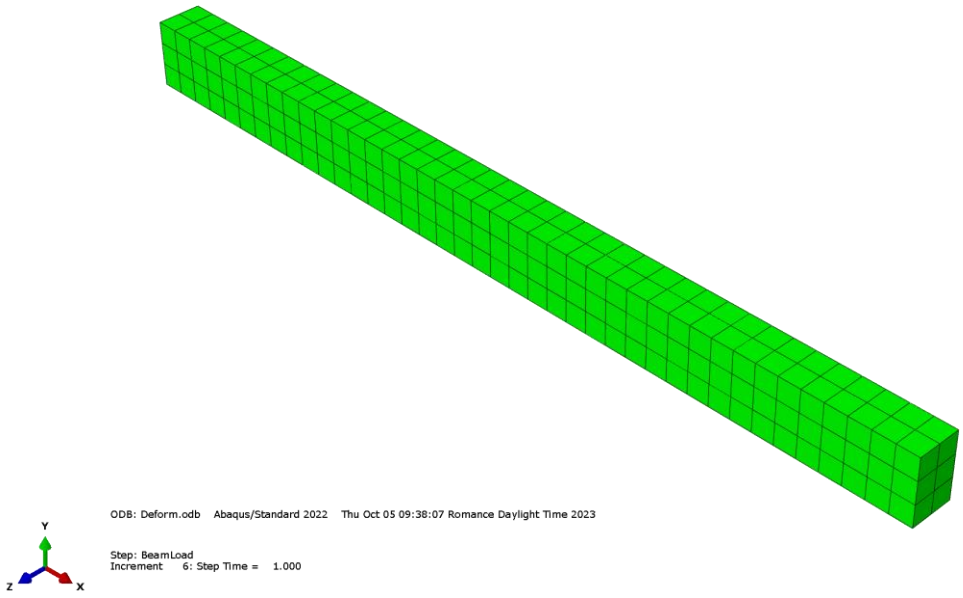

Figure 1.

b) Run a simulation in Abaqus using the file established in a). View the analysis results in the visualization module. From the simulation, take out and report the following information:

1) The deformed shape of the cantilever beam on top of the undeformed shape.

2) Contours plot of the vertical displacement on the deformed shape.

3) Contours plot of the von Mises stress on the deformed shape.

4) Default visualization in Abaqus of contours plots uses averaging of the field output between elements. Repeat the contours plot of the von Mises stress on the deformed shape but without averaging between elements, i.e., evaluate results on an element-by-element basis. Discuss the results.

5) The von Mises stress in the two most critical elements and the vertical displacement in two of the top and bottom nodes at the free end of the beam.

Solution:

1) Figure 1 shows the deformed shape on top of the undeformed geometry.

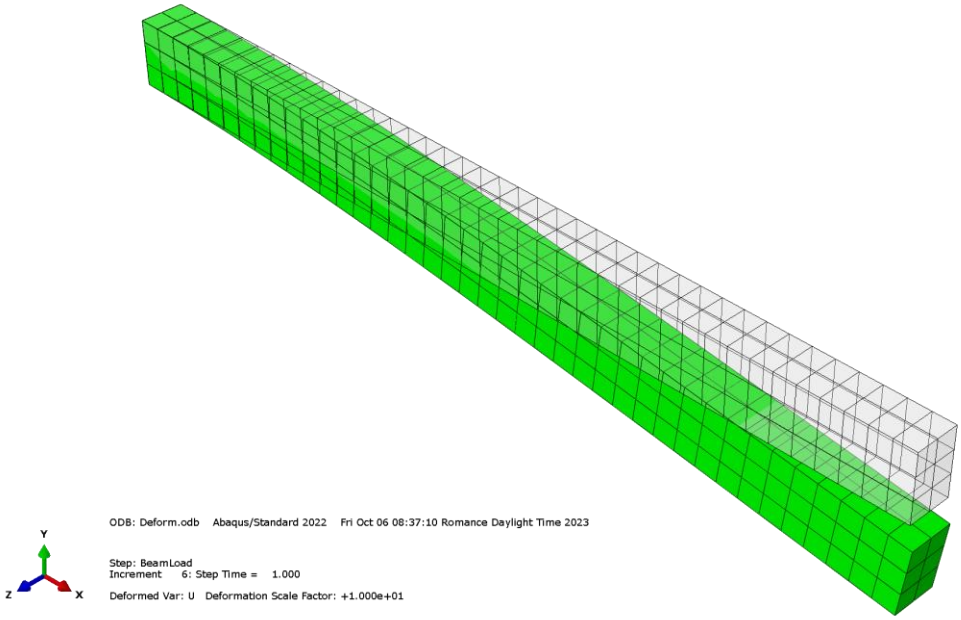

Figure 2. Note that the displacement scale factor is set to 10.

2) Figure 3 shows a contour plot of the vertical deformation U2 on the deformed shape.

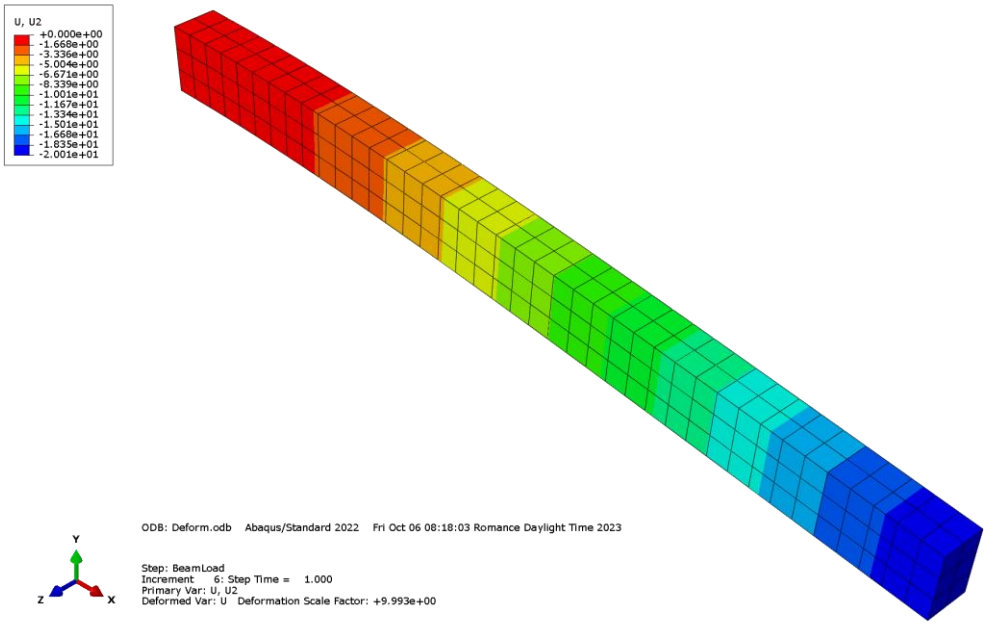

Figure 3: Displacement scale factor: 10.

3) Figure 4 shows a contour plot of the averaged (75%) von Mises stress on the deformed shape.

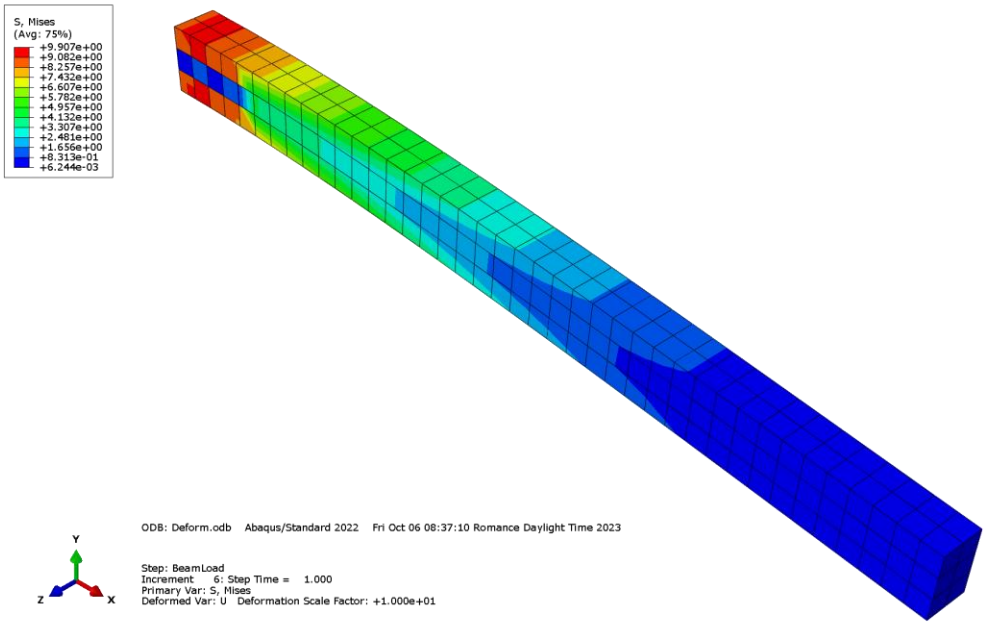

Figure 4: Displacement scale factor: 10.

4) Figure 5 shows a contour plot of the unaveraged von Mises stress on the deformed shape.

We observe very similar magnitudes of stress in Figs. 4 and 5, however, the visualization is interpolated in Fig. 4 while the stresses are shown element wise in Fig. 5.

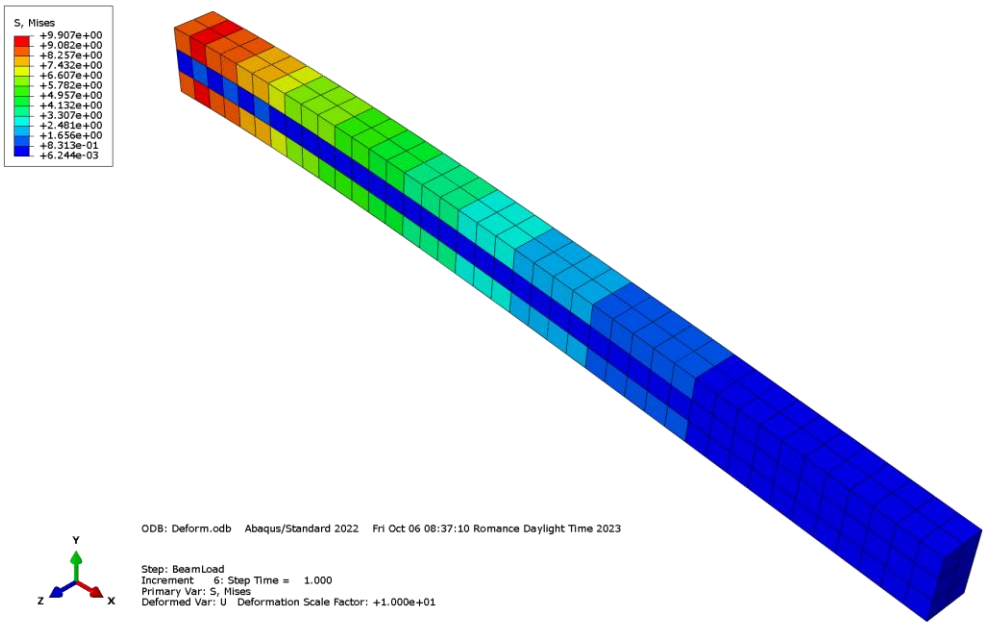

Figure 5.

	Value [MPa]	Occurs in element number
Critical von Mises stress	9.91 (Top)	9
	9.91 (Top)	12
	9.90 (Bottom)	7
	9.90 (Bottom)	10
	Value [mm]	Occurs in node number
Vertical deformation	-20.01	484 (Top)
	-20.01	488 (Top)
	-20.01	492 (Top)
	-20.01	481 (Bottom)
	-20.01	485 (Bottom)
	-20.01	489 (Bottom)

Table 1.

c) Evaluate the maximum bending stress and the maximum displacement of the cantilever beam. You are given the following solution based on elementary (Euler-Bernoulli) beam theory:

$$\sigma_{x,\max} = \frac{M}{I}\frac{H}{2} = 13.33 \text{ MPa}, v_{\max} = \frac{qL^4}{8EI} = -17.78 \text{ mm}$$

Solution:

We see that elementary beam theory gives $\sigma_{x,\max} = 13.33$ MPa for the maximum axial stress in the beam. Our model predicts a max value for the von Mises stress of 9.91 MPa, which is about 25.7% lower. The deflection from elementary beam theory is -17.78 mm, which means that our model overpredicts the deflection with about 12.5 %.
To increase the accuracy of our model, we should use a finer mesh in the beam or change the element type to an element using selective reduced integration (e.g., C3D8 elements in Abaqus).

d) Re-run the model using a characteristic element size of 25.0 mm and 12.5 mm. Compare and discuss the results against those obtained in b). What happens with the computational time and the memory requirements?

Solution:

We re-run the model with new meshes. The results are reported in Table 2.

Element size [mm]	Max. vM stress [MPa]	Compared to beam theory [%]	Compared to previous mesh	Max. deflection [mm]	Compared to beam theory [%]	Compared to previous mesh	CPU time [s]	Memory used [MB]
50	9.91	-25.7	-	-20.01	12.5	-	0.2	17
25	11.05	-17.1	11.5	-18.30	2.9	-8.5	0.9	19
12.5	12.62	-5.3	14.2	-17.92	0.8	-2.1	6	48
Element size [mm]	Max. S11 [MPa]	Compared to beam theory [%]	Compared to previous mesh	Max. deflection [mm]	Compared to beam theory [%]	Compared to previous mesh	CPU time [s]	Memory used [MB]
50	9.74	-26.9	-	-20.01	12.5	-	0.2	17
25	11.80	-11.5	21.1	-18.30	2.9	-8.5	0.9	19
12.5	14.12	5.9	19.6	-17.92	0.8	-2.1	6	48

Table 2.

Table 2 shows the maximum von Mises stress and maximum deflection for the beam, as well as the CPU time and the minimum memory required to run each model. We note that the results are similar to what we expect from our model; a finer mesh will provide more accurate results. The drawback of increasing the mesh density is also seen from the CPU time and memory used. Still, the CPU times experienced in these simulations are for all practical purposes very fast and we can run models with 3D solid elements without being concerned about the CPU cost. CPU time and memory usage are more important for larger models.

Note that the values for CPU time and memory are not absolute; they might vary from computer to computer. However, they do highlight the differences between our meshes.

e) Evaluate the internal strain energy (ALLIE in the OBD history output) and the artificial strain energy (ALLAE in the ODB history output). How can we use these energies to evaluate the performance of the FEA?

Solution:

Element size [mm]	ALLAE	ALLIE	ALLAE/ALLIE
50	45	20070	0.22 %
25	8	18357	0.04 %
12.5	2	17978	0.01 %

Table 3.

Large values of artificial strain energy (ALLAE) indicate that mesh refinement or other changes to the mesh are necessary.

One way to evaluate the importance of artificial strain energy is by comparison to the internal energy (ALLIE) component. The ratio ALLAE/ALLIE can be used to help evaluate whether an analysis is yielding an appropriate response. The internal energy is the sum of the recoverable elastic strain energy and the artificial strain energy.

The ratio ALLAE/ALLIE should not exceed 2-3% in a simple linear analysis. In this simulation, we see that all ALLAE values are below 2-3% of the ALLIE. We can therefore use reduced elements because we are within the recommended threshold in the Abaqus manual. See Workshop4.pdf for more information.

f) We will now change the element type to a C3D8 element. This is an element with selective reduced integration and should provide more accurate results for bending-dominated problems. Model the beam using element sizes of 50.0 mm and 25.0 mm. How do the FEA predictions compare to the predictions using 2D plane stress elements (CPS8 in Abaqus) in Case Study 1?

Element size [mm]	Element type	Max von Mises stress [MPa]	Max S11 [MPa]	Max deflection [mm]
50	C3D8	12.37	11.40	-17.61
	CPS8	13.34	14.46	-17.85
25	C3D8	13.03	12.80	-17.73
	CPS8	14.98	15.81	-17.86

Table 4.

Table 4 shows a comparison for two elements with two element types.

For all practical purposes, these results are more or less the same. For this particular problem, we do not increase the CPU time significantly by using solid elements for the course meshes (see task 1d). We can therefore use 3D solid elements without any concerns regarding CPU efficiency for this simple problem.

Task 2

We will now increase the complexity of the model and introduce the two holes in the beam. The dimensions and position of the holes are given in Figure 4. The beam is still embedded in the concrete wall at one end and supported by the steel rod at the hole at the opposite end of the beam.

a) Include the holes in the geometry of the part and apply appropriate boundary conditions. The beam should be modeled with a global element size of 12.5 mm. Use element type C3D8R. Report your model in Abaqus by generating a figure of the model.

Solution:

Figure 6 shows a figure of the beam model with 12.5 mm 8-node brick elements (C3D8R).

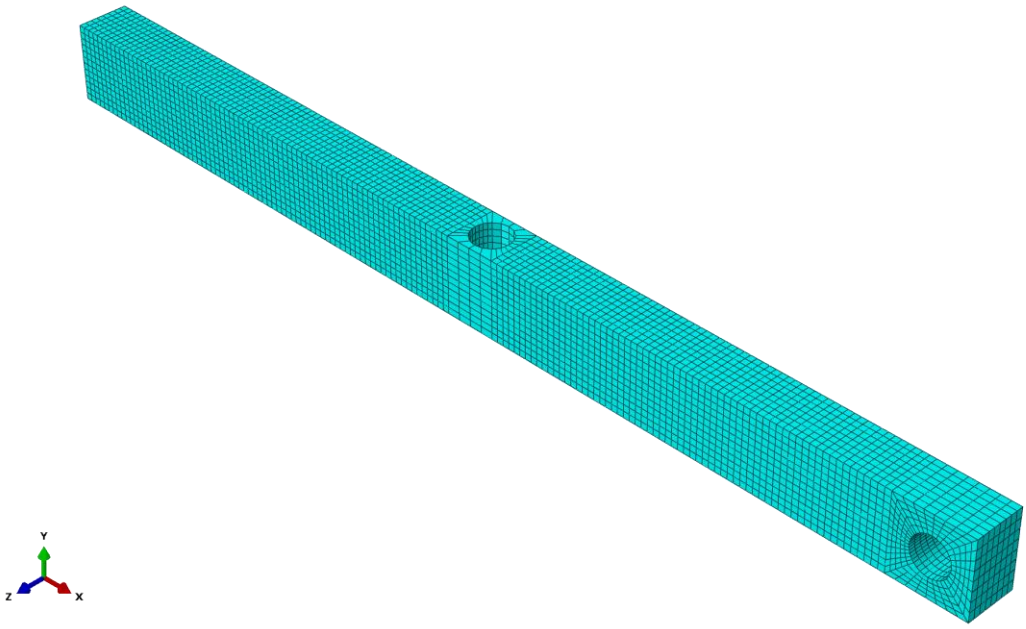

Figure 6.

b) Run a simulation in Abaqus using the file established in a). View the analysis results in the visualization module. From the simulation, take out and report the following information:

1) The deformed shape of the beam on top of the undeformed shape.

2) Contours plot of the vertical displacement on the deformed shape.

3) Contours plot of the von Mises stress and the maximum principal stress (Max. Principal in Abaqus) on the deformed shape.

Solution:

1) Figure 7 shows the deformed shape on top of the undeformed shape.

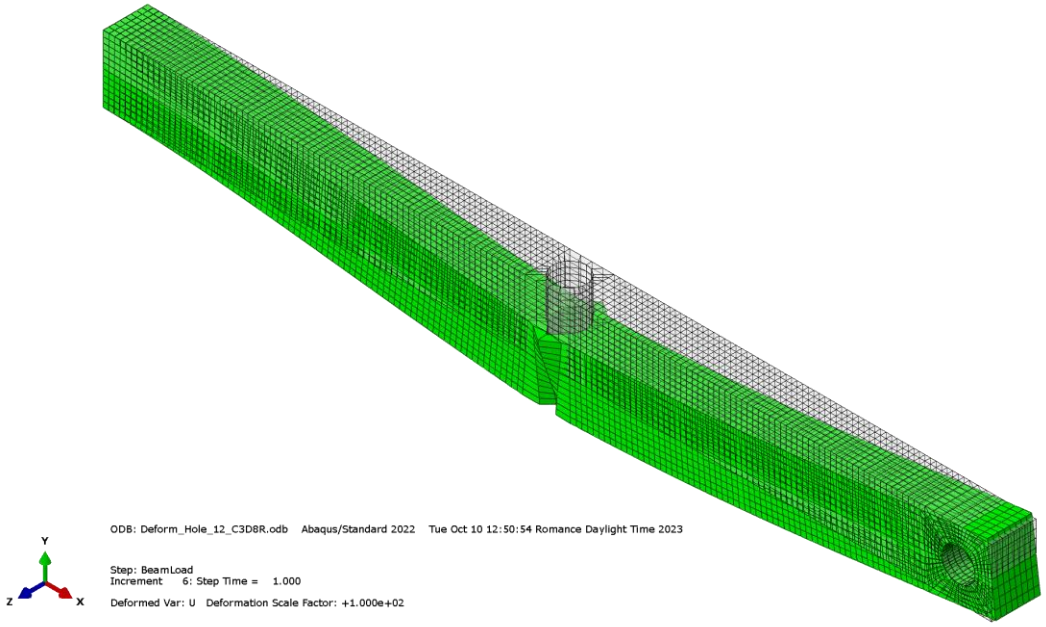

Figure 7: Displacement scale factor: 100.

2) Figure 8 shows a contour plot of the vertical displacement U2 on the deformed shape. Figure 9 shows the vertical displacement with a scale factor of 1.

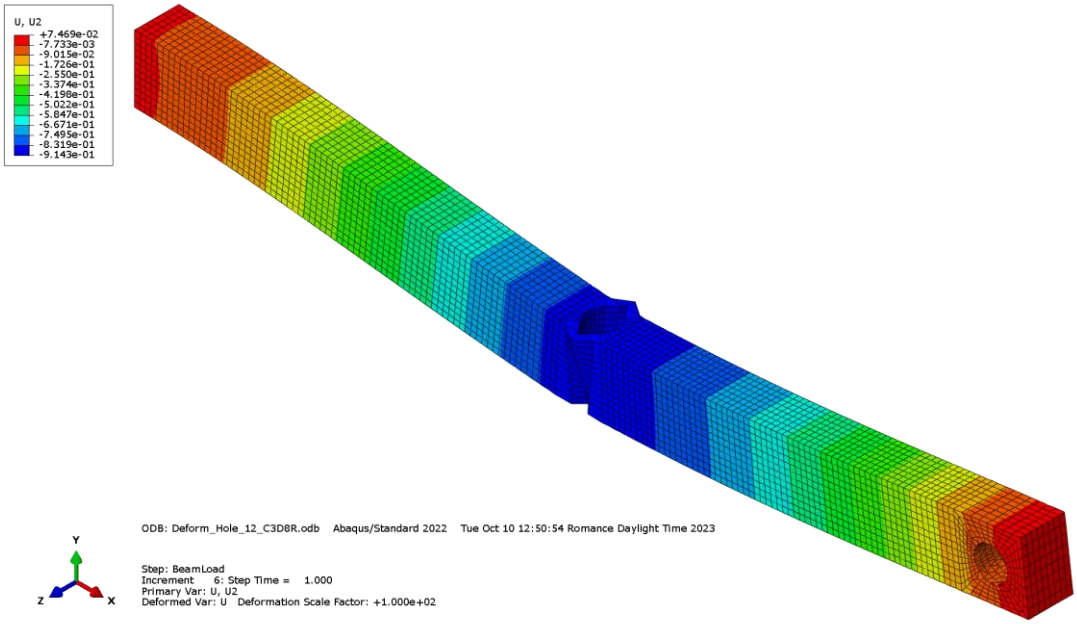

Figure 8: Displacement scale factor: 100.

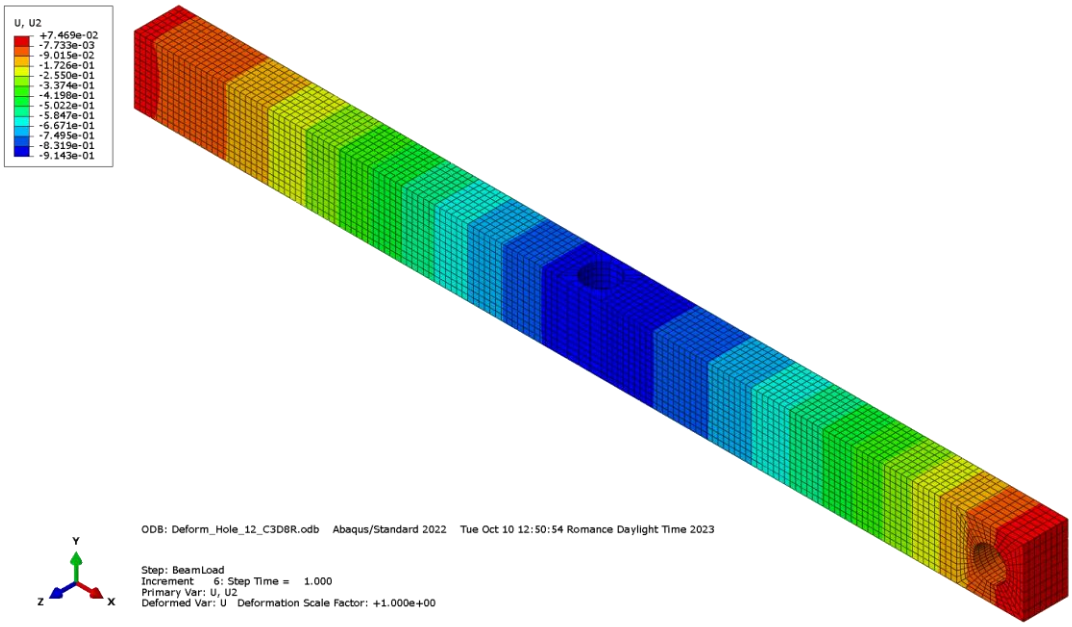

Figure 9: Displacement scale factor: 1.

3)

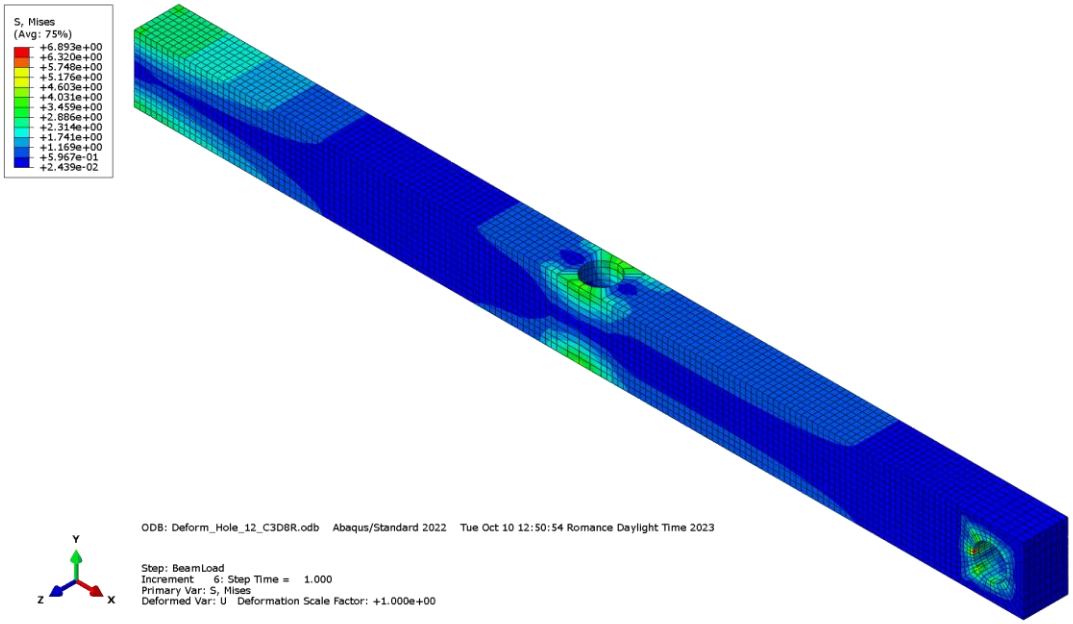

Figure 10: von Mises stress averaged (75%). Displacement scale factor: 1.

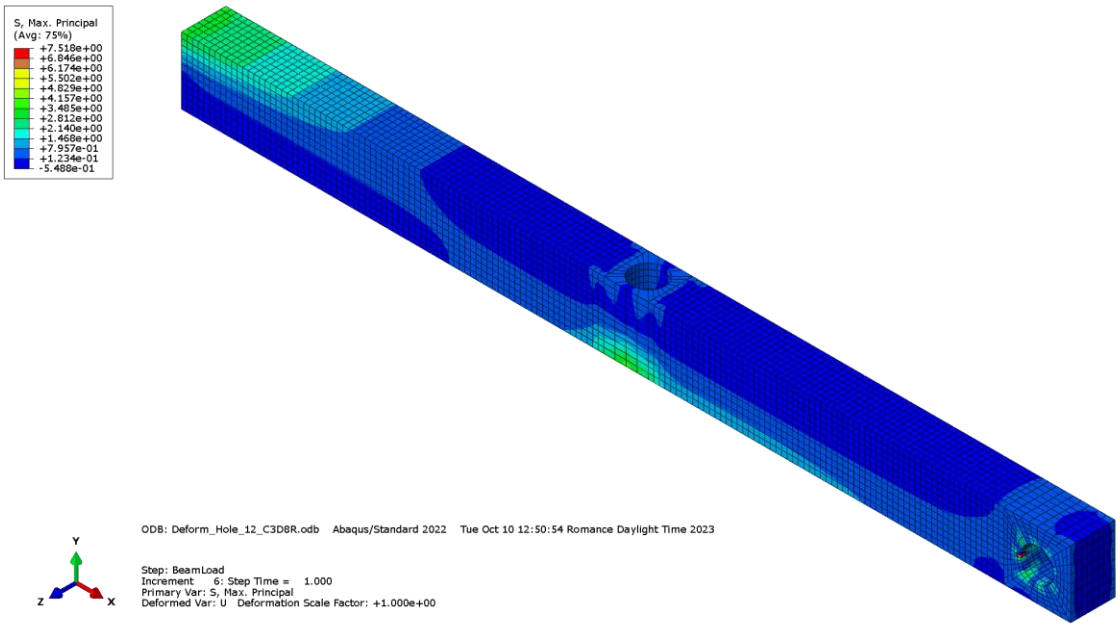

Figure 11: Maximum principal stress. Displacement scale factor: 1.

c) Evaluate the internal strain energy (ALLIE in the OBD history output) and the artificial strain energy (ALLAE in the ODB history output). What is your evaluation of the performance of the FEA with an element size of 12.5 mm and element type C3D8R?

Solution:

We evaluate the results for an element size of 12.5 mm and element type C3D8R.

Element size [mm]	ALLAE	ALLIE	ALLAE/ALLIE
12.5	205	1087	18.9%

We see that the artificial energy is almost 20% of the internal energy. This is above the recommended threshold from Abaqus (see Task 1e), i.e., the recommended threshold is 2-3 %).

The hourglass modes are also clearly visible in Figs. 7 and 8. These modes are observed as element distortions in the vicinity of the centre hole, i.e., modes leading to zero strain energy in linear elements. Thus, no stresses are created within the element. There are 12 hourglass modes for a brick element, i.e., 4 modes for each of the 3 coordinate directions. They produce linear strain modes, which cannot be accounted for using a standard one-point integration scheme. To correct this phenomenon, it is necessary to introduce anti-hourglass forces and moments or use selectively reduced integration (element type C3D8 in Abaqus).

d) We will now change the element type to a C3D8 element. Re-run the model with the new element type. Evaluate the maximum von Mises stress, the maximum

vertical displacement, and the ratio ALLAE/ALLIE. Compare the results for the two element types.

Solution:

Element size: 12.5 mm

Element type	Max. von Mises stress [MPa]	Compared to previous mesh	Max. vertical displacement [mm]	Compared to previous mesh	CPU time [s]	Memory used [MB]
C3D8R	6.9 (Unaveraged)	-	-9.1	-	6.3	48
C3D8	15.4 (Unaveraged)	223%	-5.5	-40%	9.7	48

Element type	ALLAE	ALLIE	ALLAE/ALLIE
C3D8	0	689	0

As expected, when we use fully integrated elements, the artificial energy ALLAE is equal to zero.

e) Refine your mesh by reducing the global element size by a factor of 2 and re-run the analysis. Comment on your observations. Also, check the other principal stress components. Why should we check both the principal stress components and the von Mises stress?

Solution:

To refine the mesh, we set a global seed of 6.25 to reduce the element side length by a factor 2. The mesh now looks something like this.

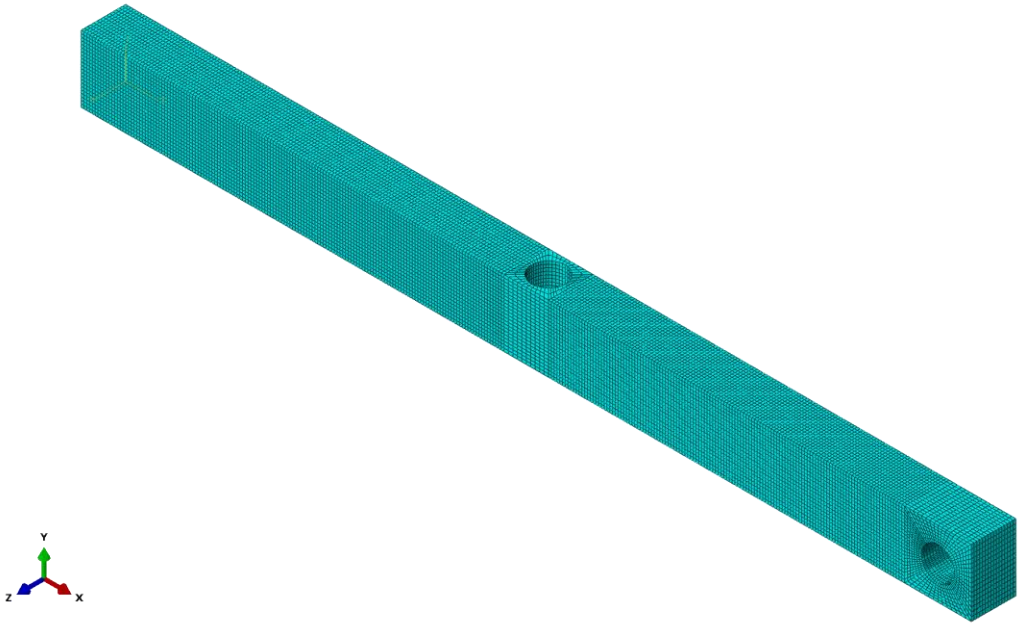

Figure 12. Refined mesh.

We can plot the vertical deformation (U2) (Figure 13) and the averaged (75%) von Mises stress (Figure 14) of the beam.

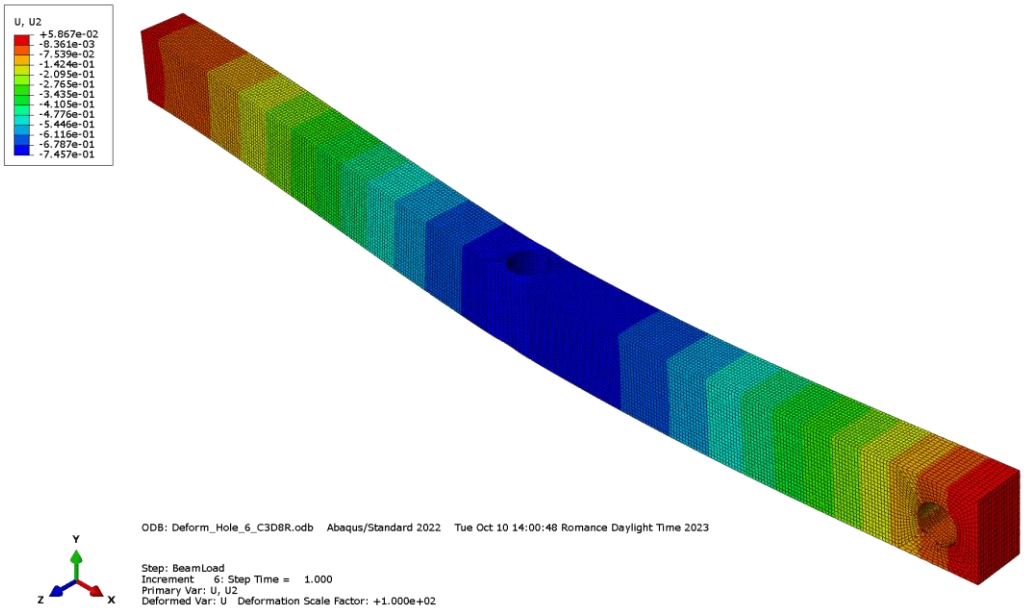

Figure 13: Vertical deformation (U2) of the beam. Displacement scale factor: 100

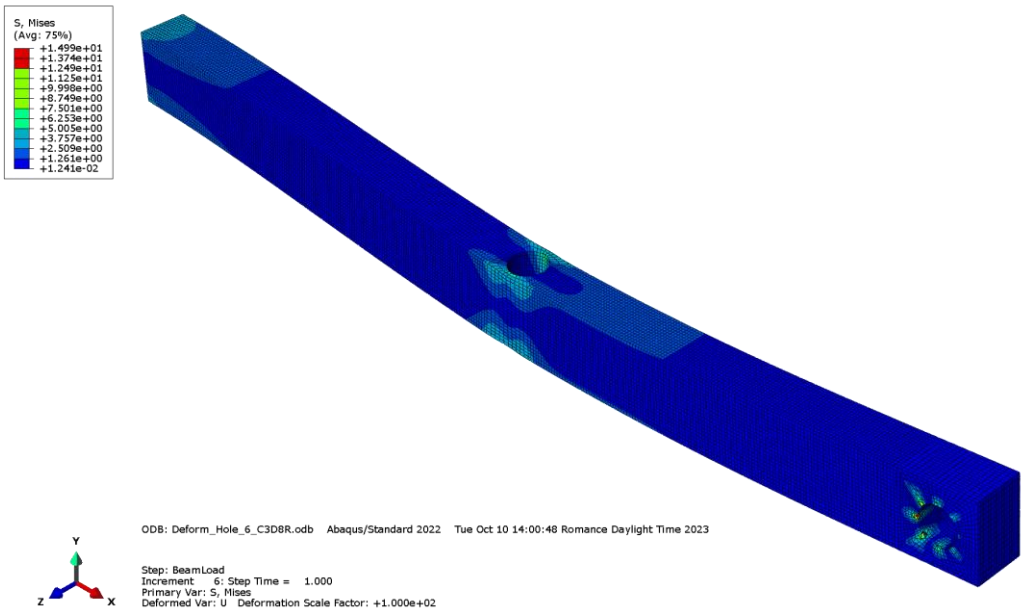

Figure 14: Averaged von Mises stress of the beam. Displacement scale factor: 100.

We can compare the results for the two meshes.

Element size [mm]	Max. principal [MPa]	Mid. Principal [MPa]	Min. Principal [MPa]	CPU time [s]	Memory used [MB]
12.5	7.52	1.22	-7.58	6.3	48
6.25	16.86	3.35	-16.93	60	303

Why should we check both the principal stress components and the von Mises stress?

The comparison between von Mises stress and principal stress is an important aspect of understanding the behaviour of materials under different loading conditions. Von Mises stress and principal stress are two different methods used to analyse and interpret the FEA results. While von Mises stress provides a measure of the equivalent stress in a material, principal stress helps identify the maximum and minimum stress values acting on a specific element in your model. Both concepts play a crucial role in designing and analyzing structures.

f) Change the element type to a C3D8 element for the refined mesh. Re-run the model with the new element type, and evaluate the maximum von Mises stress, the maximum vertical displacement, and the ratio ALLAE/ALLIE. Compare the results for the two element types.

Solution:

Element size: 6.25 mm

Element type	Max. von Mises stress [MPa]	Compared to previous mesh	Max. vertical displacement [mm]	Compared to previous mesh	CPU time [s]	Memory used [MB]
C3D8R	15.0	-	-7.5	-	60	303
C3D8	36.1	240%	-6.2	-17.3	83	303

Element type	ALLAE	ALLIE	ALLAE/ALLIE
C3D8R	37	962	3.8%
C3D8	0	782	0

We observe that when refining the mesh, the problem with hourglass modes is significantly reduced. However, the ratio ALLAE/ALLIE is still slightly above the recommended threshold by Abaqus and the simulation results should be used with caution.

g) Keep the size of the global elements as in c) and increase the pressure load by a factor of 4. Comment on your observations. How does the von Mises stress compare to the elastic limit of the material ($\sigma_y = 20\ \mathrm{N/mm^2}$)? Do you trust the structural integrity of the model for the new (increased) pressure loading?

Solution:

Now, we have increased the load by a factor 4 to **0.1** MPa. The maximum von Mises stress in the simulation is now 60.0 MPa. This mean that we are beyond the elastic capacity of our material which has a yield stress of 20 MPa. We should be cautious when we extract any information from the model. We plot the von Mises stress in Figure 15.

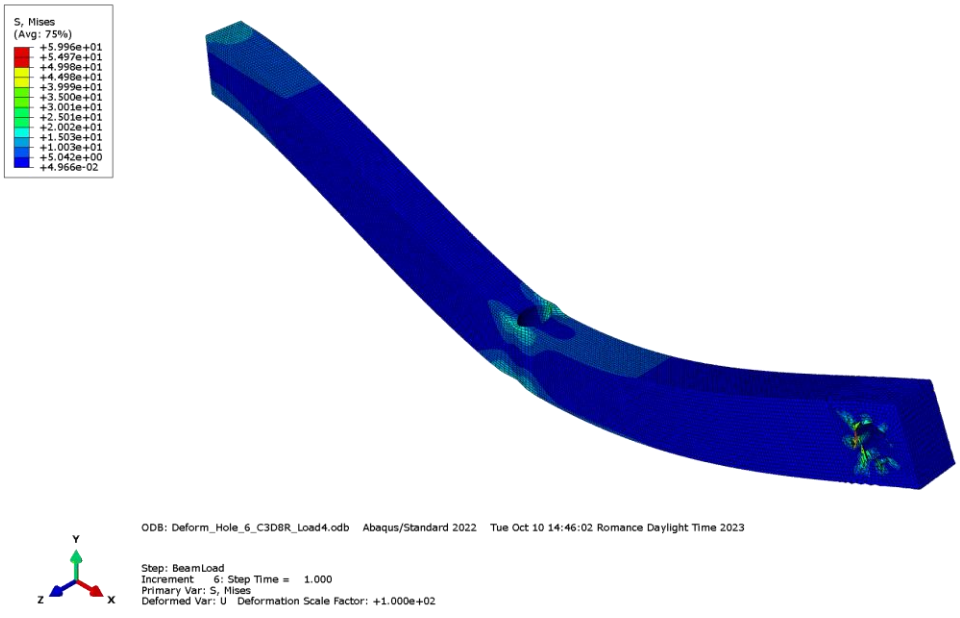

Figure 15: von Mises stress on the beam. Displacement scale factor: 100.

We can adjust the limits in our contour plot (**Options**, **Contour**, **Limits**) to identify the critical parts of the beam. In Figure 16 we have set the maximum value to **20** MPa. We see from the figure that the elements around both holes are the critical parts of the beam.

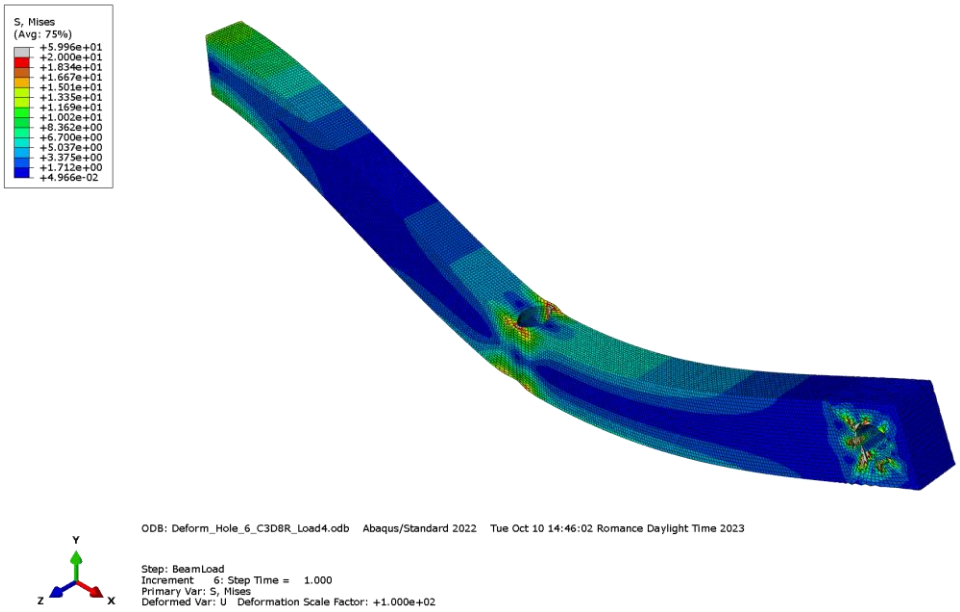

Figure 16: Mises stress on the beam with the max value set to 20 MPa. Displacement scale factor: 100.

To improve our model, we could model the material with some non-linear behaviour beyond the elastic limit. However, this is beyond the scope of this course (TKT4142).

h) In the following, we will look at some Python scripting in Abaqus.

In this example, we start with a fresh .cae file. Click **With Standard/Explicit Model** and set your **Work Directory. File, Macro Manager, Create.** We want to make a macro/Python script that creates our part. **Name: create_part** and **Directory: Work.** When we click **Continue,** Abaqus starts recording all our clicking and inputs in the cae. Click **Continue**.

Now, we do the following

1) Rename the model from **Model-1** to **Beam**.
2) Create a **3D**, **Deformable**, **Solid** part with **Extrusion**, **Approximate size 1000** and name **Beam**.
3) Create a **rectangle** in the sketch area and use the **Dimension tool** to set the length to **500** and the **height** to **100**. Set **Depth 50**. Now we see our part.
4) Click **Stop Recording** in the **Record Macro** box.

In the **Macro Manager** box, we now see our macro **create**_part.

5) Open the **work directory**, and find the file called **abaqusMacros**. Open the file in a text editor (Notepad, Sublime…) or Python. The file looks something like this.

abaqusMacros.py

```
# -*- coding: mbcs -*-
# Do not delete the following import lines
from abaqus import *
from abaqusConstants import *
import __main__

def create_part():
    import section
    import regionToolset
    import displayGroupMdbToolset as dgm
    import part
    import material
    import assembly
    import step
    import interaction
    import load
    import mesh
    import optimization
    import job
    import sketch
    import visualization
    import xyPlot
    import displayGroupOdbToolset as dgo
    import connectorBehavior
    mdb.models.changeKey(fromName='Model-1', toName='Beam')
    session.viewports['Viewport: 1'].setValues(displayedObject=None)
    s = mdb.models['Beam'].ConstrainedSketch(name='__profile__', sheetSize=1000.0)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=STANDALONE)
    s.rectangle(point1=(-180.0, 80.0), point2=(80.0, 40.0))
    s.DistanceDimension(entity1=g[2], entity2=g[4], textPoint=(61.1423645019531,
        112.111694335938), value=500.0)
    s.DistanceDimension(entity1=g[3], entity2=g[5], textPoint=(-215.761871337891,
        69.2065582275391), value=100.0)
    p = mdb.models['Beam'].Part(name='BeamPart', dimensionality=THREE_D,
        type=DEFORMABLE_BODY)
    p = mdb.models['Beam'].parts['BeamPart']
    p.BaseSolidExtrude(sketch=s, depth=50.0)
    s.unsetPrimaryObject()
    p = mdb.models['Beam'].parts['BeamPart']
    session.viewports['Viewport: 1'].setValues(displayedObject=p)
    del mdb.models['Beam'].sketches['__profile__']


```

We recognize the function **create**_part. Now, we are actually ready to use our macro. We can close **CAE** and reopen it and create a new model. Change the **Work Directory** to the location where we created the macro file **abaqusMacros.py**. Go to **File**, **Macro Manager**. We see that our macro function **create_part** appears. We can now run this (click **Run**), and our part appears almost instantly.

These are the basics of macros. Now, we can do some cleaning of the file and try to create some variables.

6) Cleaning: We can move all the imports outside the function. This is shown in the Figure below. Usually, we can also remove anything related to **session.viewports.**

```
1  # -*- coding: mbcs -*-
2  # Do not delete the following import lines
3  from abaqus import *
4  from abaqusConstants import *
5  import __main__
6  import section
7  import regionToolset
8  import displayGroupMdbToolset as dgm
9  import part
10 import material
11 import assembly
12 import step
13 import interaction
14 import load
15 import mesh
16 import optimization
17 import job
18 import sketch
19 import visualization
20 import xyPlot
21 import displayGroupOdbToolset as dgo
22 import connectorBehavior
23
24
25 def create_part():
26
27     mdb.models.changeKey(fromName='Model-1', toName='Beam')
28     s = mdb.models['Beam'].ConstrainedSketch(name='__profile__', sheetSize=1000.0)
29     g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
30     s.setPrimaryObject(option=STANDALONE)
31     s.rectangle(point1=(-180.0, 80.0), point2=(80.0, 40.0))
32     s.DistanceDimension(entity1=g[2], entity2=g[4], textPoint=(61.1423645019531,
33         112.111694335938), value=500.0)
34     s.DistanceDimension(entity1=g[3], entity2=g[5], textPoint=(-215.761871337891,
35         69.2065582275391), value=100.0)
36     p = mdb.models['Beam'].Part(name='BeamPart', dimensionality=THREE_D,
37         type=DEFORMABLE_BODY)
38     p = mdb.models['Beam'].parts['BeamPart']
39     p.BaseSolidExtrude(sketch=s, depth=50.0)
40     s.unsetPrimaryObject()
41     p = mdb.models['Beam'].parts['BeamPart']
42     del mdb.models['Beam'].sketches['__profile__']
43
44
```

7) We recognize the **length** of the rectangle (on line 32 and 33 we see **value=500.0**). Cut out **500.0** and replace it with **parameter_length**. We see the **height** on line 35. Cut out **100.0** and replace it with **parameter_height**. We see the **depth** on line 39. Cut out **100.0** and replace it with **parameter_ depth**.
8) Create the three parameters **parameter_length=2000.0**, **parameter_height=150.0** and **parameter_depth=100.0** earlier in the function **create_part**. This is shown in the below figure.

```
25  def create_part():
26      parameter_length = 2000.
27      parameter_height = 150.
28      parameter_length = 100.
29
30      mdb.models.changeKey(fromName='Model-1', toName='Beam')
31      s = mdb.models['Beam'].ConstrainedSketch(name='__profile__', sheetSize=1000.0)
32      g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
33      s.setPrimaryObject(option=STANDALONE)
34      s.rectangle(point1=(-180.0, 80.0), point2=(80.0, 40.0))
35      s.DistanceDimension(entity1=g[2], entity2=g[4], textPoint=(61.1423645019531,
36          112.111694335938), value=parameter_length)
37      s.DistanceDimension(entity1=g[3], entity2=g[5], textPoint=(-215.761871337891,
38          69.2065582275391), value=parameter_height)
39      p = mdb.models['Beam'].Part(name='BeamPart', dimensionality=THREE_D,
40          type=DEFORMABLE_BODY)
41      p = mdb.models['Beam'].parts['BeamPart']
42      p.BaseSolidExtrude(sketch=s, depth=parameter_depth)
43      s.unsetPrimaryObject()
44      p = mdb.models['Beam'].parts['BeamPart']
45      del mdb.models['Beam'].sketches['__profile__']
46
47
48
```

Now we realise that we can change these three input parameters, and we will instantly have the new geometry ready when we run the macro.

9) To illustrate further what we can do with macros, we create a new macro called **create_material.**
10) When **Abaqus** starts recording, we create the **macro** and **double-click Materials**. Create a material with **name Wood**, Density **1.5E-9**, **Young's modulus 30000** and **Poisson's ratio 0.3**. Stop recording.
11) Open the abaqusMacros.py file. Now, we see that our new macro**,** **create_material**, has been appended to the file as a function as in the below figure.

```
48  def create_material():
49      import section
50      import regionToolset
51      import displayGroupMdbToolset as dgm
52      import part
53      import material
54      import assembly
55      import step
56      import interaction
57      import load
58      import mesh
59      import optimization
60      import job
61      import sketch
62      import visualization
63      import xyPlot
64      import displayGroupOdbToolset as dgo
65      import connectorBehavior
66      mdb.models['Beam'].Material(name='Wood')
67      mdb.models['Beam'].materials['Wood'].Density(table=((1.5e-09, ), ))
68      mdb.models['Beam'].materials['Wood'].Elastic(table=((30000.0, 0.3), ))
69
70
71
```

We see that Abaqus has now imported all libraries again. We can remove these from the function since they are defined globally. We can create some variables that are our material input. We can modify the function to look like this:

```
48  def create_material():
49      parameter_density = 1.5E-09
50      parameter_poisson = 30000.
51      parameter_youngs  = 0.3
52      mdb.models['Beam'].Material(name='Wood')
53      mdb.models['Beam'].materials['Wood'].Density(table=((parameter_density, ), ))
54      mdb.models['Beam'].materials['Wood'].Elastic(table=((parameter_youngs, parameter_poisson), ))
55
56
57
```

We can create macros for a lot of our modelling. We can create **Sections, Steps, Loads, BCs,** define **Outputs, Create** and **Submit Jobs**, etc.

Running the macro without **Macro Manager**.

All our macros will be available through the macro manager, but we can also run them using **File, Run Script**. One thing we must be careful about is that we have to call the function in the Python file. It can also be a good idea to create the macros in the abaqusMacros.py file and copy them to another file (e.g., MyScript.py (or any other name)). We can run the new file instead. How to call the functions is shown below (line 58 and 60).

myScript.py

```python
# -*- coding: mbcs -*-
# Do not delete the following import lines
from abaqus import *
from abaqusConstants import *
import __main__
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior


def create_part():
    parameter_length = 2000.
    parameter_height = 150.
    parameter_length = 100.

    mdb.models.changeKey(fromName='Model-1', toName='Beam')
    s = mdb.models['Beam'].ConstrainedSketch(name='__profile__', sheetSize=1000.0)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=STANDALONE)
    s.rectangle(point1=(-180.0, 80.0), point2=(80.0, 40.0))
    s.DistanceDimension(entity1=g[2], entity2=g[4], textPoint=(61.1423645019531,
        112.111694335938), value=parameter_length)
    s.DistanceDimension(entity1=g[3], entity2=g[5], textPoint=(-215.761871337891,
        69.2065582275391), value=parameter_height)
    p = mdb.models['Beam'].Part(name='BeamPart', dimensionality=THREE_D,
        type=DEFORMABLE_BODY)
    p = mdb.models['Beam'].parts['BeamPart']
    p.BaseSolidExtrude(sketch=s, depth=parameter_depth)
    s.unsetPrimaryObject()
    p = mdb.models['Beam'].parts['BeamPart']
    del mdb.models['Beam'].sketches['__profile__']


def create_material():
    parameter_density = 1.5E-09
    parameter_poisson = 30000.
    parameter_youngs  = 0.3
    mdb.models['Beam'].Material(name='Wood')
    mdb.models['Beam'].materials['Wood'].Density(table=((parameter_density, ), ))
    mdb.models['Beam'].materials['Wood'].Elastic(table=((parameter_youngs, parameter_poisson), ))


# Call the functions.
create_part()

create_material()
```

We can edit the script further by having the functions take our parameters as input. We define the parameters globally and call the functions.

MyScript.py can look something like this:

myScript.py

```python
import part
import material
import assembly
import step
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior


def create_part(parameter_length, parameter_height, parameter_depth):

    mdb.models.changeKey(fromName='Model-1', toName='Beam')
    s = mdb.models['Beam'].ConstrainedSketch(name='__profile__', sheetSize=1000.0)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=STANDALONE)
    s.rectangle(point1=(-180.0, 80.0), point2=(80.0, 40.0))
    s.DistanceDimension(entity1=g[2], entity2=g[4], textPoint=(61.1423645019531,
        112.111694335938), value=parameter_length)
    s.DistanceDimension(entity1=g[3], entity2=g[5], textPoint=(-215.761871337891,
        69.2065582275391), value=parameter_height)
    p = mdb.models['Beam'].Part(name='BeamPart', dimensionality=THREE_D,
        type=DEFORMABLE_BODY)
    p = mdb.models['Beam'].parts['BeamPart']
    p.BaseSolidExtrude(sketch=s, depth=parameter_depth)
    s.unsetPrimaryObject()
    p = mdb.models['Beam'].parts['BeamPart']
    del mdb.models['Beam'].sketches['__profile__']


def create_material(parameter_density, parameter_poisson, parameter_youngs):
    parameter_density = 1.5E-09
    parameter_youngs  = 30000.
    parameter_poisson = 0.3
    mdb.models['Beam'].Material(name='Wood')
    mdb.models['Beam'].materials['Wood'].Density(table=((parameter_density, ), ))
    mdb.models['Beam'].materials['Wood'].Elastic(table=((parameter_youngs, parameter_poisson), ))

# Define input variables
# Part
parameter_length = 2000.
parameter_height = 150.
parameter_length = 100.

# Material parameters
parameter_density = 1.5E-09
parameter_youngs  = 30000.
parameter_poisson = 0.3

# Call the functions with input
create_part(parameter_length, parameter_height, parameter_depth)

create_material(parameter_density, parameter_poisson, parameter_youngs)

```