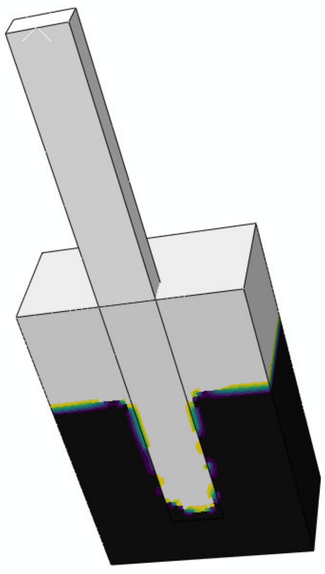
Pile jacking example in Abaqus

Getting started with CEL simulations

Dominik Zobel



version: November 16, 2020

Contents

1 Getting started with Abaqus

Getting used to work with Abaqus may seem very difficult at the beginning. The program offers an overwhelming amount of possibilities—thus the appropriate workflow and the necessary options to simulate a certain problem may not be obvious at all. These notes are meant to help getting started with simulations at the Institute of Geotechnical Engineering and therefore focus on soil mechanics and soil-structure interaction.

There are various numerical methods which are more or less effective depending on the problem to investigate. Whenever soil undergoes great deformations, the *Finite Element Method* with an *Lagrangian* analysis may yield unreliable results due to distorted mesh sections or even fail to finish the calculation.

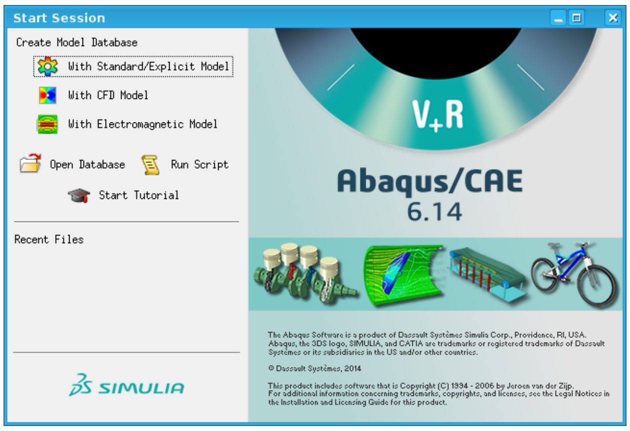
One approach, which is chosen for this tutorial, is to use the *Coupled Eulerian-Lagrangian* (CEL) method. Here, elements which are expected to flow or change their shape are modelled with Eulerian materials. The mesh of Eulerian elements will stay put and the material can flow within the element boundaries. If combined, Lagrangian and Eulerian elements are promising to model the interaction between soil and structure. In this document a simplified pile jacking CEL simulation will be conducted.

As this document was originally written in 2015, Abaqus 6.14 is used. Nevertheless, this tutorial (i. e. the workflow and all recommondations) is also valid for more recent versions like Abaqus 2020 (last tested version but even newer versions might work).

Upon starting, saving and during work Abaqus will create different files in the working directory.[1] If the model is saved, a model file `[model].cae` and a journal file `[model].jnl` will be created or updated. Journal files contain all relevant Python commands for the actions taken to create the saved model. If no issues have to be corrected manually, they can be run to automatically recreate the model from scratch.[2] After successfully creating a finished model, an input file `[model].inp` is created. Input files contain all relevant model information and are required to calculate the results in Abaqus (on a local or remote machine). The finished results will be written to the model output database `[model].odb` and can be inspected in Abaqus CAE or Viewer. Status reports and messages will be written to `[model].dat`, `[model].sta` and `[model].msg`. Those files are especially useful, if the simulation fails.

[1]The working directory can be changed by running Abaqus from the intended working folder from shell or by setting the working directory manually in the main menu `File->Set Work Directory`.

[2]The commands in the journal file can serve as a workflow example, executed in Abaqus or a compact way to store all model information. If only the finished model should be available, the model file `[model].cae` or the input files `[model].inp` do suffice.



The Abaqus Software is a product of Dassault Systèmes Simulia Corp., Providence, RI, USA. Abaqus, the 3DS logo, SIMULIA, and CATIA are trademarks or registered trademarks of Dassault Systèmes or its subsidiaries in the US and/or other countries.

© Dassault Systèmes, 2014

This product includes software that is Copyright (C) 1994 - 2006 by Jeroen van der Zijp. For additional information concerning trademarks, copyrights, and licenses, see the Legal Notices in the Installation and Licensing Guide for this product.

Figure 1.1: Start dialog of Abaqus 6.14.

Abaqus offers special solvers for certain kind of simulations. For soil-structure interaction (and many other soil mechanic investigations) the Standard/Explicit model option should be chosen in the start dialog (cf. figure 1.1). The decision between Standard and Explicit analysis will be postponed until chapter 6.

Figure 1.2 shows the interface of Abaqus CAE. The *main menu* is located at the top of the window and has two lines of additional shortcut symbols. On the left hand side is the *model tree* and its nodes. By default, the initial model will be called `Model-1` and its child elements are visible.

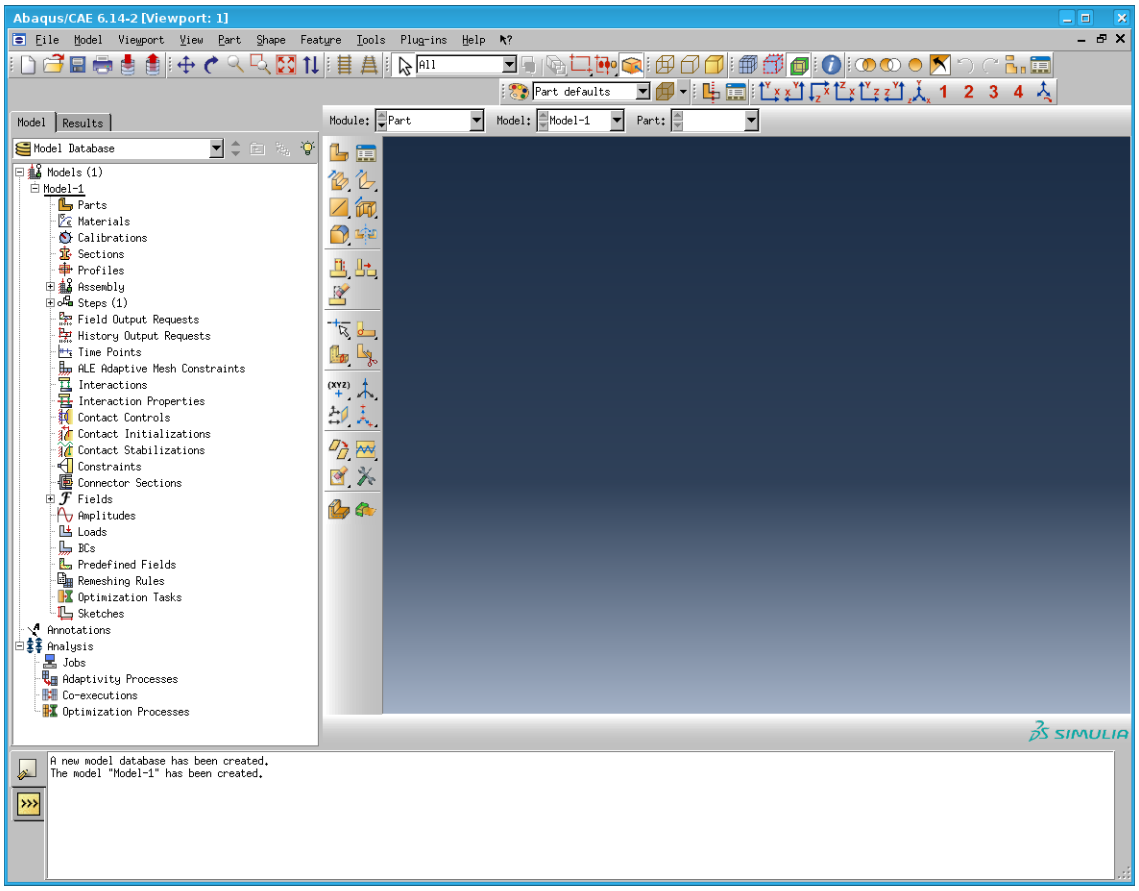
The *viewport* on the right hand side displays the model. The model can be inspected, selected and manipulated within this area. At the left border of the viewport are two columns of symbols which will be called *Quick start symbols*. Different view modes can be chosen in the drop-down menu right of `Module` above the viewport. They can also be invoked by selecting the appropriate mode in the model tree.

Messages, warnings and errors are shown in the *message log* at the bottom part of Abaqus. Beneath the message log is also a *command line* (in the »>-tab), where all commands (as shown in the journal file) can also be issued directly. In general, there are many options and commands in Abaqus, which can be invoked in more than one way.

The modules in Abaqus provie a workflow for model creation and simulation. They can either be changed by the drop-down menu named `Module` above the viewport or by clicking on the corresponding module in the model tree on the left.

At first, all required parts of the model are created individually in the *Part module* as described in chapter 2. All parts need to have material properties assigned to them, which will be covered in the *Material module* (chapter 3).

Afterwards the parts are assembled to the final model layout in the *Assembly module* (see chapter 4). After the assembly, necessary contact and interaction properties are defined (chapter 5) and boundary conditions as well as loads are defined and applied in the *Step module* (chapter 6).



```
A new model database has been created.
The model "Model-1" has been created.
```

Figure 1.2: Interface of Abaqus CAE. The main menu and two rows of shortcut symbols are near the top. The central area contains the model tree (left hand side) and the viewport (right hand side) with two columns of quick start symbols between both. The message log tab is located in the bottom area (in front of the command line tab).

Before continuing the parts have to be meshed in the *Mesh module* as described in chapter 7. Meshing can already be done after finishing the part geometries (chapter 2). However if partitions are added or changed in the assembly step, the mesh has to be redone.

hint » *In general, all modules can be edited (again) at any point, but sometimes the model has to be regenerated or otherwise updated to fit the changes.*

hint » *In this document, all general Abaqus hints are highlighted like this.*

When everything is done, a job can be created (see chapter 8) and the simulation can be run. Finally, the simulation results can be analyzed in the last module (chapter 9).

Author's remark:

These notes are based on discussions with my colleagues from the Institute of Geotechnical Engineering at the Hamburg University of Technology. Additionally, the official Abaqus manual(s) (Dassault Systèmes, 2014) as well as workshop slides (ABAQUS Inc., 2005) and the scripting notes from Overvelde (2010) were used.

2 Model creation

As Abaqus is starting, all required dependencies are loaded automatically for interactive work. But if a script with commands from a Journal- or Replay-file is run, these dependencies have to be imported at the beginning of the script.

```python
# -*- coding: utf-8 -*-
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
```

2.1 Create pile object

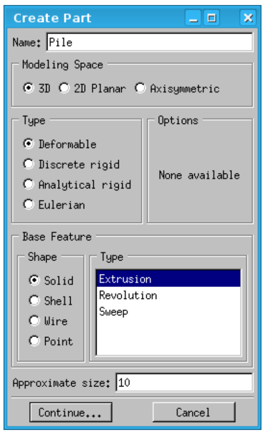
Abaqus is providing an initial model called `Model-1` at startup. First, choose an expressive name to the model like *PileJackingModel*. Either right click on `Model-1` and select `Rename` or use the main menu (`Model->Rename->Model-1`). Notice the change in the model tree and in the second drop down menu above the viewport.[1]

```python
mdb.models.changeKey(fromName='Model-1', toName='PileJackingModel');
```

Next, we create the parts of the model. To create a part, either right click on `Parts` and select `Create` in the model tree, select `Part->Create` in the main menu or use the Create Part icon ( ) in the top left corner of the quick start symbols. The dialog box in figure 2.1 will open.

Name the part *Pile* and accept the remaining default values. Since we are interested in a three dimensional analysis, 3D modelling space and a solid (deformable) shape are suitable. The approximate size and the type are used to create a sufficiently sized model out of a two dimensional sketch.

[1]Abaqus will add the shown Python command for renaming the model to the journal file. From here on, all Python commands represent either journal entries or more readable versions of the discussed action(s) described in the preceding text.


Create Part

Name: Pile

Modeling Space: [✓ 3D] 2D Planar Axisymmetric

Type: [✓ Deformable] Discrete rigid Analytical rigid Eulerian

Options: None available

Base Feature

Shape: [✓ Solid] Shell Wire Point

Type: [✓ Extrusion] Revolution Sweep

Approximate size: 10

Continue... Cancel

Figure 2.1: Create Part window. Assign a meaningful name to the part and decide its shape and type based on the requirements of the model.

Using `Extrusion` simply extends the outline of the sketch in the third dimension. `Revolution` would revolve the sketch around an axis and `Sweep` allows an outline to follow a (closed) path.

The background of the viewport changes to a grid and the quick start symbol adapt to the `Sketch module`. To draw a sketch, simple geometric elements can be used. All options can either be executed using the main menu (`Add->`) or the quick start symbols.

A single point can be drawn ( ), a polyline ( ), a rectangle ( ), a circle ( ) or an ellipse ( ). Circular arcs can be either drawn tangent to an existing line ( ), by a center point and two points on the arc ( ) or by three points ( ).

hint » *All coordinates can either be chosen on the visual grid or inserted in the prompt at the bottom of the viewport (above message log). Commands can either be executed by pressing the* `Enter`*-key or the middle mouse button (*`MMB`*).*

A sketch or its segments can also be moved with the cursor icon ( ) or deleted by the eraser icon ( ) in the lower region. To have a proper defined sketch/model, angles and lengths can and should be measured ( ) and constraints imposed ( ). The constraints function opens a new window with a set of selectable constraints. Sometimes it might be more practical to draw freely and get the right model by measurements and constraints.

hint » *Everything can be changed or removed at almost any time, but sometimes not with obvious solutions like* `Ctrl+Z`*,* `Del` *or* `Esc`*.*

Abaqus can also try to auto-constrain ( ) and auto-measure ( ) the sketch, but manually imposing at least some values and checking the validity is recommended. A simple rectangle with measurements and constraints representing the profile of the pile is shown in figure 2.2. Its area is 1 units (x-direction) by 0.5 units (y-direction).[2]

[2] Abaqus doesn't explicitly use units. Here, a length of 1 represents 1 m. The user has to take care of choosing the right unit system for all input values. Two possible unit systems are suggested in table 3.1.

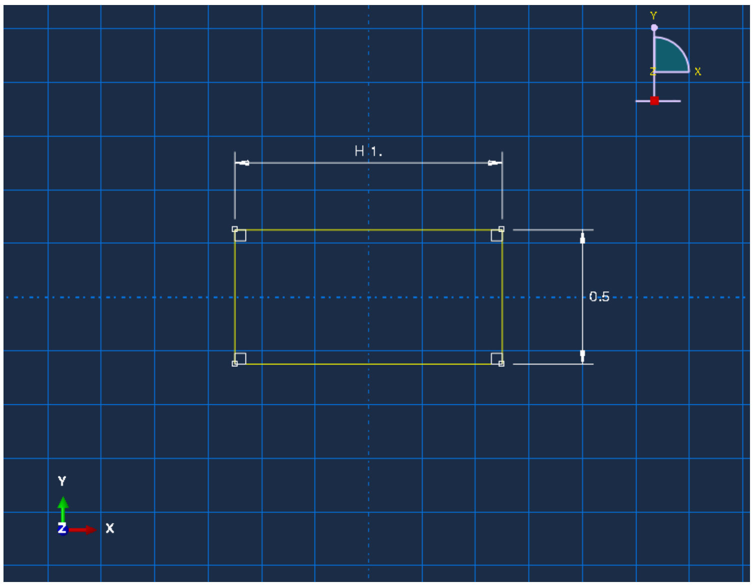


Figure 2.2: Finished sketch for pile in viewport, ready for extrusion.

hint » *When creating or manipulating objects, Abaqus will snap to the grid, to special points on the sketch and on all intersections.*

This can be problematic when working on more complex geometry. As a consequence it may be favourable to draw a construction line ( ) to create additional reference lines or artificial intersections to add additional segments more easily.

hint » *Some icons like this have more options to choose from, as hinted by the small black triangle in the lower right corner. Left click on such icons and hold the left mouse button pressed to see all available options. Release above the icon of choice to select it. The last selected symbol will be shown among the quick start symbols.*

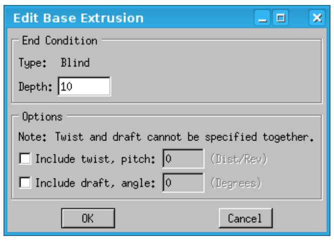
There are five ways to define a construction line ( ): Either by two points, or by one point (with horizontal or vertical constraint), or by one point and an angle or as a circle (by defining center point and radius).

hint » *If the functionality of an option is not known, move the mouse above the icon. An info box will appear with its name/description.*

Another useful function is auto-trimming ( ). Its main use is the removal of single segments between corners or intersections and will be very useful with more complex geometry made from/with simple elements.

The finished sketch will be extruded to become a part. On exiting the sketch editor the Extrude dialog opens (cf. figure 2.3). Set the depth of extrusion to 10 units and confirm. The main window should now look like figure 2.4.

All commands used to create the pile are listed below. First, a `ConstrainedSketch` element is created and a rectangle, defined by two points, upon it. In Line 23 and 26 measurements



Edit Base Extrusion

End Condition
Type: Blind
Depth: 10

Options
Note: Twist and draft cannot be specified together.
☐ Include twist, pitch: 0 (Dist/Rev)
☐ Include draft, angle: 0 (Degrees)

OK Cancel

Figure 2.3: Extrusion dialog. Besides extruding, a twist can be added to generate a screw (not used in this tutorial).

are added by `HorizontalDimension` and `VerticalDimension` commands.[3] When the sketch is finished, name, shape and type options as seen in figure 2.1 are chosen. When everything is applied and the part is successfully created, the temporary `ConstrainedSketch` is removed.[4]

```
20 drawing = mdb.models['PileJackingModel'].ConstrainedSketch(name='__profile__', sheetSize=20.0);
21 drawing.rectangle(point1=(-0.5, -0.25), point2=(0.5, 0.25));
22 drawing.HorizontalDimension(textPoint=(0.0, 0.5), value=1.0,
23    vertex1=drawing.vertices.findAt((-0.5, 0.25), ),
24    vertex2=drawing.vertices.findAt((0.5, 0.25), ));
25 drawing.VerticalDimension(textPoint=(0.75, 0.0), value=0.5,
26    vertex1=drawing.vertices.findAt((0.5, -0.25), ),
27    vertex2=drawing.vertices.findAt((0.5, 0.25), ));
28 partPile = mdb.models['PileJackingModel'].Part(dimensionality=THREE_D, name='Pile',
29    type=DEFORMABLE_BODY);
30 partPile.BaseSolidExtrude(depth=10.0, sketch=drawing);
31 del drawing;
```

2.2 Create soil object

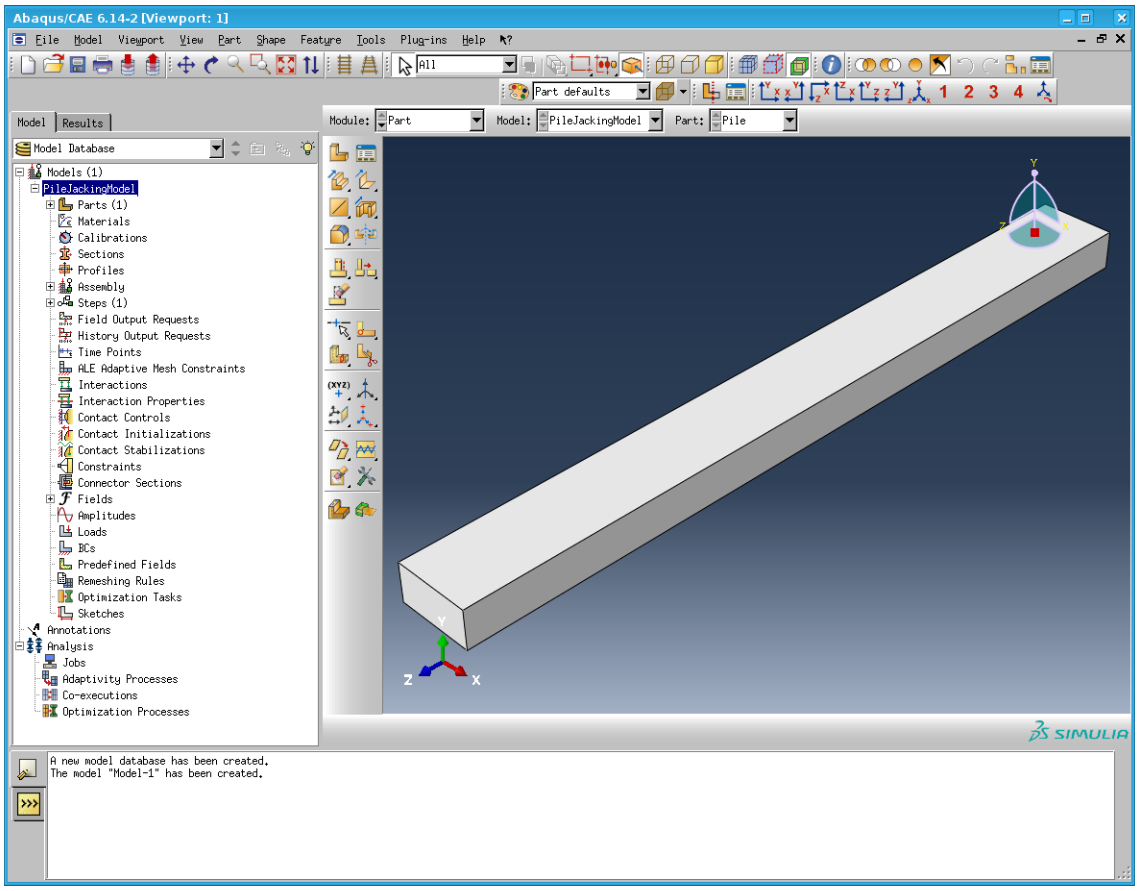
Now the soil part has to be modelled. The procedure is almost the same as described for the pile in the previous section. Create a new part and name it *Soil*. To make it an Eulerian part, choose the Eulerian type instead of Deformable in the Create Part window and accept the other default options. The sketch editor will open again and another rectangular profile should be made. This time the dimensions are 4 units (x-direction) by 2 units (y-direction) and an extrusion depth of 6 units (z-direction).

hint » *As always, save often to prevent the trouble of losing data.*

As a result the part should look like the one shown in figure 2.5. Here the element tree is expanded and in both parts a `Solid extrude-1` element and a `Section Sketch` element can be seen. Right click on `Section Sketch` and choose `Edit` to change the sketch of the element or edit `Solid extrude-1` to change the extrusion depth. If any changes are made in these settings, the whole part has to be regenerated, as a warning window will indicate.

[3] Regarding Python code: Abaqus usually references points, edges, … by its index number which is not very helpful in code. A better solution when using scripts is to use the `findAt`-function.

[4] Regarding Python code: To reference elements in the code more comfortably, we assign variable names at creation like `partPile` in line 28. If there is no assignment at creation time, it could also be done later on, e. g. `partPile = mdb.models[ṔileJackingModel]́.parts[Ṕile]́;`.



```
A new model database has been created.
The model "Model-1" has been created.
```

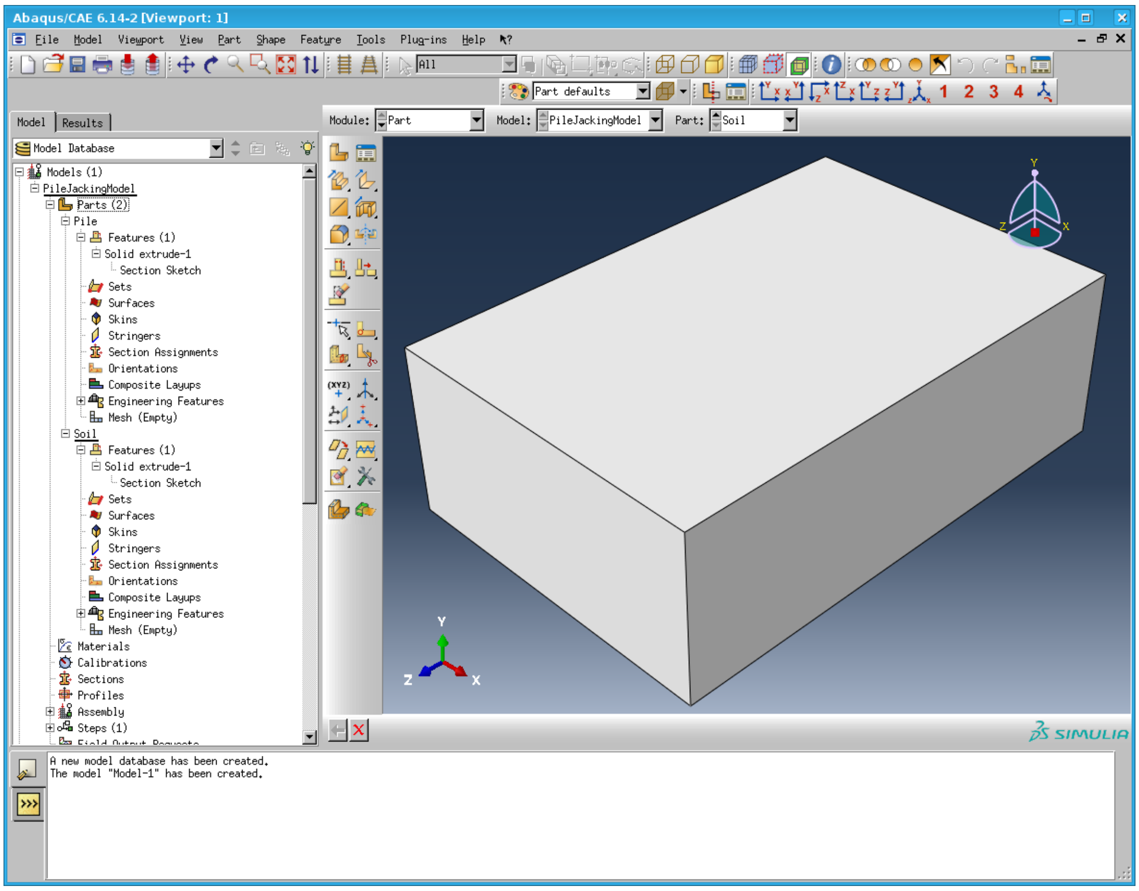
Figure 2.4: Main window after creating the pile part.

Right click on `Features` and select `Regenerate` of the corresponding part. Alternatively activate the corresponding part—if not active—and choose `Feature->Regenerate` from the main menu.

hint » *Active models/parts/instances/... are underlined. To activate an inactive element double click on it.*

It should not be surprising, that the journal data for the soil element is very similar to the pile part. The most important difference is the Eulerian part type (line 43-44).

```
35 drawing = mdb.models['PileJackingModel'].ConstrainedSketch(name='__profile__', sheetSize=20.0);
36 drawing.rectangle(point1=(-2.0, -1.0), point2=(2.0, 1.0));
37 drawing.HorizontalDimension(textPoint=(0.0, 1.5), value=4.0,
38    vertex1=drawing.vertices.findAt((-2.0, 1.0), ),
39    vertex2=drawing.vertices.findAt((2.0, 1.0), ));
40 drawing.VerticalDimension(textPoint=(2.5, 0.0), value=2.0,
41    vertex1=drawing.vertices.findAt((2.0, -1.0), ),
42    vertex2=drawing.vertices.findAt((2.0, 1.0), ));
43 partSoil = mdb.models['PileJackingModel'].Part(dimensionality=THREE_D, name='Soil',
44    type=EULERIAN);
45 partSoil.BaseSolidExtrude(depth=6.0, sketch=drawing);
46 del drawing;
```



Module: Part Model: PileJackingModel Part: Soil

```
A new model database has been created.
The model "Model-1" has been created.
```

Figure 2.5: Main window after creating the soil part. The model tree is expanded, the soil part is active and the child elements for sketch and extrusion control can be seen.

2.3 Adding and querying elements

This section applies to the `Part module` and the `Assembly module` described in chapter 4. In both modules it may be necessary to create special elements or to obtain information about the geometry. To obtain node coordinates or the distance between nodes—as well as many other useful information—select `Tools->Query` from the main menu and the Query window will open (cf. left side of figure 2.6).

Sometimes it is useful to create *datum points*, *axis* or *planes* to ease working with partitions or assembling elements later on. To create a datum element, either select `Tools->Datum` from the main menu to open the Create Datum window (cf. center of figure 2.6) or use the quick start symbols (lower midarea). You can create a datum point (), a datum axis() or a datum plane (). All datum elements can be created in different ways (hence the black triangle in the lower right corner of the icon).

Reference points are similar to datum points, but offer more possibilities. They can be coupled to geometric entities (e. g. sets, parts and instances) to define boundary conditions (movement, loads, . . .) or obtain measurement data. To create a reference point, use

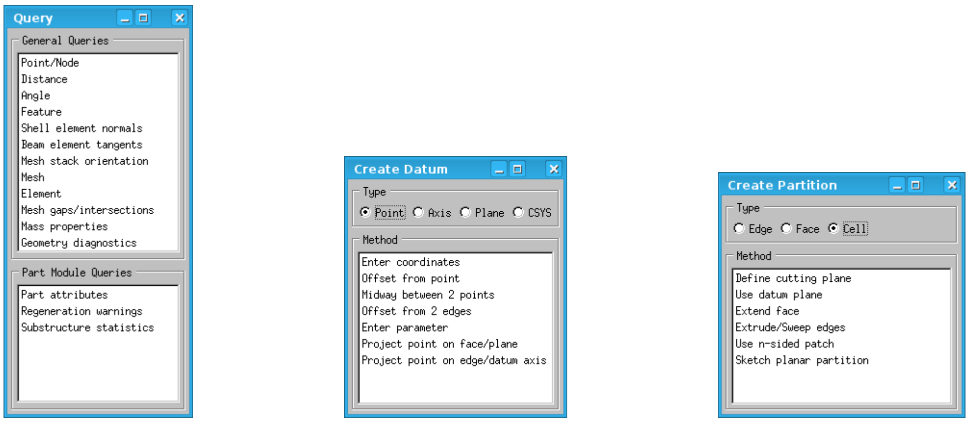


Figure 2.6: Query dialog on the left hand side and Create Datum window (with datum point options visible) in the center. The Create Partition window (with cell option) can be seen on the right hand side.

`Tools->Reference Point` from the main menu.

Sets in turn are useful to label elements or whole parts. This can be very helpful when imposing boundary conditions or other constraints on (parts of) a geometry. Sets can either be created by selecting `Tools->Set->Create` in the main menu or by right clicking on `Sets` in the model tree and choosing `Create`.

Partitions are essential for multiple material assignments and important for mesh generation. The Create Partition window can be invoked by `Tools->Partition` (cf. right hand side of figure 2.6).

Additionally, quick start symbols (central, above datum icons) are also available e. g. for creating cell partitions ( ) or face partitions ( ). It can be useful to sketch and create a face partition and extrude the partition through the cell(s) afterwards ( ).

hint » *Eulerian material can only flow within the given geometry. Therefore creating and assigning partitions with different materials (and void material) may be useful.*

Adjusting the pile part

These concepts should now be applied to the PileJackingModel. First, a reference point should be added to control the pile movement later on. Add it at the middle of longest edge on the top of the pile as shown in figure 2.7 and name it *RP_Pile*.

```
rp = partPile.ReferencePoint(point=(0.0, -0.25, 10.0));
partPile.features.changeKey(fromName='RP', toName='RP_Pile');
```

Then two sets should be added for the pile: The whole part (without the reference point) should be represented as a *setAll* as well as the reference point as *setRP*.

```
partPile.Set(name='setAll', cells=partPile.cells);
partPile.Set(name='setRP', referencePoints=(partPile.referencePoints[rp.id], ));
```

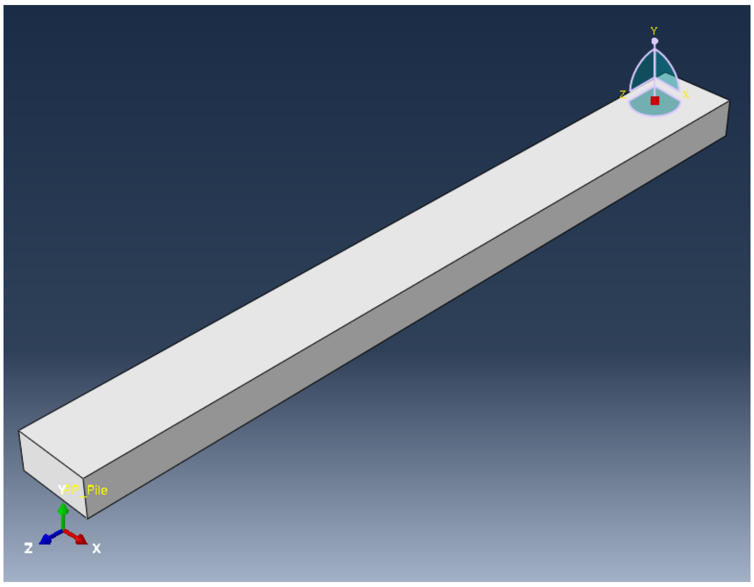


Figure 2.7: Pile with reference point at middle of longest edge on top.

Adjusting the soil part

Next, the soil should be partitioned (e. g. by cutting planes). One possibility to get the desired cutting plane is to create necessary datum point(s) first, which are on this plane.[5] In chapter 3, the lower two thirds of the soil volume will be filled with a sand material while the top third will contain void material. Additionally, the mesh of the soil around the pile should be denser than in the area farther away to have a better representation of the Eulerian volume near the pile.

So one partition should divide the soil in two equal parts in y-direction and two more partitions should separate the outer fourth in x-direction.[6] In the end, a result similar to figure 2.8 should be achieved.

In this scenario, two datum points are sufficient if correctly chosen. Additionally the three principal datum axis should be created.[7]

```
pt1 = partSoil.DatumPointByCoordinate(coords=(1.0, 1.0, 4.0));
pt2 = partSoil.DatumPointByCoordinate(coords=(-1.0, 0.0, 6.0));
xaxis = partSoil.DatumAxisByPrincipalAxis(principalAxis=XAXIS);
yaxis = partSoil.DatumAxisByPrincipalAxis(principalAxis=YAXIS);
zaxis = partSoil.DatumAxisByPrincipalAxis(principalAxis=ZAXIS);
```

The Partition can be created by a cutting plane. If one point on the plane is known and a normal can be selected, the option Point and Normal can be chosen.

[5]Only one datum point per cutting plane may be necessary, if the option `Point & Normal` is chosen instead of `3 Points`.

[6]For good simulations, the soil volume has to be big enough, such that boundary effects do not influence the results of the pile-soil interaction significantly.

[7]Regarding Python code: Abaqus names elements by its type and an increasing number like `Datum pt-1`. For easy access in code, either rename the elements or assign variable names at its creation.

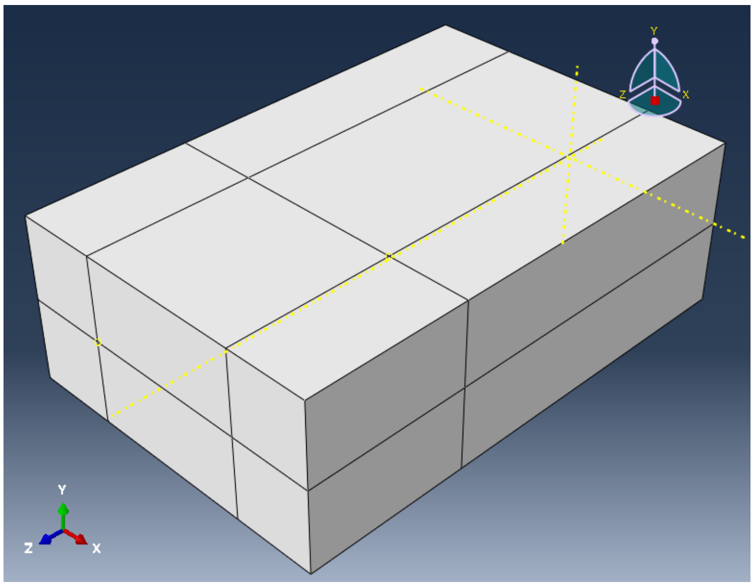


Figure 2.8: Soil element partitioned by four cutting planes.

```
68 partSoil.PartitionCellByPlanePointNormal(cells=partSoil.cells,
69    normal=partSoil.datums[xaxis.id], point=partSoil.datums[pt1.id]);
70 partSoil.PartitionCellByPlanePointNormal(cells=partSoil.cells,
71    normal=partSoil.datums[zaxis.id], point=partSoil.datums[pt1.id]);
72 partSoil.PartitionCellByPlanePointNormal(cells=partSoil.cells,
73    normal=partSoil.datums[xaxis.id], point=partSoil.datums[pt2.id]);
74 partSoil.PartitionCellByPlanePointNormal(cells=partSoil.cells,
75    normal=partSoil.datums[yaxis.id], point=partSoil.datums[pt2.id]);
```

Finally, some sets should be created for the soil part. The whole part should also be represented as a set (*setAll*). Next, all cells in the upper third should form *setCellsTop* and all cells in the two lower thirds *setCellsBottom*. Then, all (8) boundary faces in x-direction should be gathered in *setFacesX*, all (12) boundary faces in y-direction in *setFacesY* and all (6) bottom boundary faces in *setFacesBottom*.[8]

hint » *When using mouse selection to create sets, the vertices adjoining other cells outside the intended set mustn't be selected, but all edges and faces within.*

```
79 partSoil.Set(cells=partSoil.cells, name='setAll');
80 cellstop = partSoil.cells[0:0];
81 cellsbottom = partSoil.cells[0:0];
82 for singlecell in partSoil.cells:
83    if (singlecell.pointOn[0][2] > 4.0):
84       cellstop += partSoil.cells[singlecell.index:singlecell.index+1];
85    else:
86       cellsbottom += partSoil.cells[singlecell.index:singlecell.index+1];
87 #
88 partSoil.Set(cells=cellstop, name='setCellsTop');
89 partSoil.Set(cells=cellsbottom, name='setCellsBottom');
```

[8]Regarding Python code: Abaqus usually references sets/selections with a binary sequence masks by `getSequenceFromMask(('[#25b ]', ), )`. To select the correct geometry in code, sequences have to be generated (e. g. by iterating over elements as in line 82–86) to create such sets.

```
facesX = partSoil.faces[0:0];
facesY = partSoil.faces[0:0];
facesbottom = partSoil.faces[0:0];
for singleface in partSoil.faces:
   if (singleface.pointOn[0][2] <= 0.0):
      facesbottom += partSoil.faces[singleface.index:singleface.index+1];
   if (abs(singleface.pointOn[0][0]) >= 2.0):
      facesX += partSoil.faces[singleface.index:singleface.index+1];
   if (abs(singleface.pointOn[0][1]) >= 1.0):
      facesY += partSoil.faces[singleface.index:singleface.index+1];
#
partSoil.Set(faces=facesX, name='setFacesX');
partSoil.Set(faces=facesY, name='setFacesY');
partSoil.Set(faces=facesbottom, name='setFacesBottom');
```

Sometimes a single part is too complex to be represented by a simple sketch. But Abaqus supports creating complex part out of multiple simple parts in the assembly as mentioned in chapter 4. Also, geometry can be created in a different program and imported in Abaqus (`File->Import->Part`).

3 Material assignment

Enter the `Material module` by either right clicking on `Materials` in the model tree and selecting `Switch context`, or by choosing the `Property` entry in the Module drop-down menu above the viewport. Many quick start symbols change, but the datum and partition option (as well as the feature options which aren't discussed here) stay the same.

First, we define a new material by either clicking on ( ) or right clicking on `Materials` in the model tree and selecting `Create` or by choosing `Material->Create` in the main menu to open the Edit Material window.

Our first material will be steel. After naming the material, several options can be added to define its behavior. Most importantly its density (`General->Density`) and its elastic properties (`Mechanical->Elasticity->Elastic`). As hinted in section 2.1:

hint » *All dimensional values need to have fitting units. Choose a consistent system like the SI or the Geotechnics system as shown in table 3.1.*

Table 3.1: Consistent unit systems

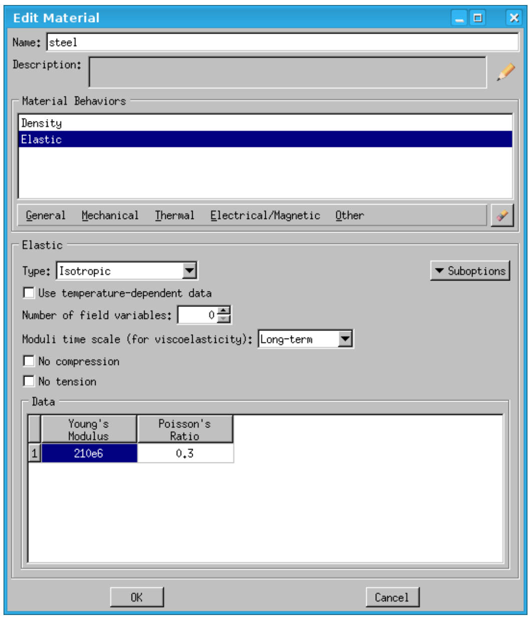
System	Length	Force	Mass	Time	Stress	Energy	Density
	[L]	$[\frac{ML}{T^2}]$	[M]	[T]	$[\frac{M}{LT^2}]$	$[\frac{ML^2}{T^2}]$	$[\frac{M}{L^3}]$
SI	m	N	kg	s	Pa $(\frac{N}{m^2})$	J	$\frac{kg}{m^3}$
Geotechnics	m	kN	t	s	kPa $(\frac{kN}{m^2})$	kJ	$\frac{t}{m^3}$

This example will use the Geotechnics system. Every unit for previously set lengths for the model parts created in chapter 2 will be regarded as meters. Accordingly to this unit system, the value for the density of steel will be $7.87\,t/m^3$, the Young's modulus $210 \cdot 10^6$ kPa and Poisson's ratio 0.3. Enter these values in the respective property tab (cf. figure 3.1) and click `Ok` to save the properties and close the window.

```
mdb.models['PileJackingModel'].Material(name='steel');
mdb.models['PileJackingModel'].materials['steel'].Density(table=((7.87, ), ));
mdb.models['PileJackingModel'].materials['steel'].Elastic(table=((210e6, 0.3), ));
```

Now the sand material has to be defined in the same way. Its density is $1.6\,t/m^3$ and it is assumed to be linear elastic (Young's modulus $30 \cdot 10^3$ kPa and Poisson's ratio 0.25).

```
mdb.models['PileJackingModel'].Material(name='sand');
mdb.models['PileJackingModel'].materials['sand'].Density(table=((1.6, ), ));
mdb.models['PileJackingModel'].materials['sand'].Elastic(table=((30e3, 0.25), ));
```



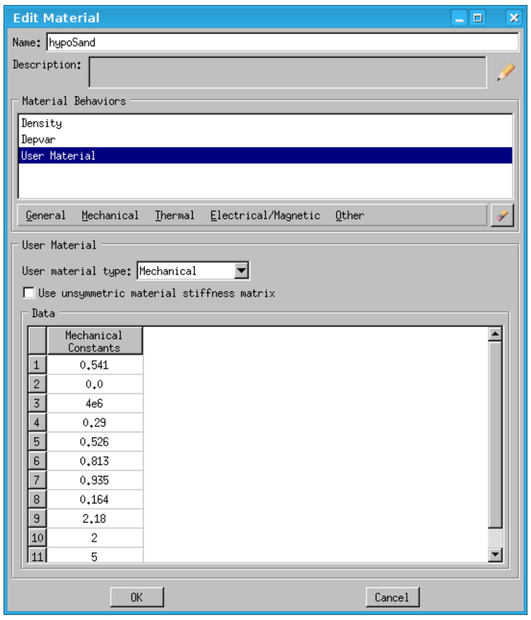
Name: steel
Description:

Material Behaviors
Density
Elastic

Elastic
Type: Isotropic
☐ Use temperature-dependent data
Number of field variables: 0
Moduli time scale (for viscoelasticity): Long-term
☐ No compression
☐ No tension

Data

	Young's Modulus	Poisson's Ratio
1	210e6	0.3



Name: hypoSand
Description:

Material Behaviors
Density
Depvar
User Material

User Material
User material type: Mechanical
☐ Use unsymmetric material stiffness matrix

Data

	Mechanical Constants
1	0.541
2	0.0
3	4e6
4	0.29
5	0.526
6	0.813
7	0.935
8	0.164
9	2.18
10	2
11	5

Figure 3.1: The material dialog with density and elastic behavior options added on the right hand side works internally. On the left hand side is an example of integrating a user defined material (not used in this tutorial).

If a user defined material behavior (e. g. hypoplastic behaviour) should be simulated, the amount of solution-dependent variables has to be defined (`General->Depvar`) and the values given to the user routine (`General->User Material`) as shown on the right hand side of figure 3.1. This will not be used in this tutorial.

All necessary materials are defined and after creating sections, they can be applied to the geometry. So for all parts, a new section is created by either right clicking on `Sections` in the model tree and choosing `Create`, selecting `Section->Create` in the main menu or using the quick start symbol (🔧).

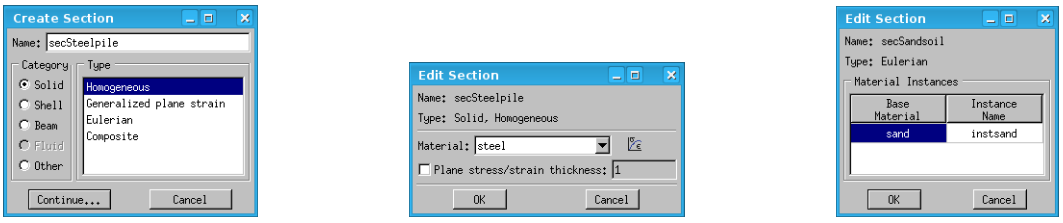
As the Create Section window opens, the material name, type and category can be edited. Choose *secPile* for the section associating the steel material with the pile geometry. Since the pile geometry is homogeneous and solid, no changes are necessary here. Clicking on `Continue` opens the Edit Section window (cf. figure 3.2).

```
mdb.models['PileJackingModel'].HomogeneousSolidSection(name='secPile', material='steel',
   thickness=None);
```

Repeat the process but this time name the section *secSoil* and choose an Eulerian type. This time a base material has to be chosen in the Edit Section window.

```
mdb.models['PileJackingModel'].EulerianSection(name='secSoil', data={'instsand': 'sand'});
```

Finally the material sections have to be assigned to the geometry. To assign the pile geometry to the steel material, make the pile the active element (underlined name) and choose `Assign->Section` in the main menu, right click on `Section Assignments` and


Create Section
Name: secSteelpile
Category: (•) Solid ( ) Shell ( ) Beam ( ) Fluid ( ) Other
Type: [✓ Homogeneous] Generalized plane strain Eulerian Composite
Continue... Cancel

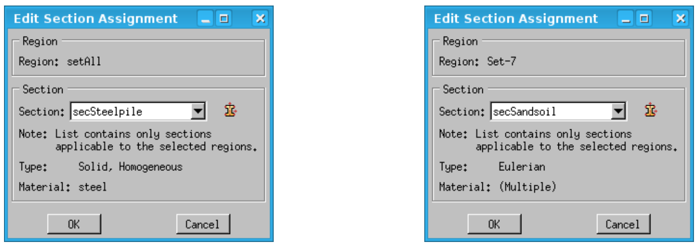
Edit Section
Name: secSteelpile
Type: Solid, Homogeneous
Material: steel
☐ Plane stress/strain thickness: 1
OK Cancel

Edit Section
Name: secSandsoil
Type: Eulerian
Material Instances

Base Material	Instance Name
sand	instsand

OK Cancel

Figure 3.2: The Section Create on the left hand side and two Section Edit dialogs. The dialog in the center is called when homogeneous types are chosen, the dialog on the right hand side for Eulerian types.


Edit Section Assignment
Region
Region: setAll
Section
Section: secSteelpile
Note: List contains only sections applicable to the selected regions.
Type: Solid, Homogeneous
Material: steel
OK Cancel

Edit Section Assignment
Region
Region: Set-7
Section
Section: secSandsoil
Note: List contains only sections applicable to the selected regions.
Type: Eulerian
Material: (Multiple)
OK Cancel

Figure 3.3: Section Assignment of steel material for pile and sand material for soil.

selecting `Create` or by the quick start symbol ( ). A new region can be selected or a previously defined set can be loaded below the viewport. Click on `Sets` to select the *setAll* set.

Choose the appropriate section (*secPile*) in the section drop-down menu of the Edit Section Assignment window and the assignment is complete. (see left hand side of figure 3.3). Only applicable sections will be displayed and can be assigned.

hint » *Correctly assigned geometry will change its color from grey to pale green.*

Generally, the assignment procedure should be repeated for all other elements. When dealing with Eulerian elements (like this soil element), the region is set automatically and a new set is created when an Eulerian material is defined.[1] ,[2] Eulerian elements can have different materials, which have to be assigned as a predefined field (see chapter 6), but the basic material assignment is done.

```
partPile.SectionAssignment(sectionName='secPile', region=partPile.sets['setAll'], offset=0.0,
   offsetField='', offsetType=MIDDLE_SURFACE, thicknessAssignment=FROM_SECTION);
partSoil.SectionAssignment(sectionName='secSoil', region=partSoil.sets['setAll'], offset=0.0,
   offsetField='', offsetType=MIDDLE_SURFACE, thicknessAssignment=FROM_SECTION);
```

[1]The region of the Eulerian element can still be changed, if desired, but only after the initial assignment.
[2]Regarding Python code: There is no automatic set creation/selection when scripting. Therefore no new set like `Set-7` in figure 3.3 is created and the existing `setAll` is chosen for assignment.

4 Assembly

After all parts are finished, the model itself can be assembled. Either right click on `Assembly` in the model tree and select `Switch Context` or choose `Assembly` from the drop-down menu named `Module` above the viewport. The quick start symbols contain eight special assembly icons and some already known ones (features, datums and partitions).

An assembled model (represented in blue) consists of different instances. One instance can be made from one part, but one part can also be used to create multiple instances. All instances can be transformed (rotation and translation) and aligned with others.

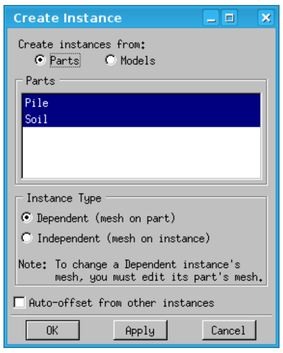
To create an instance, either right click on `Instances` in the model tree and select `Create`, choose `Instance->Create` in the main menu or click on the quick start symbol ( ). Select both parts in the Create Instance window and click `Ok` (cf. figure 4.1).[1]

```
132 mdb.models['PileJackingModel'].rootAssembly.DatumCsysByDefault(CARTESIAN);
133 instPile = mdb.models['PileJackingModel'].rootAssembly.Instance(name='instPile', part=partPile,
134     dependent=ON);
135 instSoil = mdb.models['PileJackingModel'].rootAssembly.Instance(name='instSoil', part=partSoil,
136     dependent=ON);
```

Now the instances have to be transformed to get the right model composition. The initial coordinates of the instances are dependent on the corresponding part coordinates. So considering the part's position in the model when creating it makes assembling the model more convenient.

Later on, the upper third of the soil volume will contain void material and the rest will be

[1]Regarding Python code: For easier future acces of both instances, variable names `instSoil` and `instPart` will be assigned in the code (line 133 and 135).


Create Instance

Create instances from:
[✓ Parts] Models

Parts
Pile
Soil

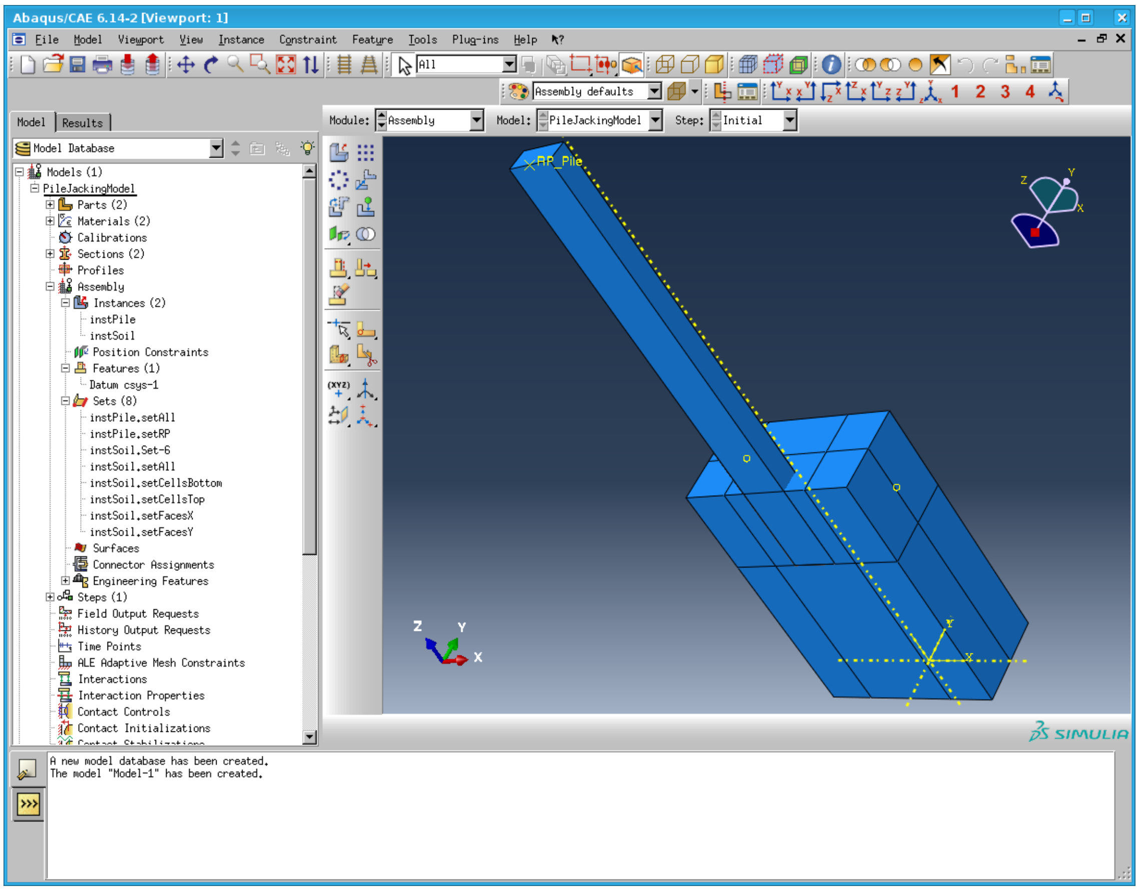
Instance Type
[✓ Dependent (mesh on part)]
Independent (mesh on instance)

Note: To change a Dependent instance's mesh, you must edit its part's mesh.

Auto-offset from other instances

OK Apply Cancel

Figure 4.1: Create instance window with both parts selected.



Module: Assembly Model: PileJackingModel Step: Initial

```
A new model database has been created.
The model "Model-1" has been created.
```

Figure 4.2: Final assembly of pile and soil geometry.

filled with sand material. The pile should be positioned just above the soil (aligned with the plane of the horizontal partition), centered in x-direction and at the back of the soil in y-direction like in figure 4.2.

To translate an instance, either choose `Instance->Translate` from the main menu or use the quick start symbol ( ). After selecting the instance(s) to translate, either select start and end point in the viewport or input the coordinates. When using coordinate input, only the relative vector is important.[2]

```
mdb.models['PileJackingModel'].rootAssembly.translate(instanceList=('instPile', ),
    vector=(0.0, -0.75, 4.0));
```

If a rotation should to be applied, either choose `Instance->Rotate` from the main menu or use the quick start symbol ( ). A rotation axis by two points has to be defined, before the rotation angle can be chosen.

Another important function for the model setup is to add geometric constraints between instances ( ). Either choose an option from `Constraint->` in the main menu or use one of the seven quick start symbols ( ) to make faces parallel, move face

[2]Therefore it can be useful to use `(0.0, 0.0, 0.0)` as start vector and define the desired translation as end vector.

to face, make edges parallel, move edge to edge, make elements coaxial, coincident point or make some CSYS[3] parallel.

Sometimes a sketched part in the `Part module` cannot be made as complex as needed, but all geometric features can be created separately. The complex structure can be assembled in the `Assembly module` by translating, rotating and aligning all parts involved. Using the Merge/Cut function (`Instance->Merge/Cut`) or the quick start symbol (), a new and more complex part can be created from those instances and used thereafter.

[3]CSYS: user defined coordinate system

5 Interactions and constraints

General interactions and constraints are accessible in the `Interaction module` in the `Module` drop-down menu above the viewport or by right clicking on either `Interactions` or `Constraints` in the model tree and choosing `Switch Context`.

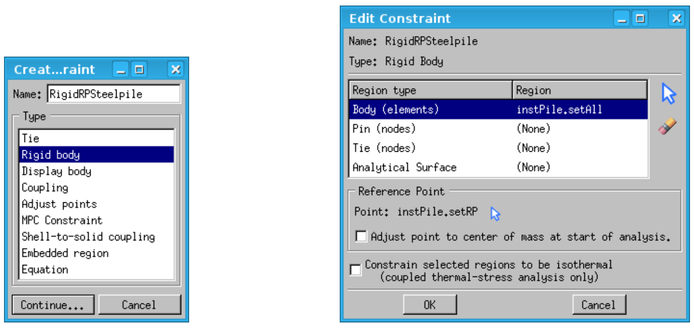
As a first constraint, all movement and velocity (as defined in chapter 6) of the pile should be controlled by the reference point *RP_Pile*. A quick start symbol for creating reference points ( ) is available in this module, but we have already defined one.

To create a constraint, either click on the quick start symbol ( ), right click on `Constaints` in the model tree and choose `Create` or select `Constraint->Create` from the main menu. The Create constraint window will open and in this case, a `Rigid body` constraint should be applied, since the pile is much stiffer than the soil and its deformations are negligible in comparison. As an advantage, the simulation will finish sooner as if the steel pile deformations (and therefore the degrees of freedom of each individual node of the pile) were considered.

To define the `Rigid body` constraint, a (body) region and a reference point have to be selected by clicking on the cursor symbol. Since the pile and the reference point are already defined as sets, these definitions can be used (cf. figure 5.1).

```
mdb.models['PileJackingModel'].RigidBody(bodyRegion=instPile.sets['setAll'],
    name='RigidRPSteelpile', refPointRegion=instPile.sets['setRP']);
```

All elements or surfaces which (could or should) interact with each other need to have definitions for contact and interaction. Therefore an interaction is added by either selecting `Interaction->Create` in the main menu, the quick start symbol ( ) or by right clicking


Creat...raint

Name: RigidRPSteelpile

Type
- Tie
- Rigid body
- Display body
- Coupling
- Adjust points
- MPC Constraint
- Shell-to-solid coupling
- Embedded region
- Equation

Continue... Cancel

Edit Constraint

Name: RigidRPSteelpile
Type: Rigid Body

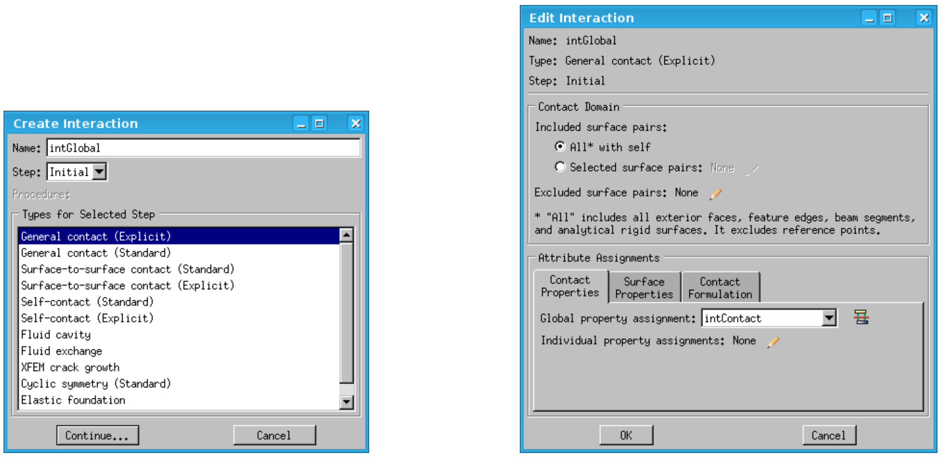
Region type	Region
Body (elements)	instPile.setAll
Pin (nodes)	(None)
Tie (nodes)	(None)
Analytical Surface	(None)

Reference Point
Point: instPile.setRP
☐ Adjust point to center of mass at start of analysis.

☐ Constrain selected regions to be isothermal (coupled thermal-stress analysis only)

OK Cancel

Figure 5.1: The Create Constraint window on the left hand side and the Edit Constraint window on the right hand side.



Name: intGlobal
Step: Initial
Procedure:
Types for Selected Step
General contact (Explicit)
General contact (Standard)
Surface-to-surface contact (Standard)
Surface-to-surface contact (Explicit)
Self-contact (Standard)
Self-contact (Explicit)
Fluid cavity
Fluid exchange
XFEM crack growth
Cyclic symmetry (Standard)
Elastic foundation

Name: intGlobal
Type: General contact (Explicit)
Step: Initial
Contact Domain
Included surface pairs:
[✓ All* with self]
Selected surface pairs: None
Excluded surface pairs: None
* "All" includes all exterior faces, feature edges, beam segments, and analytical rigid surfaces. It excludes reference points.
Attribute Assignments
Global property assignment: intContact
Individual property assignments: None

Figure 5.2: The Create Interaction window on the left hand side and the Edit Interaction window after defining the interaction property on the right hand side.

on `Interactions` in the model tree and choosing `Create`. The Create Interaction window will open as shown in the left hand side of figure 5.2.

Some interactions can be used for Standard and Explicit analysis, but others only for one. For simple FEM examples, surface-to-surface contact is often the best choice, but a simpler (and computationally more expensive way) is General contact. This example will use an Explicit analysis where Eulerian and Lagrangian elements interact (CEL). Therefore General contact (Explicit) has to be selected.

hint » *The interactions and the steps in the next chapter have to be chosen consistent with the type of analysis to be done.*

For this General contact example, the interaction should be able to have contact with any other element (select `All * with self` option) Now a global property assignment has to be defined, before the interaction can be created.

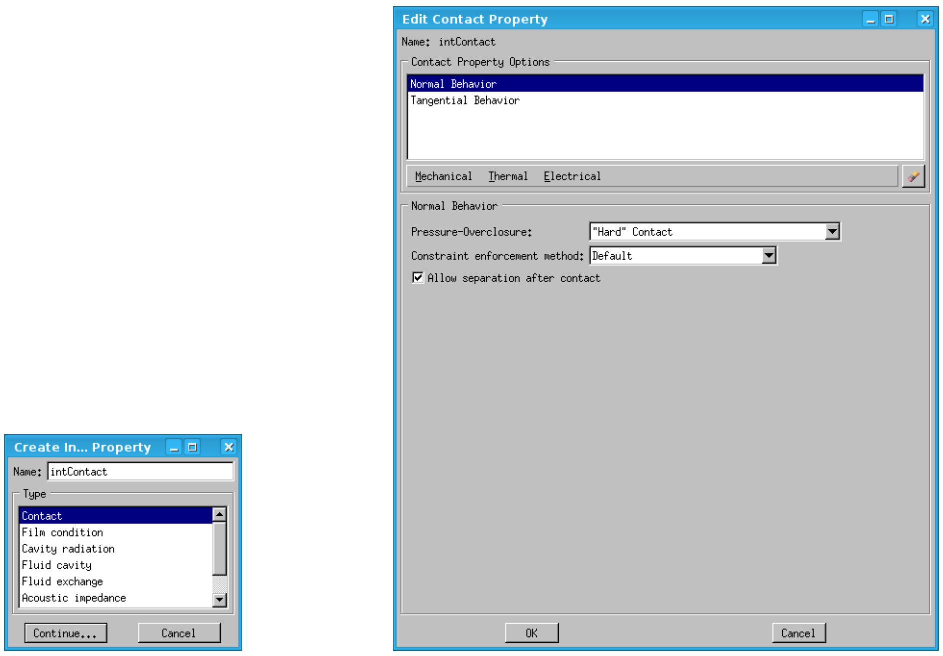
Click on the Create Interaction Property icon ( ) and select Contact to define a contact interaction property.[1] To simulate the mechanical interaction between the instances, the normal contact behaviour (`Mechanical->Normal behavior`) and the tangential contact behavior (`Mechanical->Tangential behavior`) should be added.[2]

In this example the default options for both will do, i. e. frictionless behaviour in tangential direction and hard contact in normal direction (cf. figure 5.3). The resulting model is shown in figure 5.4.

hint » *Small circles on the respective geometry symbolize applied constraints.*

[1] If this step had been done before creating an interaction, the property would already be available in the drop-down menu.

[2] If friction is to be modelled, the soil–pile interaction can be described by Coulomb's law with a penalty method.



Name: intContact

Type

- Contact
- Film condition
- Cavity radiation
- Fluid cavity
- Fluid exchange
- Acoustic impedance

Name: intContact

Contact Property Options

- Normal Behavior
- Tangential Behavior

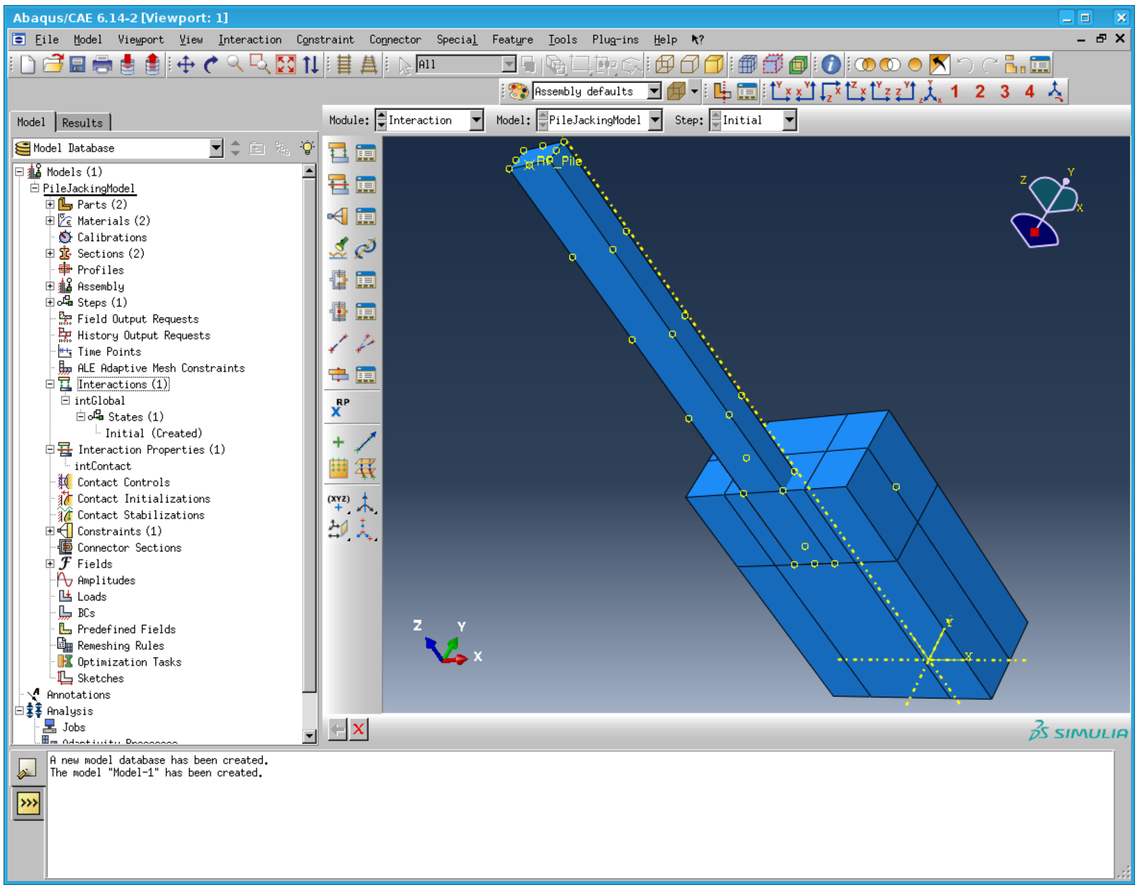
Normal Behavior

Pressure-Overclosure: "Hard" Contact

Constraint enforcement method: Default

☑ Allow separation after contact

Figure 5.3: The Create Interaction Property window on the left hand side and the Edit Contact Property window on the right hand side.



Module: Interaction   Model: PileJackingModel   Step: Initial

Model Database

```
Models (1)
  PileJackingModel
    Parts (2)
    Materials (2)
    Calibrations
    Sections (2)
    Profiles
    Assembly
    Steps (1)
    Field Output Requests
    History Output Requests
    Time Points
    ALE Adaptive Mesh Constraints
    Interactions (1)
      intGlobal
        States (1)
          Initial (Created)
    Interaction Properties (1)
      intContact
    Contact Controls
    Contact Initializations
    Contact Stabilizations
    Constraints (1)
    Connector Sections
    Fields
    Amplitudes
    Loads
    BCs
    Predefined Fields
    Remeshing Rules
    Optimization Tasks
    Sketches
Annotations
Analysis
  Jobs
```

```
A new model database has been created.
The model "Model-1" has been created.
```

Figure 5.4: Abaqus window after applying the interactions and constraints.

```
mdb.models['PileJackingModel'].ContactProperty('intContact');
mdb.models['PileJackingModel'].interactionProperties['intContact'].NormalBehavior(
   allowSeparation=ON, constraintEnforcementMethod=DEFAULT, pressureOverclosure=HARD);
mdb.models['PileJackingModel'].interactionProperties['intContact'].TangentialBehavior(
   formulation=FRICTIONLESS);
mdb.models['PileJackingModel'].ContactExp(createStepName='Initial', name='intGlobal');
mdb.models['PileJackingModel'].interactions['intGlobal'].includedPairs.setValuesInStep(
   stepName='Initial', useAllstar=ON);
mdb.models['PileJackingModel'].interactions['intGlobal'].contactPropertyAssignments.appendInStep(
   assignments=((GLOBAL, SELF, 'intContact'), ), stepName='Initial');
```

6 Steps

6.1 Creating steps

After the model geometry is set, it is time to define and apply boundary conditions and loads on the model in the `Step module`. This module can be accessed by either right clicking on `Steps` and choosing `Switch context` in the model tree or choosing `Step` in the `Module` drop-down menu above the viewport.

Steps are created to define a sequence of actions performed on the model. A model always has an initial step to define an original state. It is also possible to define boundary conditions, more interactions and predefined fields for this first step. Every additional step has to have a duration[1] and can define custom loads.

To create a new step, either click on `Steps` and choose `Create` in the model tree, select `Step->Create` from the main menu or click on its quick start symbol ( ). The Create Step window will open (cf. figure 6.1).

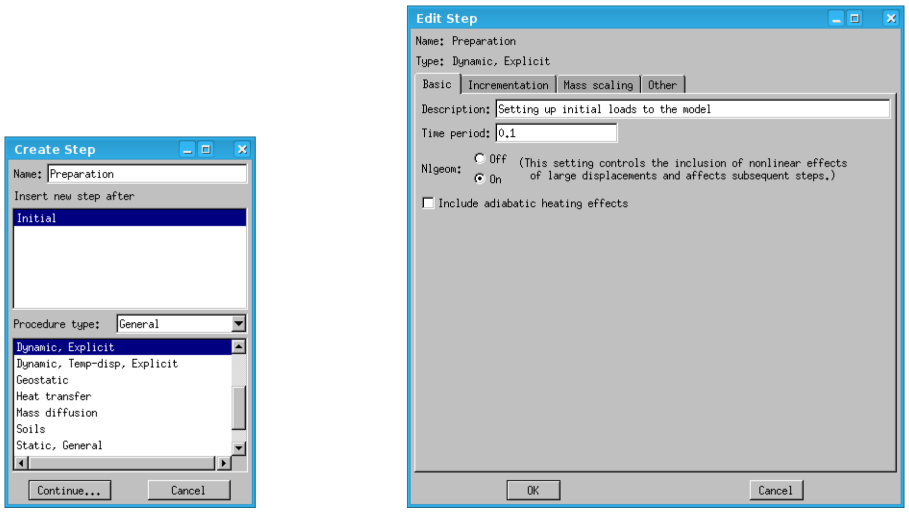
The first step will be named *Preparation* and can be used to set up initial loads like gravity. In this example, its type should be set to `Dynamic, Explicit` to be compatible with the next step. In the Edit Step window, an optional description can be inserted and the time period has to be chosen (here 0.1 s).

Now the same procedure has to be repeated for another `Dynamic, Explicit` step named *PileJacking*, where the pile will be driven into the soil within 3 s. A reasonable duration is depending on the loads applied in a step, especially if they are explicitly time dependent (like velocities).

The decision whether to use `Standard` or `Explicit` analysis is depending on the problem itself. As a rule of thumb, `Explicit` is favored for large, nonlinear problems, fast dynamics, or problems with significant discontinuities, while `Standard` is rather used for linear (dynamic) and nonlinear static problems (cf. ABAQUS Inc., 2005). To use the CEL method with Lagrangian and Eulerian elements, `Explicit` has to be chosen.

```
mdb.models['PileJackingModel'].ExplicitDynamicsStep(name='Preparation', previous='Initial',
    description='Preparation step', timePeriod=0.1);
mdb.models['PileJackingModel'].ExplicitDynamicsStep(name='PileJacking', previous='Preparation',
    description='Jacking the pile into the soil', timePeriod=3.0);
```

[1]The duration is also dimensionless, but in the unit system used for this example, it matches a second.


Create Step

Name: Preparation

Insert new step after

Initial

Procedure type: General

Dynamic, Explicit
Dynamic, Temp-disp, Explicit
Geostatic
Heat transfer
Mass diffusion
Soils
Static, General

Continue... Cancel

Edit Step

Name: Preparation
Type: Dynamic, Explicit

Basic | Incrementation | Mass scaling | Other

Description: Setting up initial loads to the model

Time period: 0.1

Nlgeom: Off [✓ On] (This setting controls the inclusion of nonlinear effects of large displacements and affects subsequent steps.)

☐ Include adiabatic heating effects

OK Cancel

Figure 6.1: The Create Step window and the Edit Step window for the *Preparation* step.

6.2 Boundary conditions, loads and predefined fields

Within each step, boundary conditions and loads will be applied. Therefore, the `Load module` has to be selected, either by the drop-down menu above the viewport or by right clicking on `Loads` or `BCs` within a step and choosing `Switch context`. Each step will have its own set of loads and boundary conditions.

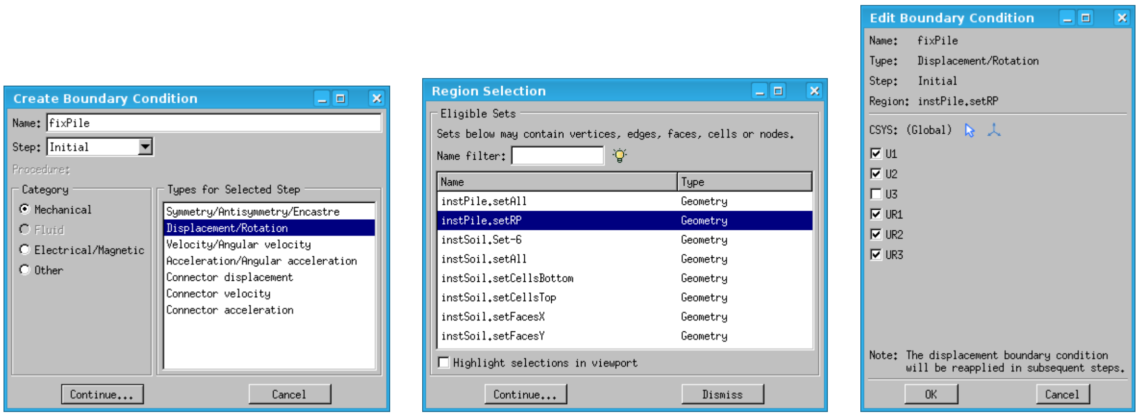
hint » *Some options like Interactions, Contact, Loads and BCs are available in each step and can be found on a higher level in the model tree below the Steps entry. Within the steps, those options can be defined, turned on and turned off. They are saved within the higher level entries, where they can be defined globally or deleted.*

To define boundary conditions, either select `BC->Create` in the main menu, right click on `BCs` within the inital step in the model tree and select `Create` or use the quick start symbol ( ). The Create Boundary Condition dialog will open (cf. figure 6.2).

The first boundary condition to be defined in the *Initial step* will be the fixation of the pile in all directions. Thus a `Displacement/Rotation` boundary condition should be applied. It can be more convenient to create two boundary conditions, one to fix all directions but the vertical one (which is valid for the whole simulation) and one to fix the downward movement in the initial step.[2]

The movement of the pile is constrained to the movement of the reference point, so all loads/boundary conditions of the pile will be defined for *setRP* of *instPile*. The corresponding dialogs for the permanent fixation (*fixPile*) are shown in figure 6.2.

[2]It is equally valid to restrict all movement in the Initial step with one boundary condition and modify it whenerver something changes in any following step.



Name: fixPile
Step: Initial
Procedure:
Category
(● Mechanical) Fluid Electrical/Magnetic Other
Types for Selected Step
Symmetry/Antisymmetry/Encastre
[✓ Displacement/Rotation]
Velocity/Angular velocity
Acceleration/Angular acceleration
Connector displacement
Connector velocity
Connector acceleration

Eligible Sets
Sets below may contain vertices, edges, faces, cells or nodes.
Name filter:

Name	Type
instPile.setAll	Geometry
[✓ instPile.setRP]	Geometry
instSoil.Set-6	Geometry
instSoil.setAll	Geometry
instSoil.setCellsBottom	Geometry
instSoil.setCellsTop	Geometry
instSoil.setFacesX	Geometry
instSoil.setFacesY	Geometry

☐ Highlight selections in viewport

Name: fixPile
Type: Displacement/Rotation
Step: Initial
Region: instPile.setRP
CSYS: (Global)
☑ U1
☑ U2
☐ U3
☑ UR1
☑ UR2
☑ UR3
Note: The displacement boundary condition will be reapplied in subsequent steps.

Figure 6.2: Boundary condition dialogs. After choosing to create a boundary condition, a type must be selected (left hand side), the geometry chosen (middle) and the values selected (right hand side).

```
mdb.models['PileJackingModel'].DisplacementBC(name='fixPile', createStepName='Initial',
   amplitude=UNSET, distributionType=UNIFORM, fieldName='', region=instPile.sets['setRP'],
   localCsys=None, u1=SET, u2=SET, u3=UNSET, ur1=SET, ur2=SET, ur3=SET);
```

hint » *When using the default (global) coordinate system* `CSYS: (GLOBAL)`, *the directions will match the standard coordinates:* $1 = x$, $2 = y$, $3 = z$.

Additionally create another `Displacement/Rotation` boundary condition named *fixDownwardMovement* in the intial step, which only fixates `U3`.

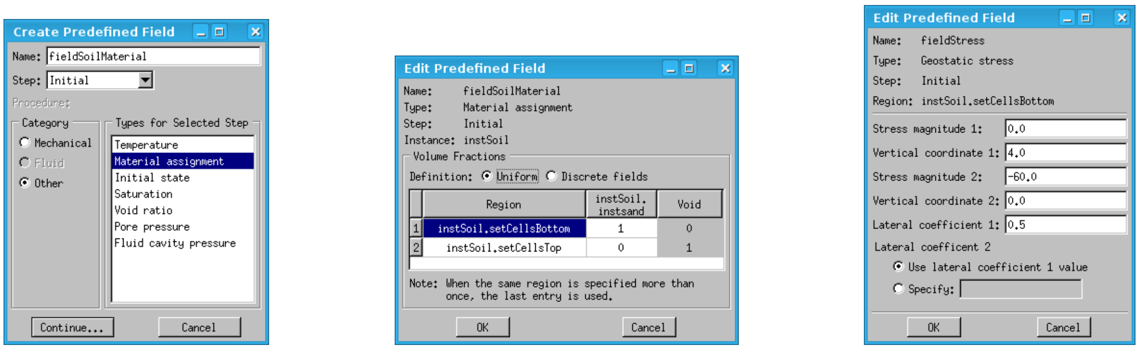
```
mdb.models['PileJackingModel'].DisplacementBC(name='fixDownwardMovement', amplitude=UNSET,
   createStepName='Initial', distributionType=UNIFORM, fieldName='', localCsys=None,
   region=instPile.sets['setRP'], u1=UNSET, u2=UNSET, u3=SET, ur1=UNSET, ur2=UNSET, ur3=UNSET);
```

In the *PileJacking step* the pile should be driven down 3 m. The duration for this step is 3 s, so the downward velocity has to be 1 m/s. Since the soil itself is only 4 m high in this example, the boundary effects will notably influence the result. Normally, there is much more space around the pile to minimize all boundary effects.

Right click on `fixDownwardMovement (Propagated)` in the current step's boundary conditions in the model tree and select `Deactivate` to deactivate this boundary condition from this step on forward. This can also be done in the Boundary Condition manager, which can be accessed either by the quick start symbol ( ) next to the create boundary condition icon or by selection `BC->Manager` in the main menu.

Create a `Velocity/Angular velocity` boundary condition for *setPileRP* named *drivePile* and enter the downward velocity `V3: -1`.

```
mdb.models['PileJackingModel'].boundaryConditions['fixDownwardMovement'].deactivate(
   'PileJacking');
mdb.models['PileJackingModel'].VelocityBC(name='drivePile', createStepName='PileJacking',
   amplitude=UNSET, distributionType=UNIFORM, fieldName='', region=instPile.sets['setRP'],
   localCsys=None, v1=UNSET, v2=UNSET, v3=-1.0, vr1=UNSET, vr2=UNSET, vr3=UNSET);
```


Create Predefined Field

Name: fieldSoilMaterial

Step: Initial

Procedure:

Category: Mechanical, Fluid, [✓ Other]

Types for Selected Step: Temperature, [✓ Material assignment], Initial state, Saturation, Void ratio, Pore pressure, Fluid cavity pressure

Edit Predefined Field

Name: fieldSoilMaterial

Type: Material assignment

Step: Initial

Instance: instSoil

Volume Fractions

Definition: [✓ Uniform] Discrete fields

	Region	instSoil.instsand	Void
1	instSoil.setCellsBottom	1	0
2	instSoil.setCellsTop	0	1

Note: When the same region is specified more than once, the last entry is used.

Edit Predefined Field

Name: fieldStress

Type: Geostatic stress

Step: Initial

Region: instSoil.setCellsBottom

Stress magnitude 1: 0.0

Vertical coordinate 1: 4.0

Stress magnitude 2: -60.0

Vertical coordinate 2: 0.0

Lateral coefficient 1: 0.5

Lateral coefficent 2

[✓ Use lateral coefficient 1 value]

Specify:

Figure 6.3: Create Predefined Field window on the left hand side and Edit Predefined Field window for Material assignment in the middle. The Edit Predefined Field window for Geostatic stress is shown on the right hand side.

For the Eulerian material to be able to move through the mesh, a `Predefined Fields` entry has to be created in the initial step to assign the material for the soil. Either select the quick start symbol ( ), choose `Predefined Field->Create` in the main menu or right click on `Predefined Fields` in the initial step and choose `Create`. The Create Predefined Field window will open as shown in figure 6.3.

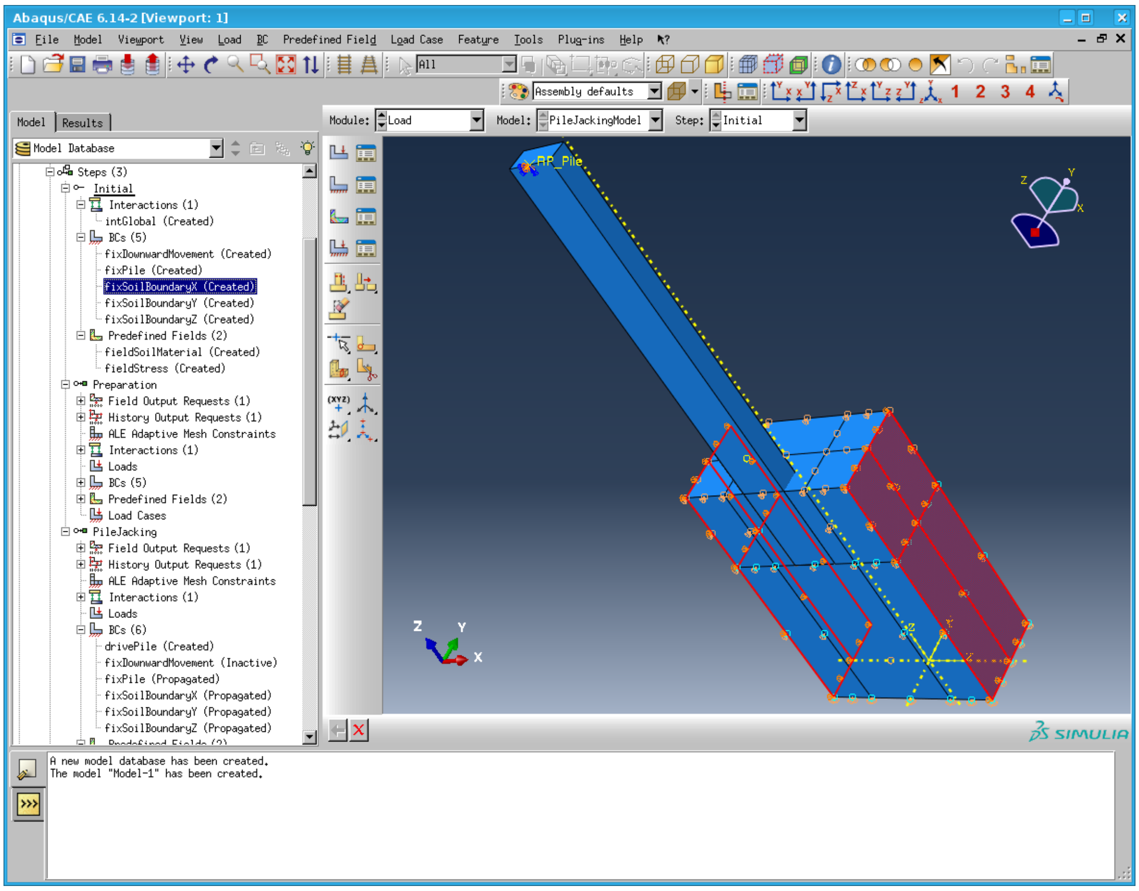
Choose Material assignment in the initial step and select the Eulerian instance. Assign `setCellsBottom` with soil material and `setCellsTop` with void to represent sand in the bottom part of the soil element and empty space above.

```
mdb.models['PileJackingModel'].MaterialAssignment(name='fieldSoilMaterial',
   instanceList=(instSoil, ), assignmentList=((instSoil.sets['setCellsBottom'], (1, )),
   (instSoil.sets['setCellsTop'], (0, ))), useFields=False);
```

Another predefined field can be used to model the geostatic stress within the sand material. This definition has to be applied in the initial step, too. Choose `Mechanical` as category to be able to select `Geostatic stress`. Choose the *setCellsBottom* set and configure the parameters to have 0 kPa at the upper plane of the sand material volume and 60 kPa at the lower boundary (choose −60 since it is pressure and not tension). The lateral coefficient can be set to $K_0 = 1 - \sin(\varphi) \approx 0.5$ (cf. right hand side of figure 6.3).

```
mdb.models['PileJackingModel'].GeostaticStress(lateralCoeff1=0.5, lateralCoeff2=None,
   name='fieldStress', region=instSoil.sets['setCellsBottom'], vCoord1=4.0, vCoord2=0.0,
   stressMag1=0.0, stressMag2=-60.0);
```

The movement of the sand at the soil boundaries has to be restricted, so that the material cannot flow out of the soil body. Since Eulerian material can move through the mesh, displacement boundary conditions aren't useful here. Therefore, three `Velocity/Angular velocity boundary` conditions are created and applied on the surfaces with the same normal direction. Sets of the boundary surfaces were created in chapter 2, so that e. g. a boundary condition restricting the movement in x-direction (*fixSoilBoundaryX*) can be applied on the corresponding set (*setFacesX*).


Abaqus/CAE 6.14-2 [Viewport: 1]
File Model Viewport View Load BC Predefined Field Load Case Feature Tools Plug-ins Help
Assembly defaults
Model Results
Model Database
Module: Load
Model: PileJackingModel
Step: Initial
RP_Pile
Z Y X
SIMULIA

```
Steps (3)
  Initial
    Interactions (1)
      intGlobal (Created)
    BCs (5)
      fixDownwardMovement (Created)
      fixPile (Created)
      fixSoilBoundaryX (Created)
      fixSoilBoundaryY (Created)
      fixSoilBoundaryZ (Created)
    Predefined Fields (2)
      fieldSoilMaterial (Created)
      fieldStress (Created)
  Preparation
    Field Output Requests (1)
    History Output Requests (1)
    ALE Adaptive Mesh Constraints
    Interactions (1)
    Loads
    BCs (5)
    Predefined Fields (2)
    Load Cases
  PileJacking
    Field Output Requests (1)
    History Output Requests (1)
    ALE Adaptive Mesh Constraints
    Interactions (1)
    Loads
    BCs (6)
      drivePile (Created)
      fixDownwardMovement (Inactive)
      fixPile (Propagated)
      fixSoilBoundaryX (Propagated)
      fixSoilBoundaryY (Propagated)
      fixSoilBoundaryZ (Propagated)
```

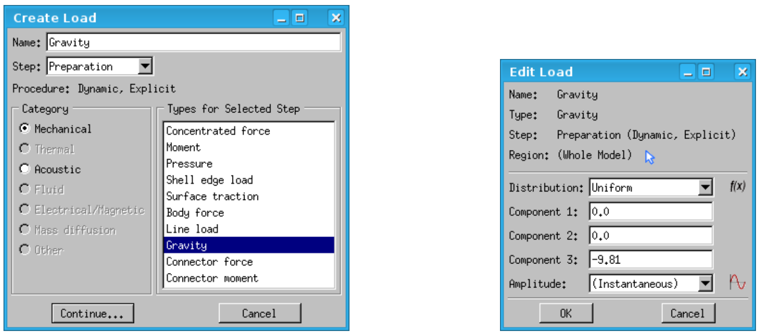
```
A new model database has been created.
The model "Model-1" has been created.
```

Figure 6.4: Abaqus window when creating soil surface boundary conditions. The surface region normal to the x-direction, where the *fixSoilBoundaryX* boundary condition is applied, is highlighted.

After applying boundary conditions for x-direction, y-direction and the bottom of the soil, no material should be able to flow in or out of the model except the top. The result can be seen in figure 6.4.

```python
mdb.models['PileJackingModel'].VelocityBC(name='fixSoilBoundaryX', createStepName='Initial',
   amplitude=UNSET, distributionType=UNIFORM, fieldName='', localCsys=None,
   region=instSoil.sets['setFacesX'],
   v1=0.0, v2=UNSET, v3=UNSET, vr1=UNSET, vr2=UNSET, vr3=UNSET);
mdb.models['PileJackingModel'].VelocityBC(name='fixSoilBoundaryY', createStepName='Initial',
   amplitude=UNSET, distributionType=UNIFORM, fieldName='', localCsys=None,
   region=instSoil.sets['setFacesY'],
   v1=UNSET, v2=0.0, v3=UNSET, vr1=UNSET, vr2=UNSET, vr3=UNSET);
mdb.models['PileJackingModel'].VelocityBC(name='fixSoilBoundaryZ', createStepName='Initial',
   amplitude=UNSET, distributionType=UNIFORM, fieldName='', localCsys=None,
   region=instSoil.sets['setFacesBottom'],
   v1=UNSET, v2=UNSET, v3=0.0, vr1=UNSET, vr2=UNSET, vr3=UNSET);
```

Similar to boundary conditions loads can be applied. Right click on `Loads` and choose `Create` in the model tree, select `Load->Create` in the main menu or click on its quick start symbol ( ). The Create Load window will open and like creating boundary conditions, the type of load, the region to apply to and the values need to be selected.



Create Load

Name: Gravity

Step: Preparation

Procedure: Dynamic, Explicit

Category: [✓ Mechanical] Thermal Acoustic Fluid Electrical/Magnetic Mass diffusion Other

Types for Selected Step: Concentrated force, Moment, Pressure, Shell edge load, Surface traction, Body force, Line load, [✓ Gravity], Connector force, Connector moment

Edit Load

Name: Gravity

Type: Gravity

Step: Preparation (Dynamic, Explicit)

Region: (Whole Model)

Distribution: Uniform

Component 1: 0.0

Component 2: 0.0

Component 3: -9.81

Amplitude: (Instantaneous)

Figure 6.5: The Create Load window on the left hand side and the Edit Load window for a gravitational force on the right hand side.

In this example, gravity will be applied to the whole model in the *Preparation step* by its gravitational acceleration 9.81 $\mathrm{m/s^2}$ in the (negative) z-direction (cf. figure 6.5).

```
mdb.models['PileJackingModel'].Gravity(comp3=-9.81, createStepName='Preparation',
    distributionType=UNIFORM, field='', name='Gravity');
```

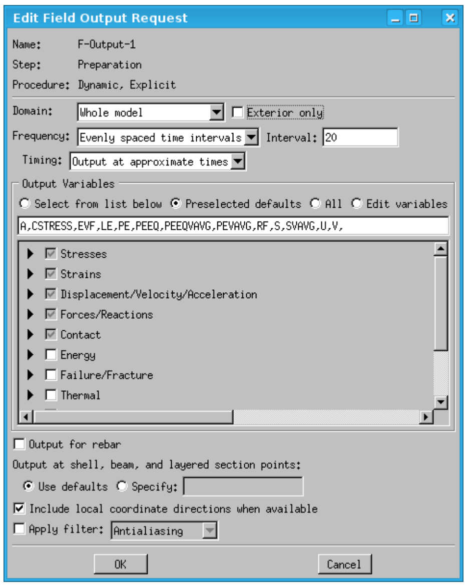
6.3 Output requests

Like boundary conditions, loads and predefined fields, output requests can be issued (in the `Step module`) either in each step or in general. A default Field Output Request and a default History Output Request will be present. Sometimes it may be useful to create new output requests, either by right clicking on the appropriate name in the model tree and choosing `Create`, by browsing `Output->` in the main menu and choosing `Create` or by using the quick start symbol for a new field output request ( ) or history output request ( ). As a rule of thumb, anything visible in the whole 3D-Model must be requested as Field Output and all further investigated time history/results of single points and sets should be requested as History Output.

hint » *A field output saves the requested states for certain point(s) in time for all elements. A history output saves the requested states for all time increments but only for the selected element(s).*

The present output requests can be edited by the output manager, which is accessible by the quick start symbol next to the output request creation ( ) by the model tree or the main menu (cf. figure 6.6).

In this example especially the flow of the material should be observed, which is saved in the *Eulerian Volume Fraction* (EVF). By default and among others, the *Volume-averaged Stresses* (SVAVG), the *Displacements* (U) and the *Accelerations* (A) for the soil and the *Displacements* (U) and *Reaction Forces* (RF) for the pile (i. e. its reference point) will be recorded. Other entries of the field output are of minor importance here and could be removed from the calculation.



Edit Field Output Request

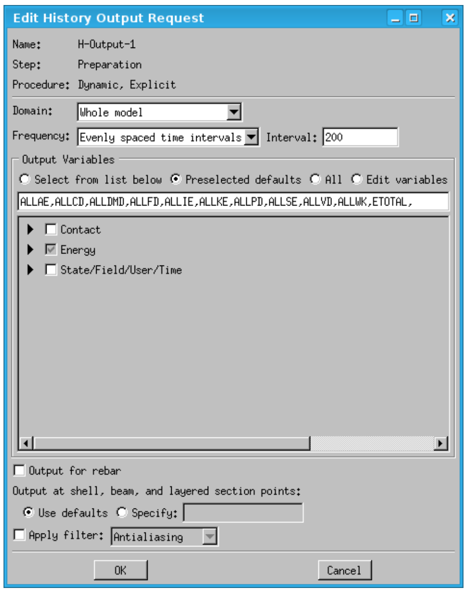
Name: F-Output-1
Step: Preparation
Procedure: Dynamic, Explicit

Domain: Whole model [ ] Exterior only
Frequency: Evenly spaced time intervals Interval: 20
Timing: Output at approximate times

Output Variables
Select from list below [✓ Preselected defaults] All Edit variables
A,CSTRESS,EVF,LE,PE,PEEQ,PEEQVAVG,PEVAVG,RF,S,SVAVG,U,V,

[✓] Stresses
[✓] Strains
[✓] Displacement/Velocity/Acceleration
[✓] Forces/Reactions
[✓] Contact
[ ] Energy
[ ] Failure/Fracture
[ ] Thermal

[ ] Output for rebar
Output at shell, beam, and layered section points:
[✓ Use defaults] Specify:
[✓] Include local coordinate directions when available
[ ] Apply filter: Antialiasing



Edit History Output Request

Name: H-Output-1
Step: Preparation
Procedure: Dynamic, Explicit

Domain: Whole model
Frequency: Evenly spaced time intervals Interval: 200

Output Variables
Select from list below [✓ Preselected defaults] All Edit variables
ALLAE,ALLCD,ALLDMD,ALLFD,ALLIE,ALLKE,ALLPD,ALLSE,ALLVD,ALLWK,ETOTAL,

[ ] Contact
[✓] Energy
[ ] State/Field/User/Time

[ ] Output for rebar
Output at shell, beam, and layered section points:
[✓ Use defaults] Specify:
[ ] Apply filter: Antialiasing

Figure 6.6: Default field output request and history output request window.

hint » *Consider which field outputs are necessary since they can generate big output databases very fast. Issuing more output requests at more nodes/elements and increasing the outputs per seconds can also increase the computation time.*

If only reference points or a few geometric objects should be observed, it is useful to create separate output requests for them. Choose the objects/sets as `Domain` instead of the whole model (see figure 6.6).

7 Meshing

Before creating and submitting the job the model has to be meshed. This could also be done much earlier (e. g. before assembling the parts), but has to be after all geometric changes are already applied.

hint » *Every time the geometry of a part is changed or partitions are added/removed, the mesh needs to be redone.*

There are two meshing viewports: Meshing can be done per part (dependent meshes) or for the assembled model (independent meshes). To mesh dependent meshes per part, first change to the `Part module` and then to the `Mesh module`. Otherwise Abaqus will try to mesh the whole assembled model, which—in this case—isn't desired.

Meshing is used to create nodes for calculating and evaluating the simulation. The meshing can be influenced by seeding the parts as a whole ( ) and edgewise ( ) to get the right amount of seeds per edge/part. Both options can also be accessed by the main menu or the model tree.

hint » *Most times it is favorable to have cube-like elements (structured hex) without any dominant direction.*

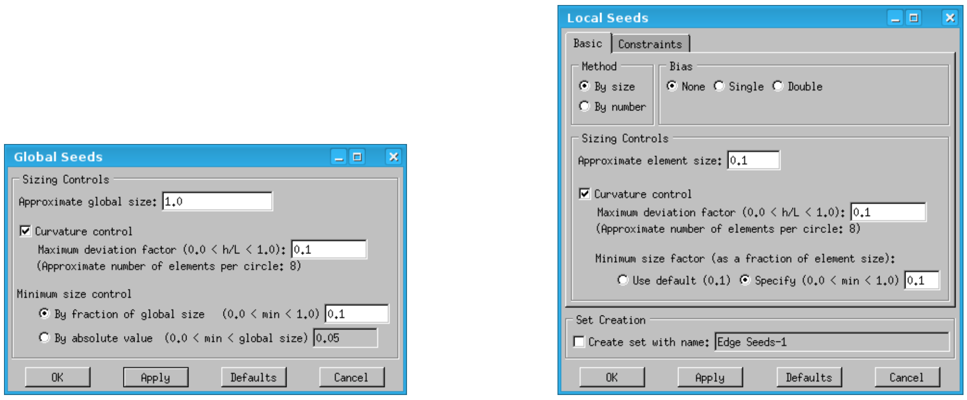
If this kind of mesh can be applied, the element will be colored green. Otherwise if only a sweeped mesh can be applied, the element will be yellow. But if a hex mesh is intended but can't be applied out of the box, partitions can help to divide the part in more suitable subparts. This can also be done to get a finer mesh locally. To change the mesh type (if possible), open the Assign Mesh Controls window ( ).

hint » *A coarser mesh has less nodes and needs less time to calculate. At the same time, the result might not be as precise as desired or not available at interesting points.*

First, the pile should be meshed. If not done already, switch to the `Mesh module` and activate the pile part. Seed the whole part and choose an approximate global size of 1.0 (cf. left hand side of figure 7.1). Apply and mesh the part by the quick start symbol ( ), the model tree or the main menu.

```
partPile.seedPart(deviationFactor=0.1, minSizeFactor=0.1, size=1.0);
partPile.generateMesh();
```

The meshing of the soil part will be a bit different. After making the soil element active, a global seed with an approximate global size of 0.4 will be set. Now the edges close to the pile jacking volume will be seeded ( ) as shown in the bottom of figure 7.2. Choose in



Global Seeds

Sizing Controls
Approximate global size: 1.0
[✓] Curvature control
Maximum deviation factor (0.0 < h/L < 1.0): 0.1
(Approximate number of elements per circle: 8)
Minimum size control
[✓ By fraction of global size (0.0 < min < 1.0) 0.1]
By absolute value (0.0 < min < global size) 0.05

OK Apply Defaults Cancel

Local Seeds

Basic | Constraints
Method: [✓ By size] By number
Bias: [✓ None] Single Double
Sizing Controls
Approximate element size: 0.1
[✓] Curvature control
Maximum deviation factor (0.0 < h/L < 1.0): 0.1
(Approximate number of elements per circle: 8)
Minimum size factor (as a fraction of element size):
Use default (0.1) [✓ Specify (0.0 < min < 1.0)] 0.1
Set Creation
[ ] Create set with name: Edge Seeds-1

OK Apply Defaults Cancel

Figure 7.1: Global Seeds window and Local Seeds window.

the Local Seeds window (cf. right hand side of figure 7.1) an approximate element size of 0.1.[1] Apply the mesh to the geometry to have a finer mesh region where the material flow is important and a coarser one everywhere else.

```
partSoil.seedPart(deviationFactor=0.1, minSizeFactor=0.1, size=0.4);
fineredges = partSoil.edges[0:0];
for singleedge in partSoil.edges:
   if ((singleedge.pointOn[0][1] <= 0.0) and (abs(singleedge.pointOn[0][0]) <= 1.0)):
      fineredges += partSoil.edges[singleedge.index:singleedge.index+1];
#
partSoil.seedEdgeBySize(constraint=FINER, deviationFactor=0.1, edges=fineredges,
   minSizeFactor=0.1, size=0.1);
partSoil.generateMesh();
```

[1]Sometimes it may be useful to define the amount of seeds with the by number method.

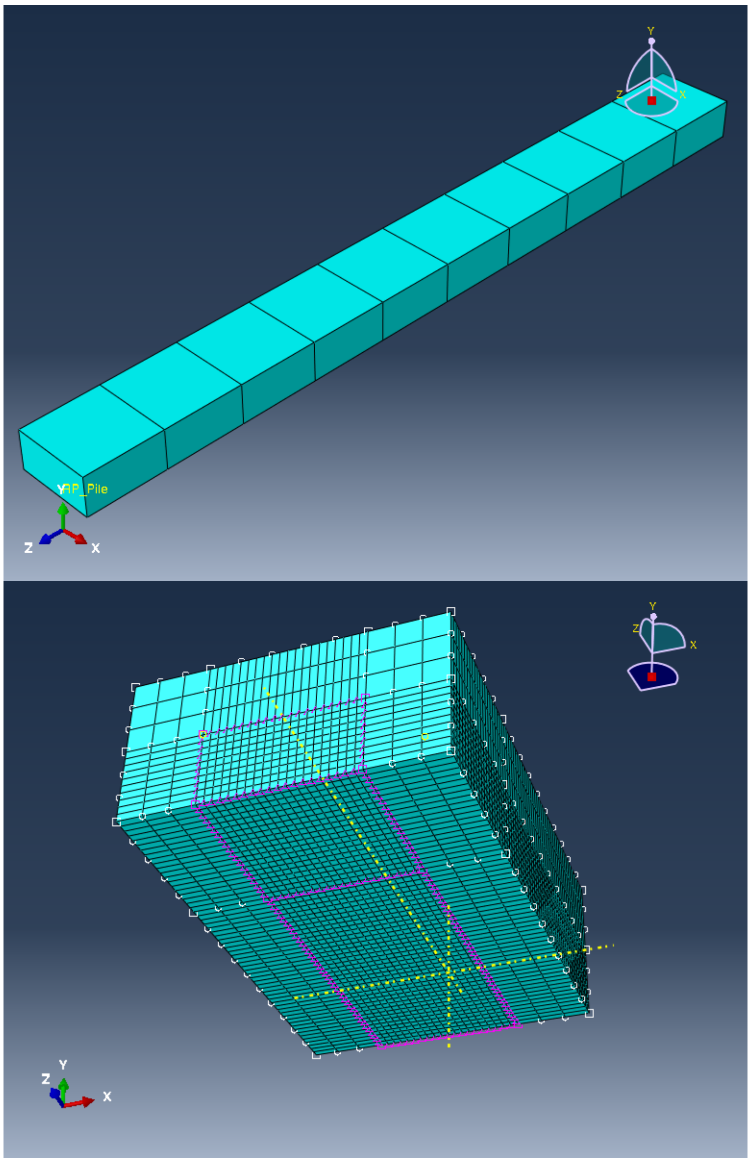


Figure 7.2: Viewport of meshed pile (above) and meshed soil element (below). The viewport of the soil element also shows the global seeds (white circles) and the manually set edge seeds (purple triangles lines).

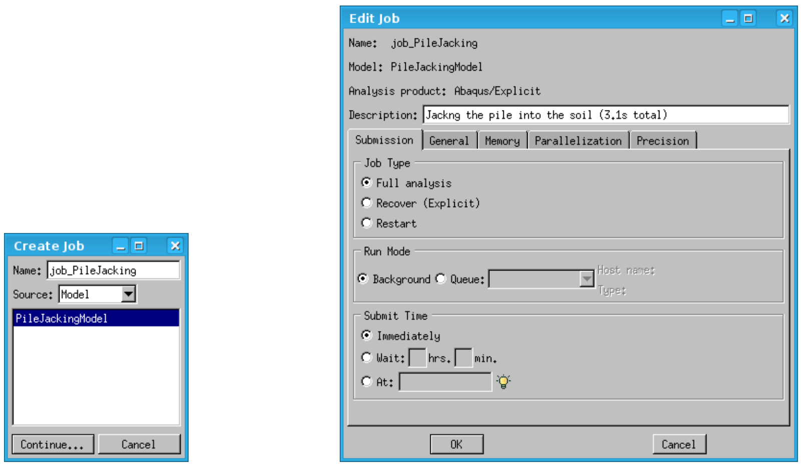
8 Getting the job done

8.1 Job creation and submission

All preparations are done and a new job can be created in the `Job module`. Either right click on `Jobs` and choose `Create`, select `Job->Create` in the main menu or use the quick start symbol ( ). The Create Job window will open as shown in figure 8.1 and the new job has to be named (this will be the name of the input file). Give a short description in the Edit Job window and for longer/critical calculations, always change the precision in the `Precision` tab to Double Precision. If using a user routine (e. g. user defined material), the file needs to be added in the `General` tab.

Right click on the newly created job and choose `Write Input` to create an input file. After creating a job a stand-alone text file with all configuration is written—the input file `[model].inp`. Abaqus can run this file to do the simulation.

```
mdb.Job(activateLoadBalancing=False, atTime=None, contactPrint=OFF,
   description='Jacking the pile into the soil (3.1s total)', echoPrint=OFF,
   explicitPrecision=DOUBLE_PLUS_PACK, historyPrint=OFF, memory=90, memoryUnits=PERCENTAGE,
   model='PileJackingModel', modelPrint=OFF, multiprocessingMode=DEFAULT,
   name='job_PileJacking', nodalOutputPrecision=FULL, numCpus=1, numDomains=1,
   parallelizationMethodExplicit=DOMAIN, queue=None, resultsFormat=ODB, scratch='',
   type=ANALYSIS, userSubroutine='', waitHours=0, waitMinutes=0);
mdb.jobs['job_PileJacking'].writeInput(consistencyChecking=OFF);
```


Create Job

Name: job_PileJacking

Source: Model

PileJackingModel

Edit Job

Name: job_PileJacking

Model: PileJackingModel

Analysis product: Abaqus/Explicit

Description: Jackng the pile into the soil (3.1s total)

Job Type: [✓ Full analysis] Recover (Explicit) Restart

Run Mode: [✓ Background] Queue:

Submit Time: [✓ Immediately] Wait: hrs. min. At:

Figure 8.1: Create Job window (left hand side) and Edit Job window (right hand side).

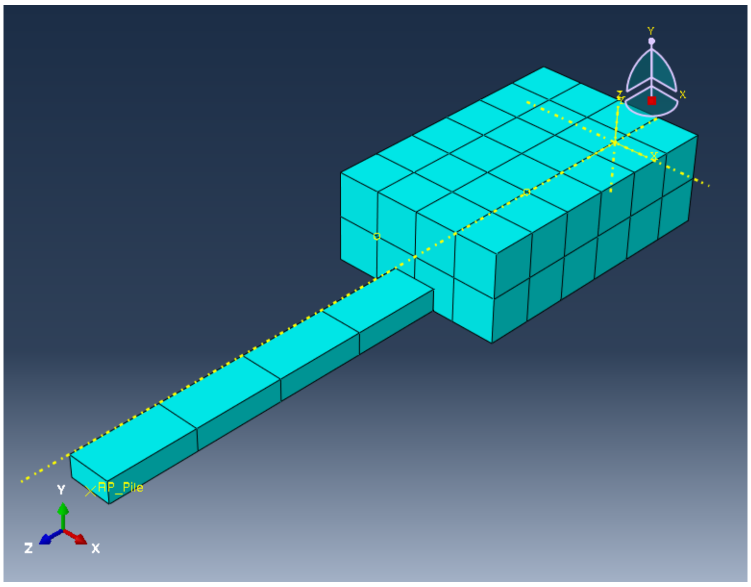


Figure 8.2: Same model with a very coarse mesh—for discussion of the input file only.

After creating a job and before submission, it is advisable to run a data check. Either select `Job->Data Check->` in the main menu or right click on the job and choose `Data Check`. Most formal errors in the model can be found and will be reported. A successfull data check is a prerequisite for running the simulation.

hint » *Always run a data check before submission to exclude most formal errors before starting the simulation.*

Data checks and simulations always create additional status files, especially `[model].dat` and `[model].sta`. Both text files can be used to gather information about the calculation as well as warnings and error messages. They can also be accessed by right clicking on the job name and choosing `Monitor`. If problems are encountered, try to correct them and rerun the data check/simulation.

Small jobs can be run in the Abaqus CAE gui while bigger jobs are preferably outsourced to a computation cluster (if available).

8.2 Concerning Input files

This section will show and discuss the input file created by this tutorial. The only difference is that a much coarser mesh is used for the soil body than described in chapter 7 to prevent an unnecessary data overhead for the discussion. The mesh applied to the pile has an approximate global size of 2, the mesh for the soil consists only of the global seeds with a size of 1 resulting in a model shown in figure 8.2. The input file looks like this:

The Heading is named after the job description and line 2 includes the original name of this job file (`[model].inp`) and the model name. Starting in line 8, actions within the `Part module` are shown.

```
 1  *Heading
 2   Jacking the pile into the soil (3s total)
 3  ** Job name: job_PileJacking Model name: PileJackingModel
 4  ** Generated by: Abaqus/CAE 6.14-1
 5  *Preprint, echo=NO, model=NO, history=NO, contact=NO
 6  **
 7  ** PARTS
 8  **
```

Giving meaningful names to elements does not only look nice, it helps to find and understand elements and structures. In line 9 to 34 the coordinates of the 24 nodes of the *Pile* are numbered and described, which are defined by the (coarse) mesh.

```
 9  *Part, name=Pile
10  *Node
11        1,          0.5,        0.25,         10.
12        2,          0.5,       -0.25,         10.
13        3,          0.5,        0.25,          8.
14        4,          0.5,       -0.25,          8.
15        5,          0.5,        0.25,          6.
16        6,          0.5,       -0.25,          6.
17        7,          0.5,        0.25,          4.
18        8,          0.5,       -0.25,          4.
19        9,          0.5,        0.25,          2.
20       10,          0.5,       -0.25,          2.
21       11,          0.5,        0.25,          0.
22       12,          0.5,       -0.25,          0.
23       13,         -0.5,        0.25,         10.
24       14,         -0.5,       -0.25,         10.
25       15,         -0.5,        0.25,          8.
26       16,         -0.5,       -0.25,          8.
27       17,         -0.5,        0.25,          6.
28       18,         -0.5,       -0.25,          6.
29       19,         -0.5,        0.25,          4.
30       20,         -0.5,       -0.25,          4.
31       21,         -0.5,        0.25,          2.
32       22,         -0.5,       -0.25,          2.
33       23,         -0.5,        0.25,          0.
34       24,         -0.5,       -0.25,          0.
```

The element type is a short version of **C**ontinuous **3D** element with **8** nodes and **R**educed integration.

```
35  *Element, type=C3D8R
36  1, 13, 14, 16, 15,  1,  2,  4,  3
37  2, 15, 16, 18, 17,  3,  4,  6,  5
38  3, 17, 18, 20, 19,  5,  6,  8,  7
39  4, 19, 20, 22, 21,  7,  8, 10,  9
40  5, 21, 22, 24, 23,  9, 10, 12, 11
```

Line 41 marks the 25th node (`0, -0.25, 10`). It is used as internal reference point in 42–43 and as a set in 49–50. The set of the whole part is defined in line 45–48.

```
41  *Node
42       25,           0.,       -0.25,         10.
43  *Nset, nset=Pile-RefPt_, internal
44  25,
45  *Nset, nset=setAll, generate
46    1,  24,   1
47  *Elset, elset=setAll, generate
48   1,  5,  1
49  *Nset, nset=setRP
50   25,
```

The section *secPile* is created to assign steel material to the element. The definition of the *Pile* part is finished in line 54.

```
51  ** Section: secPile
52  *Solid Section, elset=setAll, material=steel
53  ,
54  *End Part
55  **
```

The definition for the *Soil* part starts here. It is essentially the same as for the *Pile* part above. Line 58–162 list and describe the coordinates of the 105 nodes defined by the coarse mesh.

```
 56  *Part, name=Soil
 57  *Node
 58        1,           2.,           0.,           4.
 59        2,           2.,           0.,           0.
 60        3,           2.,           1.,           0.
 61        4,           2.,           1.,           4.
 62        5,           1.,           0.,           4.
 63        6,           1.,           0.,           0.
 64        7,           1.,           1.,           0.
 65        8,           1.,           1.,           4.
 66        9,          -1.,           0.,           0.
 67       10,          -1.,           1.,           0.
 68       11,          -1.,           0.,           4.
 69       12,          -1.,           1.,           4.
 70       13,          -2.,           0.,           6.
 71       14,          -1.,           0.,           6.
 72       15,          -1.,           1.,           6.
 73       16,          -2.,           1.,           6.
 74       17,          -2.,           0.,           4.
 75       18,          -2.,           1.,           4.
 76       19,           2.,           0.,           6.
 77       20,           2.,           1.,           6.
 78       21,           1.,           0.,           6.
 79       22,           1.,           1.,           6.
 80       23,          -2.,           1.,           0.
 81       24,          -2.,           0.,           0.
 82       25,          -1.,          -1.,           0.
 83       26,          -1.,          -1.,           4.
 84       27,          -2.,          -1.,           4.
 85       28,          -2.,          -1.,           0.
 86       29,          -1.,          -1.,           6.
 87       30,           1.,          -1.,           4.
 88       31,           1.,          -1.,           6.
 89       32,           2.,          -1.,           6.
 90       33,           2.,          -1.,           4.
 91       34,          -2.,          -1.,           6.
 92       35,           1.,          -1.,           0.
 93       36,           2.,          -1.,           0.
 94       37,           2.,           0.,           3.
 95       38,           2.,           0.,           2.
 96       39,           2.,           0.,           1.
 97       40,           2.,           1.,           1.
 98       41,           2.,           1.,           2.
 99       42,           2.,           1.,           3.
100       43,           1.,           0.,           3.
101       44,           1.,           0.,           2.
102       45,           1.,           0.,           1.
103       46,           1.,           1.,           1.
104       47,           1.,           1.,           2.
105       48,           1.,           1.,           3.
106       49,           0.,           0.,           0.
107       50,           0.,           1.,           0.
108       51,           0.,           0.,           4.
109       52,           0.,           1.,           4.
110       53,          -1.,           1.,           1.
111       54,          -1.,           1.,           2.
112       55,          -1.,           1.,           3.
113       56,          -1.,           0.,           3.
```

Soil coordinates continued

```
     57,          -1.,           0.,            2.
     58,          -1.,           0.,            1.
     59,          -2.,           1.,            5.
     60,          -2.,           0.,            5.
     61,          -1.,           1.,            5.
     62,          -1.,           0.,            5.
     63,           2.,           0.,            5.
     64,           2.,           1.,            5.
     65,           1.,           0.,            5.
     66,           1.,           1.,            5.
     67,           0.,           0.,            6.
     68,           0.,           1.,            6.
     69,          -2.,           1.,            3.
     70,          -2.,           1.,            2.
     71,          -2.,           1.,            1.
     72,          -2.,           0.,            1.
     73,          -2.,           0.,            2.
     74,          -2.,           0.,            3.
     75,          -1.,          -1.,            1.
     76,          -1.,          -1.,            2.
     77,          -1.,          -1.,            3.
     78,          -2.,          -1.,            3.
     79,          -2.,          -1.,            2.
     80,          -2.,          -1.,            1.
     81,          -1.,          -1.,            5.
     82,           0.,          -1.,            4.
     83,           1.,          -1.,            5.
     84,           0.,          -1.,            6.
     85,           2.,          -1.,            5.
     86,          -2.,          -1.,            5.
     87,           0.,          -1.,            0.
     88,           1.,          -1.,            1.
     89,           1.,          -1.,            2.
     90,           1.,          -1.,            3.
     91,           2.,          -1.,            3.
     92,           2.,          -1.,            2.
     93,           2.,          -1.,            1.
     94,           0.,           0.,            1.
     95,           0.,           0.,            2.
     96,           0.,           0.,            3.
     97,           0.,           1.,            1.
     98,           0.,           1.,            2.
     99,           0.,           1.,            3.
    100,           0.,           0.,            5.
    101,           0.,           1.,            5.
    102,           0.,          -1.,            5.
    103,           0.,          -1.,            3.
    104,           0.,          -1.,            2.
    105,           0.,          -1.,            1.
```

The element type of the *Soil* part is a short version of **E**ulerian **C**ontinuous **3D** element with **8** nodes and **R**educed integration.

```
*Element, type=EC3D8R
 1,  5, 43, 48,  8,  1, 37, 42,  4
 2, 43, 44, 47, 48, 37, 38, 41, 42
 3, 44, 45, 46, 47, 38, 39, 40, 41
 4, 45,  6,  7, 46, 39,  2,  3, 40
 5, 45, 94, 97, 46,  6, 49, 50,  7
 6, 94, 58, 53, 97, 49,  9, 10, 50
 7, 44, 95, 98, 47, 45, 94, 97, 46
 8, 95, 57, 54, 98, 94, 58, 53, 97
```

Soil elements continued

```
172  9, 43, 96, 99, 48, 44, 95, 98, 47
173 10, 96, 56, 55, 99, 95, 57, 54, 98
174 11,  5, 51, 52,  8, 43, 96, 99, 48
175 12, 51, 11, 12, 52, 96, 56, 55, 99
176 13, 60, 62, 61, 59, 13, 14, 15, 16
177 14, 17, 11, 12, 18, 60, 62, 61, 59
178 15, 21, 65, 66, 22, 19, 63, 64, 20
179 16, 65,  5,  8, 66, 63,  1,  4, 64
180 17,  11,  51,  52,  12,  62, 100, 101,  61
181 18,  51,   5,   8,  52, 100,  65,  66, 101
182 19,  62, 100, 101,  61,  14,  67,  68,  15
183 20, 100,  65,  66, 101,  67,  21,  22,  68
184 21, 74, 56, 55, 69, 17, 11, 12, 18
185 22, 73, 57, 54, 70, 74, 56, 55, 69
186 23, 72, 58, 53, 71, 73, 57, 54, 70
187 24, 24,  9, 10, 23, 72, 58, 53, 71
188 25,  9, 58, 72, 24, 25, 75, 80, 28
189 26, 58, 57, 73, 72, 75, 76, 79, 80
190 27, 57, 56, 74, 73, 76, 77, 78, 79
191 28, 56, 11, 17, 74, 77, 26, 27, 78
192 29,  14,  62, 100,  67,  29,  81, 102,  84
193 30,  62,  11,  51, 100,  81,  26,  82, 102
194 31,  67, 100,  65,  21,  84, 102,  83,  31
195 32, 100,  51,   5,  65, 102,  82,  30,  83
196 33,  5, 65, 83, 30,  1, 63, 85, 33
197 34, 65, 21, 31, 83, 63, 19, 32, 85
198 35, 11, 62, 60, 17, 26, 81, 86, 27
199 36, 62, 14, 13, 60, 81, 29, 34, 86
200 37,  11,  56,  96,  51,  26,  77, 103,  82
201 38,  56,  57,  95,  96,  77,  76, 104, 103
202 39,  57,  58,  94,  95,  76,  75, 105, 104
203 40,  58,   9,  49,  94,  75,  25,  87, 105
204 41,  51,  96,  43,   5,  82, 103,  90,  30
205 42,  96,  95,  44,  43, 103, 104,  89,  90
206 43,  95,  94,  45,  44, 104, 105,  88,  89
207 44,  94,  49,   6,  45, 105,  87,  35,  88
208 45,  6, 45, 88, 35,  2, 39, 93, 36
209 46, 45, 44, 89, 88, 39, 38, 92, 93
210 47, 44, 43, 90, 89, 38, 37, 91, 92
211 48, 43,  5, 30, 90, 37,  1, 33, 91
```

The set for the whole part is defined in line 212–215, *setCellsTop* in 216–221 and *setCellsBottom* in 222–230.

```
212 *Nset, nset=setAll, generate
213    1,  105,    1
214 *Elset, elset=setAll, generate
215   1,  48,   1
216 *Nset, nset=setCellsTop
217    1,  4,  5,  8, 11, 12, 13, 14, 15, 16,  17,  18,  19, 20, 21, 22
218   26, 27, 29, 30, 31, 32, 33, 34, 51, 52,  59,  60,  61, 62, 63, 64
219   65, 66, 67, 68, 81, 82, 83, 84, 85, 86, 100, 101, 102
220 *Elset, elset=setCellsTop
221  13, 14, 15, 16, 17, 18, 19, 20, 29, 30, 31, 32, 33, 34, 35, 36
222 *Nset, nset=setCellsBottom
223    1,  2,  3,  4,  5,  6,  7,  8,   9,  10,  11, 12, 17, 18, 23, 24
224   25, 26, 27, 28, 30, 33, 35, 36,  37,  38,  39, 40, 41, 42, 43, 44
225   45, 46, 47, 48, 49, 50, 51, 52,  53,  54,  55, 56, 57, 58, 69, 70
226   71, 72, 73, 74, 75, 76, 77, 78,  79,  80,  82, 87, 88, 89, 90, 91
227   92, 93, 94, 95, 96, 97, 98, 99, 103, 104, 105
228 *Elset, elset=setCellsBottom
229   1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 21, 22, 23, 24
230  25, 26, 27, 28, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48
```

The sets created to apply the boundary conditions on the soil are defined/saved here.

```
231 *Nset, nset=setFacesX
232   1,  2,  3,  4, 13, 16, 17, 18, 19, 20, 23, 24, 27, 28, 32, 33
233  34, 36, 37, 38, 39, 40, 41, 42, 59, 60, 63, 64, 69, 70, 71, 72
234  73, 74, 78, 79, 80, 85, 86, 91, 92, 93
235 *Elset, elset=setFacesX
236   1,  2,  3,  4, 13, 14, 15, 16, 21, 22, 23, 24, 25, 26, 27, 28
237  33, 34, 35, 36, 45, 46, 47, 48
238 *Nset, nset=setFacesY
239    3,   4,   7,   8,  10,  12, 15, 16, 18, 20, 22, 23, 25, 26, 27, 28
240   29,  30,  31,  32,  33,  34, 35, 36, 40, 41, 42, 46, 47, 48, 50, 52
241   53,  54,  55,  59,  61,  64, 66, 68, 69, 70, 71, 75, 76, 77, 78, 79
242   80,  81,  82,  83,  84,  85, 86, 87, 88, 89, 90, 91, 92, 93, 97, 98
243   99, 101, 102, 103, 104, 105
244 *Elset, elset=setFacesY, generate
245   1,  48,   1
246 *Nset, nset=setFacesBottom
247   2,  3,  6,  7,  9, 10, 23, 24, 25, 28, 35, 36, 49, 50, 87
248 *Elset, elset=setFacesBottom
249   4,  5,  6, 24, 25, 40, 44, 45
```

The section *secSoil* is created to assign the Eulerian material sand to the element. The definition of this part is done in line 255.

```
250 ** Section: secSoil
251 *Eulerian Section, elset=setAll
252 sand, instsand
253 *Surface, type=EULERIAN MATERIAL, name=instsand
254 instsand
255 *End Part
256 **
257 **
```

The assembly of the whole model starts here. The parts are instanced as *instPile* and *instSoil*.

```
258 ** ASSEMBLY
259 **
260 *Assembly, name=Assembly
261 **
262 *Instance, name=instPile, part=Pile
263          0.,       -0.75,          4.
264 *End Instance
265 **
266 *Instance, name=instSoil, part=Soil
267 *End Instance
268 **
```

The constraint between reference point on top of the pile and the pile itself is defined in line 269–270.

```
269 ** Constraint: RigidRPSteelpile
270 *Rigid Body, ref node=instPile.setRP, elset=instPile.setAll
271 *End Assembly
272 **
273 ** MATERIALS
274 **
```

The sand and the steel material are defined below. Both have a density and two elastic behavior values.

```
275 *Material, name=sand
276 *Density
277  1.6,
278 *Elastic
279 30000., 0.25
280 *Material, name=steel
281 *Density
282  7.87,
283 *Elastic
284  2.1e+08, 0.3
285 **
```

The interaction properties in normal and tangential direction for the general contact definition (see line 326–330) are defined here.

```
286 ** INTERACTION PROPERTIES
287 **
288 *Surface Interaction, name=intContact
289 *Friction
290 0.,
291 *Surface Behavior, pressure-overclosure=HARD
292 **
```

All boundary conditions in the initial step are located here. For each one, the name and type are defined, the instance, set and the restricted coordinate is selected.

```
293 ** BOUNDARY CONDITIONS
294 **
295 ** Name: fixDownwardMovement Type: Displacement/Rotation
296 *Boundary
297 instPile.setRP, 3, 3
298 ** Name: fixPile Type: Displacement/Rotation
299 *Boundary
300 instPile.setRP, 1, 1
301 instPile.setRP, 2, 2
302 instPile.setRP, 4, 4
303 instPile.setRP, 5, 5
304 instPile.setRP, 6, 6
305 ** Name: fixSoilBoundaryX Type: Velocity/Angular velocity
306 *Boundary, type=VELOCITY
307 instSoil.setFacesX, 1, 1
308 ** Name: fixSoilBoundaryY Type: Velocity/Angular velocity
309 *Boundary, type=VELOCITY
310 instSoil.setFacesY, 2, 2
311 ** Name: fixSoilBoundaryZ Type: Velocity/Angular velocity
312 *Boundary, type=VELOCITY
313 instSoil.setFacesBottom, 3, 3
314 **
```

The material assignment for Eulerian elements is done by predefined fields. In line 319, the sand material (*instsand*) is assigned to the *setCellsBottom* set, while unassigned geometry will contain void material by default. Geostatic stress is applied to this set in line 321–322.

```
315 ** PREDEFINED FIELDS
316 **
317 ** Name: fieldSoilMaterial   Type: Material assignment
318 *Initial Conditions, type=VOLUME FRACTION
319 instSoil.setCellsBottom, instSoil.instsand, 1.
320 ** Name: fieldStress   Type: Geostatic stress
321 *Initial Conditions, type=STRESS, GEOSTATIC
322 instSoil.setCellsBottom, 0., 4., -60., 0., 0.5, 0.5
323 **
324 ** INTERACTIONS
325 **
```

The contact interaction described here is linked to the *intContact* interaction property defined earlier.

```
326 ** Interaction: intGlobal
327 *Contact, op=NEW
328 *Contact Inclusions, ALL EXTERIOR
329 *Contact Property Assignment
330  ,  , intContact
331 ** ----------------------------------------------------------------
332 **
```

All previous definitions describe the initial setup. After line 333 the first created step *Preparation* starts. Note the `Dynamic, Explicit` analysis demanded in line 337.

```
333 ** STEP: Preparation
334 **
335 *Step, name=Preparation, nlgeom=YES
336 Preparation step
337 *Dynamic, Explicit
338 , 0.1
339 *Bulk Viscosity
340 0.06, 1.2
341 **
```

Each user created step can have its own loads, boundary conditions and output requests and only changes to the previous step are listed.

```
342 ** LOADS
343 **
344 ** Name: Gravity   Type: Gravity
345 *Dload
346 , GRAV, 9.81, 0., 0., -1.
347 **
348 ** OUTPUT REQUESTS
349 **
350 *Restart, write, number interval=1, time marks=NO
351 **
```

In each step both output requests can be issued. No changes were made in this example, so the preselected variables are dealt with in field output and history output (line 354 and 358). This step ends at line 359.

```
352 ** FIELD OUTPUT: F-Output-1
353 **
354 *Output, field, variable=PRESELECT
355 **
356 ** HISTORY OUTPUT: H-Output-1
357 **
358 *Output, history, variable=PRESELECT
359 *End Step
360 ** ----------------------------------------------------------------
361 **
```

Definitions of the *PileJacking* step.

```
362 ** STEP: PileJacking
363 **
364 *Step, name=PileJacking, nlgeom=YES
365 Jacking the pile into the soil
366 *Dynamic, Explicit
367 , 3.
368 *Bulk Viscosity
369 0.06, 1.2
370 **
371 ** BOUNDARY CONDITIONS
372 **
```

From line 373 on, the boundary conditions are adapted to drive the pile into the soil.

```
373 ** Name: drivePile Type: Velocity/Angular velocity
374 *Boundary, op=NEW, type=VELOCITY
375 instPile.setRP, 3, 3, -1.
376 ** Name: fixDownwardMovement Type: Displacement/Rotation
377 *Boundary, op=NEW
378 ** Name: fixPile Type: Displacement/Rotation
379 *Boundary, op=NEW
380 instPile.setRP, 1, 1
381 instPile.setRP, 2, 2
382 instPile.setRP, 4, 4
383 instPile.setRP, 5, 5
384 instPile.setRP, 6, 6
385 ** Name: fixSoilBoundaryX Type: Velocity/Angular velocity
386 *Boundary, op=NEW, type=VELOCITY
387 instSoil.setFacesX, 1, 1
388 ** Name: fixSoilBoundaryY Type: Velocity/Angular velocity
389 *Boundary, op=NEW, type=VELOCITY
390 instSoil.setFacesY, 2, 2
391 ** Name: fixSoilBoundaryZ Type: Velocity/Angular velocity
392 *Boundary, op=NEW, type=VELOCITY
393 instSoil.setFacesBottom, 3, 3
394 **
395 ** OUTPUT REQUESTS
396 **
397 *Restart, write, number interval=1, time marks=NO
398 **
```

Output requests of *PileJacking* step continued

```
** FIELD OUTPUT: F-Output-1
**
*Output, field, variable=PRESELECT
**
** HISTORY OUTPUT: H-Output-1
**
*Output, history, variable=PRESELECT
*End Step
```

9 Visualize the results

The partial or complete results of a simulation can be analyzed and visualized in the `Visualization module`. The results are stored in the output database file `[model].odb`. An arbitrary model output database can be opened by selecting `File->Open`.[1]

If the output database is in the same directory as the opened model file `[model].cae` or if the calculation was small enough and has been run in the graphical interface, it can directly be accessed by right clicking on `Jobs` and choosing `Results`. Choosing `Visualization` in the module drop-down menu above the viewport loads the last used output data base.

If the simulation didn't successfully finish, check the `[model].sta` and `[model].dat` files and try to correct all errors.

9.1 Field output

Figure 9.1 shows the deformed model after simulation and with some graphical adaptions described below. Either select `Plot->Contours->On Deformed Shape` in the main menu, or use its quick start symbol ( ) to view the deformed shape of the model.

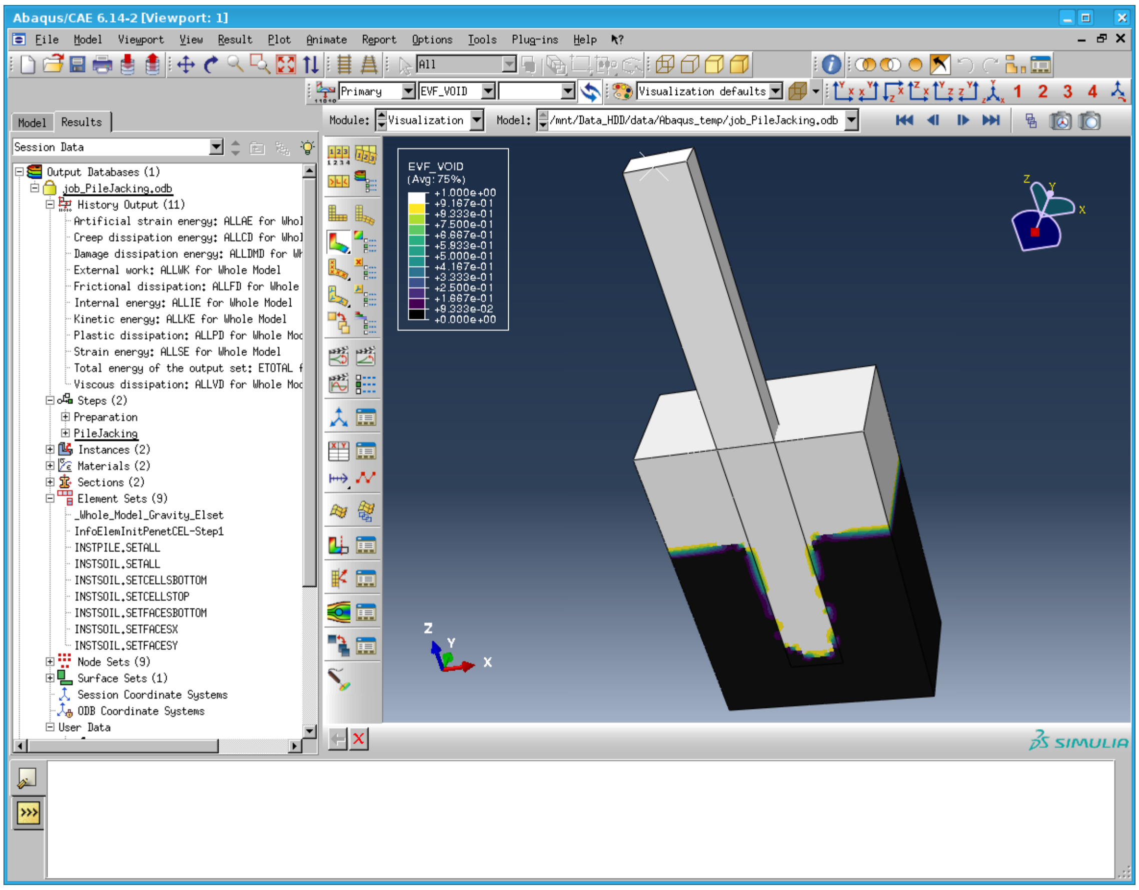
The variables selected in field/history output requests can be visualized or plotted here. To view a certain field output variable, either select `Result->Field Output` in the main menu or the Field Output Dialog symbol ( ) beneath the main menu. Next to this symbol, the output variable can also be selected by three drop-down menus.

Graphical options like which lines are shown and which color spectrum is used can be changed within the Contour Plot Options window and the Common Plot Options window (cf. figure 9.2). To open the Contour Plot Options window, select `Options->Contour` in the main menu or use its quick start symbol ( ). The Common Plot Options window can be accessed by `Options->Common` in the main menu.

To show or hide graphical objects like the compass in the viewport, open the Viewport Annotation Options window by selecting `Viewport->Viewport Annotation Options` in the main menu.

hint » *Images of the model which are to be inserted in a thesis or paper should be saved as Encapsulated PostScript (`.eps`) files, to minimize losing image quality when scaling.*

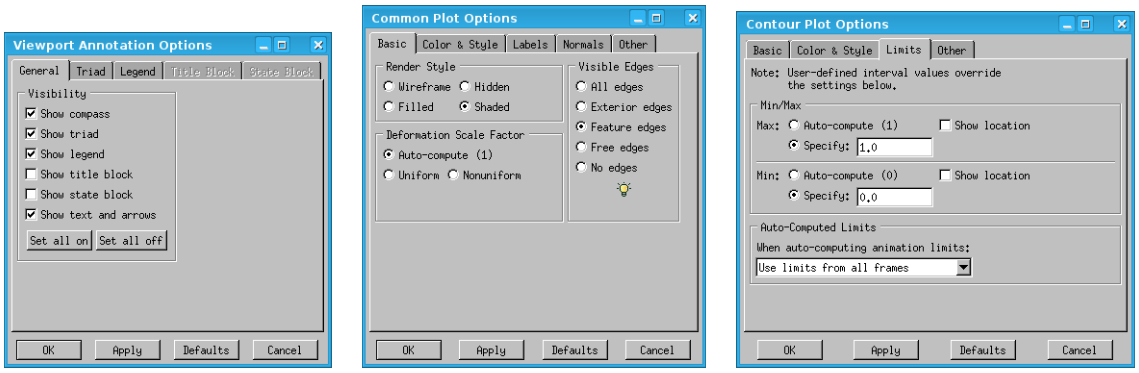
[1]If the output database was created with an older version of Abaqus, it has to be upgraded first. If it was created with a more recent version, it can't be opened.


Abaqus/CAE 6.14-2 [Viewport: 1]
File Model Viewport View Result Plot Animate Report Options Tools Plug-ins Help
Model Results
Session Data
Module: Visualization
Model: /mnt/Data_HDD/data/Abaqus_temp/job_PileJacking.odb
Primary EVF_VOID
Visualization defaults
SIMULIA

Output Databases (1)
job_PileJacking.odb
History Output (11)
Artificial strain energy: ALLAE for Whol
Creep dissipation energy: ALLCD for Whol
Damage dissipation energy: ALLDMD for Wh
External work: ALLWK for Whole Model
Frictional dissipation: ALLFD for Whole
Internal energy: ALLIE for Whole Model
Kinetic energy: ALLKE for Whole Model
Plastic dissipation: ALLPD for Whole Moc
Strain energy: ALLSE for Whole Model
Total energy of the output set: ETOTAL f
Viscous dissipation: ALLVD for Whole Moc
Steps (2)
Preparation
PileJacking
Instances (2)
Materials (2)
Sections (2)
Element Sets (9)
_Whole_Model_Gravity_Elset
InfoElemInitPenetCEL-Step1
INSTPILE.SETALL
INSTSOIL.SETALL
INSTSOIL.SETCELLSBOTTOM
INSTSOIL.SETCELLSTOP
INSTSOIL.SETFACESBOTTOM
INSTSOIL.SETFACESX
INSTSOIL.SETFACESY
Node Sets (9)
Surface Sets (1)
Session Coordinate Systems
ODB Coordinate Systems
User Data

EVF_VOID
(Avg: 75%)
+1.000e+00
+9.167e-01
+8.333e-01
+7.500e-01
+6.667e-01
+5.833e-01
+5.000e-01
+4.167e-01
+3.333e-01
+2.500e-01
+1.667e-01
+8.333e-02
+0.000e+00

Figure 9.1: View of the deformed model after simulation.

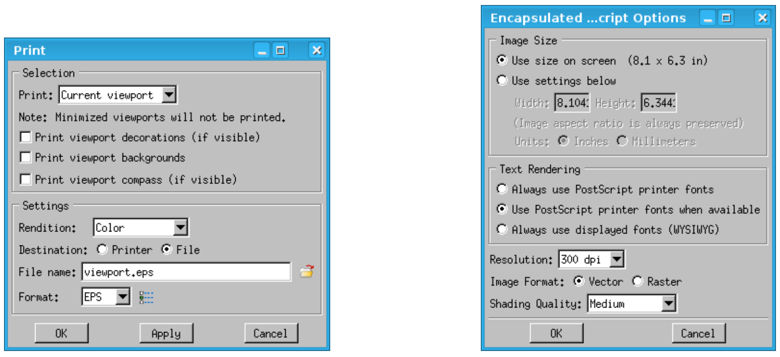


Viewport Annotation Options
General | Triad | Legend | Title Block | State Block
Visibility
☑ Show compass
☑ Show triad
☑ Show legend
☐ Show title block
☐ Show state block
☑ Show text and arrows
Set all on | Set all off
OK Apply Defaults Cancel

Common Plot Options
Basic | Color & Style | Labels | Normals | Other
Render Style
Wireframe Hidden
Filled [✓ Shaded]
Deformation Scale Factor
[✓ Auto-compute (1)]
Uniform Nonuniform
Visible Edges
All edges
Exterior edges
[✓ Feature edges]
Free edges
No edges
OK Apply Defaults Cancel

Contour Plot Options
Basic | Color & Style | Limits | Other
Note: User-defined interval values override
the settings below.
Min/Max
Max: Auto-compute (1) ☐ Show location
[✓ Specify:] 1.0
Min: Auto-compute (0) ☐ Show location
[✓ Specify:] 0.0
Auto-Computed Limits
When auto-computing animation limits:
Use limits from all frames
OK Apply Defaults Cancel

Figure 9.2: Viewport Annotation dialog on the left hand side, the Common Plot Options window in the middle and Contour Plot Options dialog on the right hand side.



Print

Selection

Print: Current viewport

Note: Minimized viewports will not be printed.

☐ Print viewport decorations (if visible)

☐ Print viewport backgrounds

☐ Print viewport compass (if visible)

Settings

Rendition: Color

Destination: Printer [✓ File]

File name: viewport.eps

Format: EPS

Encapsulated ...cript Options

Image Size

[✓ Use size on screen (8.1 x 6.3 in)] Use settings below

Width: 8.104 Height: 6.344

(Image aspect ratio is always preserved)

Units: [✓ Inches] Millimeters

Text Rendering

Always use PostScript printer fonts [✓ Use PostScript printer fonts when available] Always use displayed fonts (WYSIWYG)

Resolution: 300 dpi

Image Format: [✓ Vector] Raster

Shading Quality: Medium

Figure 9.3: Print window and Encapsulated PostScript Options window with customized options.

Remove all unwanted elements from the viewport using the Annotation Option window and select `File->Print` in the main menu. Select the current viewport and `eps` as output format in the Print window (cf. figure 9.3) and go to the Encapsulated PostScript Options window by clicking on the appropriate icon ( ) to choose 300 dpi as resolution.

The simulation can also be animated for any state variable, either by using the quick start symbol ( ), or by selecting `Animate->Time History` in the main menu. The animation can be saved by selecting `Animate->Save As`.

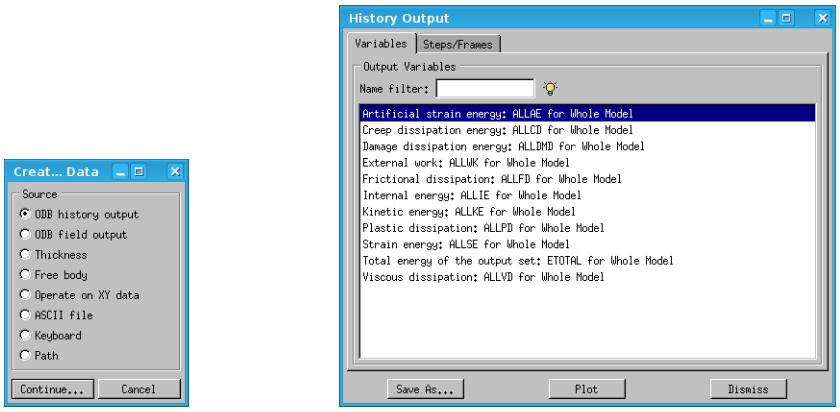
In case some variable states beneath the surface are of interest, a cut can be created. Open the View Cut Manager by selecting `Tools->View Cut->Manager` in the main menu or its icon ( ) next to the Activate View Cut icon ( ). The principal planes are already available and can be selected or custom planes can be created.

9.2 History output

Plots can be created by using the history output. Open the History Output window by either selecting `Result->History Output` in the main menu, or by right clicking on `XYData` in the model tree, choosing `Create` and selecting `ODB history output` in the Create XYData window (cf. figure 9.4). As for field output variables, only the selected values in the model output requests can be chosen. To save a data set for plotting elsewhere, choose `Report->XY` in the main menu and select the XY Data to save.

Sometimes it may be interesting not only to plot values over time, but over another variable. For every plot a `_temp_` variable will be created automatically in the model tree under `XYData`. The data can also be saved manually by `Save As...` from the History or Field Output window. Choose the variables and save their results, so that both entries are listed in the model tree under `XYData`.

Now select `Tools->XY Data->Create` in the main menu or right click on `XYData` in the model tree and choose `Create`. In the Create XYData window select `Operate on XY data`.



[✓ ODB history output]
ODB field output
Thickness
Free body
Operate on XY data
ASCII file
Keyboard
Path

```
Artificial strain energy: ALLAE for Whole Model
Creep dissipation energy: ALLCD for Whole Model
Damage dissipation energy: ALLDMD for Whole Model
External work: ALLWK for Whole Model
Frictional dissipation: ALLFD for Whole Model
Internal energy: ALLIE for Whole Model
Kinetic energy: ALLKE for Whole Model
Plastic dissipation: ALLPD for Whole Model
Strain energy: ALLSE for Whole Model
Total energy of the output set: ETOTAL for Whole Model
Viscous dissipation: ALLVD for Whole Model
```

Figure 9.4: Create XY Data window on the left hand side and History Output window on the right hand side.

Select the `combine()` Operator and double click on both XY data sets, so that something similar to `combine( "dataset1", "dataset2" )` is shown in the upper area. Now the data set can be plotted or saved.

Bibliography

ABAQUS Inc. (2005). *Abaqus/Explicit: Advanced Topics.* URL: »http://imechanica.org/node/15260.

Dassault Systèmes (2014). *Abaqus 6.14 Manual.*

Overvelde, J. T. B. (2010). *Learn Abaqus script in one hour.* URL: »http://bertoldi.seas.harvard.edu/files/bertoldi/files/learnabaqusscriptinonehour.pdf.