QSA Q-Bat - Tutorial 3

QuickerSim Automotive Ltd

Contents

1. Introduction **3**

2. Preparing the model **4**
2.1. Cell 4
2.2. Cooling plate 8
2.3. Heat component 11
2.4. Battery module 13

3. Calculations **15**
3.1. Model assembly and reduction 15
3.2. Running the calculations 17
3.3. Post-processing and export 18

1. Introduction

This tutorial is a modification of Tutorial 1. The differences are marked in bold and italics.

This tutorial presents how to create and simulate the heating of a simple battery geometry, consisting of one large cell, a cooling plate and surrounding air. The geometry is presented in Figure 1.

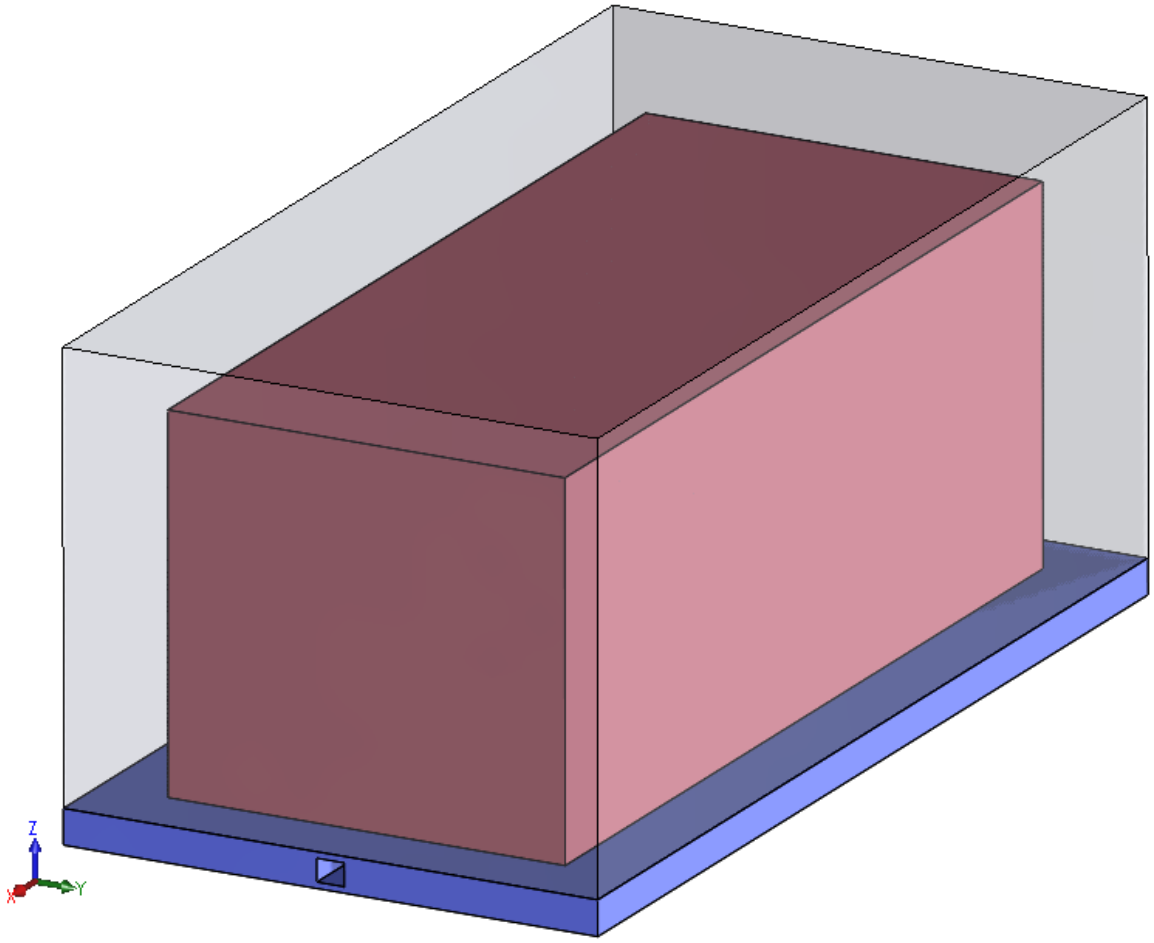

Figure 1: Battery assembly

Excel spreadsheets and msh files used in this tutorial have been prepared beforehand and are available as an attachment.

2. Preparing the model

Let's start by enforcing a good habit - clear your workspace and command window in Matlab and make sure all other windows are closed.

```
clc
clear
close all
```

The case described in this tutorial is not complicated and therefore does not require parallel toolbox.

2.1 Cell

We can now begin creating components of our assembly. First we will create the cell, which is represented by a simple prism, depicted on Figure 2.

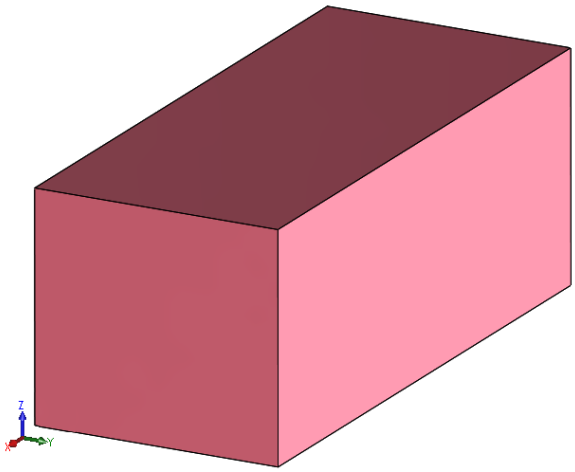

Figure 2: Cell

We have to create the cell prototype, which will be a virtual cell (not included anywhere in the model), with set properties and dimensions. We specify the type of our cell (prismatic or cylindrical) and its dimensions - the first three indices are the coordinates of the lower left corner of the prism and the other three - are its dimensions (x,y,z) in meters. It is also required to specify the material and electric properties of the cell, by appointing an Excel spreadsheet in which the data is collected.

```
cell_prototype = defineCellPrototype('type','prismatic','dimensions', ...
        [0, 0, 0, 0.1155, 0.264, 0.1055], 'data', ...
        'tutorials/tutorial1/data/cell_data.xlsx');
```

```
Reading mesh from Gmsh, version = 4.1
Reading section: Entites...  Done!
Reading section: Nodes... Done!
Reading section: Elements...  Done!
Mesh read succesfuly.
Material data rho depends on .
```

```
Material data lambda_x, lambda_y, lambda_z depends on .
Material data cp depends on .
Material data R_0 depends on T, SOC.
Material data R_1 depends on T, SOC.
Material data C depends on T, SOC.
Material data capacity depends on .
```

The data in the spreadsheet must be compliant with a certain model. Parameters are specified by writing the parameter name in a cell and its value below. It is also possible to enter non-constant parameters, depending for example on temperature. ***This can be done by adding a column with a T title next to the column containing the parameter values. For the cell electric parameters can also depend on the State of Charge (SOC), as shown in the table below.***

rho	lambda_y	lambda_x	lambda_z	cp	T	R_0	R_1	C	SOC	capacity
2000	21	3	21	550	-15	0.0013	0.0008	55000	1	130000
					-15	0.0014	0.0008	50000	0.9	
					-15	0.0014	0.0008	50000	0.8	
					-15	0.0014	0.0008	45000	0.7	
					-15	0.0014	0.0008	44000	0.6	
					-15	0.0014	0.0008	44000	0.5	
					-15	0.0014	0.0008	40000	0.4	
					-15	0.0014	0.0008	33000	0.3	
					-15	0.0017	0.0008	25000	0.2	
					-15	0.0018	0.0008	23000	0.1	
					-15	0.002	0.0008	12000	0	
					0	0.0009	0.0008	55000	1	
					0	0.001	0.0008	50000	0.9	
					0	0.001	0.0008	50000	0.8	
					0	0.001	0.0008	45000	0.7	
					0	0.001	0.0008	44000	0.6	
					0	0.001	0.0008	44000	0.5	
					0	0.001	0.0008	40000	0.4	
					0	0.001	0.0008	33000	0.3	
					0	0.001	0.0008	25000	0.2	
					0	0.0012	0.0008	23000	0.1	
					0	0.0012	0.0008	12000	0	
					50	0.0008	0.0008	55000	1	
					50	0.0008	0.0008	50000	0.9	
					50	0.0008	0.0008	50000	0.8	
					50	0.0008	0.0008	45000	0.7	
					50	0.0008	0.0008	44000	0.6	

Figure 3: Cell properties

Now we can display the created component and check the generated mesh.

```
cell_prototype.plot
```

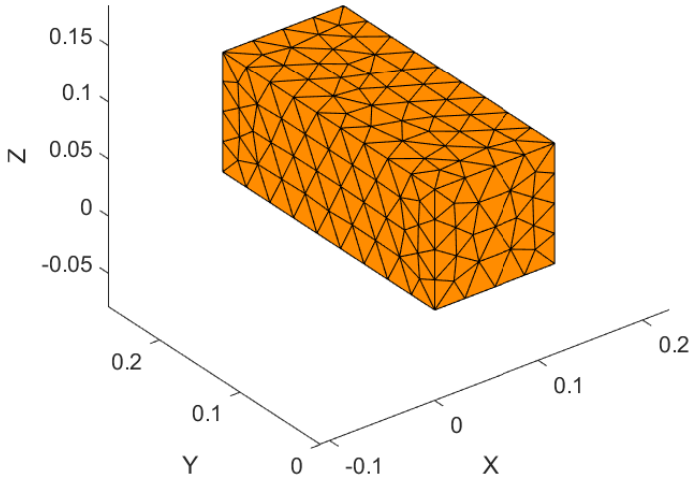

Figure 4: Cell mesh

Remember to save the cell prototype, in case you would like to use it in different models.

```
cell_prototype.saveComponent('tutorials\tutorial1\cell_proto')
```

In the next step we will have to put the cell in our simulation domain. We use the *instantiateInLocation* function and specify the location matrix of our cell. We must define the prototype which we want to use (in this case - the cell prototype).

```
cells = instantiateInLocation('prototype', cell_prototype, ...
    'location_matrix', [0.02, 0, 0]);
```

Now we want to add the electric current profile onto the cell. Using an Excel spreadsheet we load the current profile (as shown below), and rename the variables.

```
current = readtable("tutorials/tutorial3/data/electric_current.xlsx");
current.Properties.VariableNames = {'t','current'};
```

	A	B
1	time	current
2	1	200.0
3	10	200.0
4	20	133.3
5	30	113.3
6	40	266.7
7	50	266.7
8	60	180.0
9	70	186.7
10	80	193.3
11	90	200.0
12	100	200.0
13	110	200.0
14	120	166.7

Figure 5: Electric current profile data

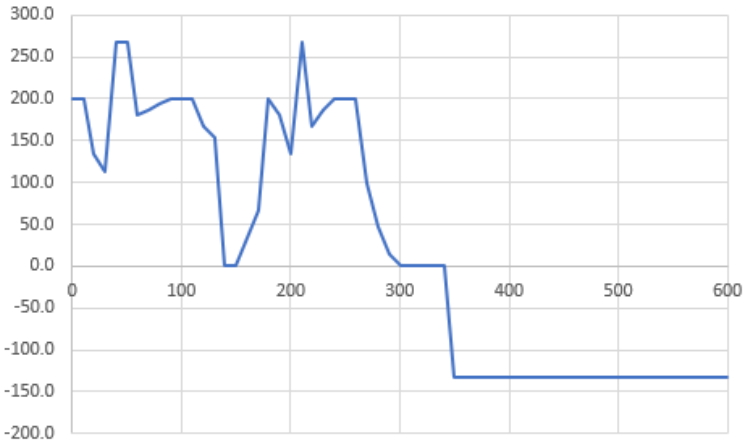

Figure 6: Electric current profile plot

Don't forget to assign the loaded profile to our cell.

```
cells.setCellCurrentProfile(current);
```

In order to be able to append the cells to our battery module we have to define them inside a cell casket.

```
cell_casket = defineCellCasket('cells', cells);
```

2.2 Cooling plate

After finishing the cell, we can move on to other components of the assembly. Next we will create the cooling plate, on which the cell stands (shown on Figure 7).

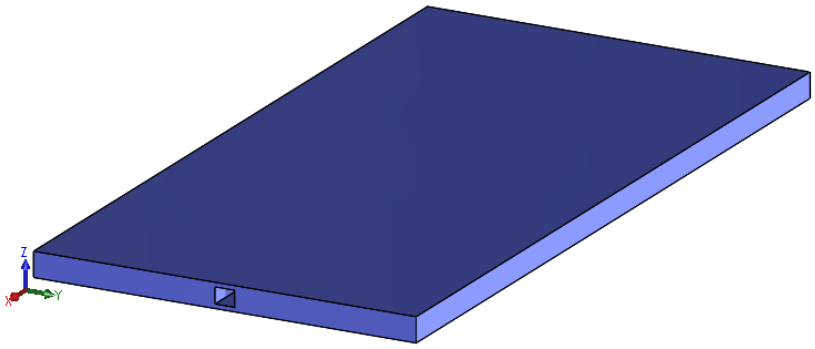

Figure 7: Cooling plate

In this tutorial the cooling plate geometry will be made by loading a previously made msh file. While defining the prototype, we have to define the msh file path, the pipe radius, mass flow rate, inlet temperature and material properties of the cooling plate and cooling fluid.

The pipe wall, inlet and outlet ids are know, so they can be defined by the User (if they are not known, the pipe walls and inlet/outlet can be chosen interactively, what is shown in Tutorial 2).

```
cooling_plate_prototype = defineCoolingPlatePrototype('mesh_path', ...
    'tutorials/tutorial3/msh/cp.msh', 'pipe_radius', 0.007172/2, ...
    'pipe_mass_flow_rate', 0.02, 'pipe_inlet_temperature', 15, ...
    'cooling_plate_data','tutorials/tutorial3/data/cp_data.xlsx', ...
    'coolant_data', 'tutorials/tutorial3/data/coolant_data.xlsx', ...
    'pipe_wall_ids', [7, 10, 9, 8], 'pipe_inlet_ids', [5, 6, 7, 8], ...
    'pipe_outlet_ids', [15, 16, 17, 18]);
```

```
Reading mesh from Gmsh, version = 4.1
Reading section: Entites...  Done!
Reading section: Nodes... Done!
Reading section: Elements...  Done!
Mesh read succesfully.
Pipe wall selected
Pipe inlet and outlet selected
Material data lambda depends on .
Material data rho depends on .
Material data cp depends on .
Material data rho depends on T.
Material data cp depends on T.
Material data lambda depends on T.
Material data nu depends on T.
Material data mi depends on T.
Material data beta depends on .
```

The Excel spreadsheets used in this step are shown below. To simplify the model, the cooling fluid's properties are assumed as constant and independent of the temperature.

	A	B	C	D
1				
2				
3		rho	lambda	cp
4		2700	130	870
5				

Figure 8: Cooling plate - material properties

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1																
2																
3			T	rho		T	cp		T	lambda		T	nu	mi		beta
4			0	1000		0	4180		0	0.6		0	0.000001	0.001		0.00341
5			100	1000		100	4180		100	0.6		100	0.000001	0.001		
6																

Figure 9: Cooling plate - cooling fluid properties

At this point it is very important to check if the pipe walls, inlet and outlet have been specified correctly.

```
cooling_plate_prototype.showPipeWall
```

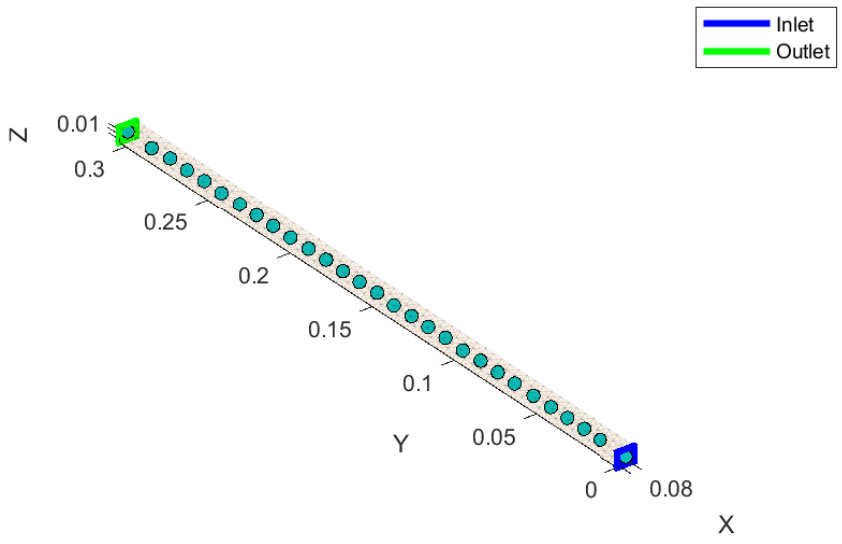

Figure 10: Cooling plate - pipe wall

Again, remember to save the cooling plate prototype:

```
cooling_plate_prototype.saveComponent('tutorials\tutorial3\cp_proto')
```

Once the prototype is created, we have to insert it into the simulation domain.

```
cooling_plate = instantiateInLocation('prototype', ...
    cooling_plate_prototype, 'location_matrix', [0, -0.02, -0.02]);
```

The component can now be displayed to make sure it has been created and placed correctly.

```
ui.utils.ComponentDisplayer.displayComponent(cooling_plate)
```

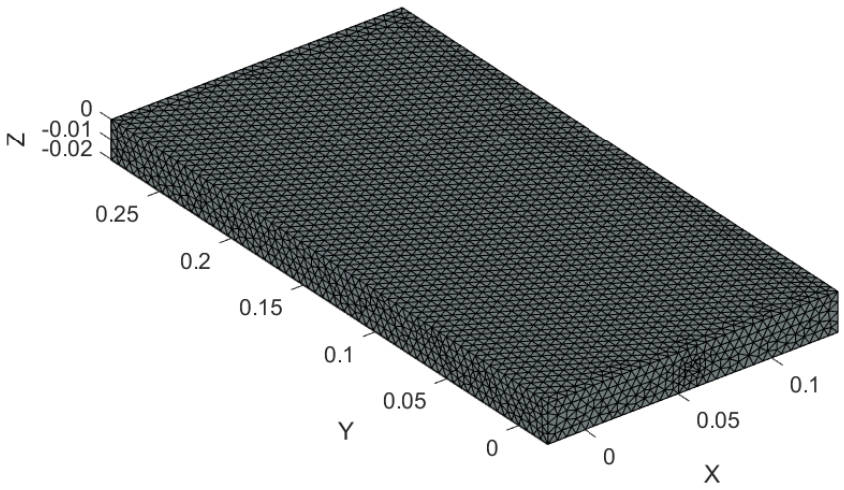

Figure 11: Cooling plate mesh

2.3 Heat component

The last component we need to create is the air surrounding the cell, which will be modeled as a passive heat component. A mesh has been created beforehand and its path will be given directly to the software.

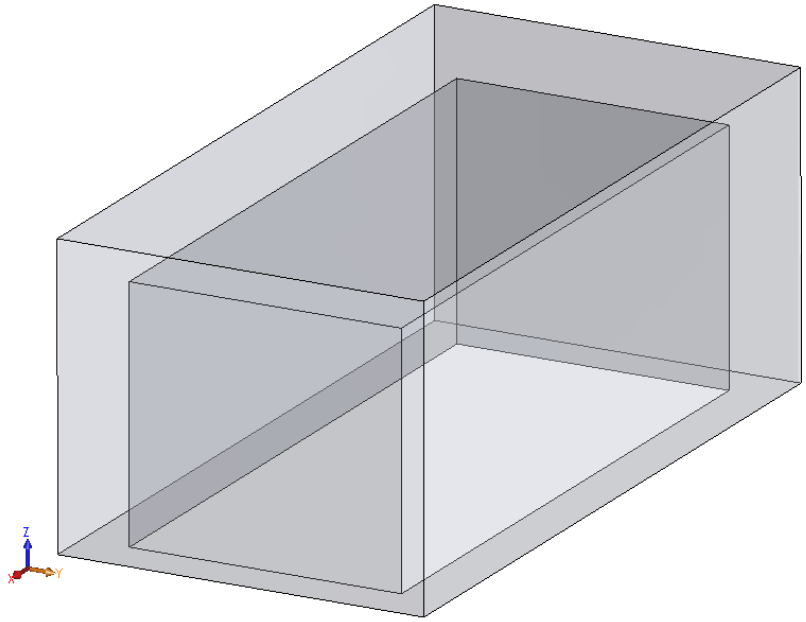

Figure 12: Air

Create the component in the same manner as before. The properties of the air are defined in an Excel spreadsheet (displayed on Figure 13).

	A	B	C	D	E	F	G	H	I	J
1										
2										
3				rho			cp			lambda
4				1.22			1000			0.024
5										
6										

Figure 13: Air - material properties

```
air_prototype = defineHeatComponentPrototype('mesh_path', ...
    'tutorials/tutorial3/msh/air.msh', 'heat_component_data', ...
    'tutorials/tutorial3/data/air_data.xlsx');
```

```
Reading mesh from Gmsh, version = 4.1
Reading section: Entites...  Done!
Reading section: Nodes... Done!
Reading section: Elements...  Done!
Mesh read succesfully.
Material data rho depends on .
Material data cp depends on .
Material data lambda depends on .
```

Additionally in this tutorial, we will add a boundary condition on the outer walls of the heat component to simulate a battery housing. For this purpose we use the *addRobinBCWithConductiveSheet* ***function where we specify the wall ids, heat transfer coefficient , surrounding temperature, and thermal conductivity and thickness of the housing.***

```
air_prototype.bcs.addRobinBCWithConductiveSheet('ids', [1, 2, 3, 4, 6], ...
    'alpha', 30, 't_inf', 15, 'lambda', 870, 'thickness', 0.002);
```

Remember to save the created prototype.

```
air_prototype.saveComponent('tutorials/tutorial3/air_proto');
```

The same way as before, we have to place the object into our simulation domain:

```
air = instantiateInLocation('prototype', air_prototype, ...
    'location_matrix',[0, 0, 0]);
```

The component must be translated, so it will be located in a proper place in the assembly.

```
air.translate([0, -0.02, 0.1255]);
```

The element is now placed correctly and can be displayed.

```
ui.utils.ComponentDisplayer.displayComponent(air)
```

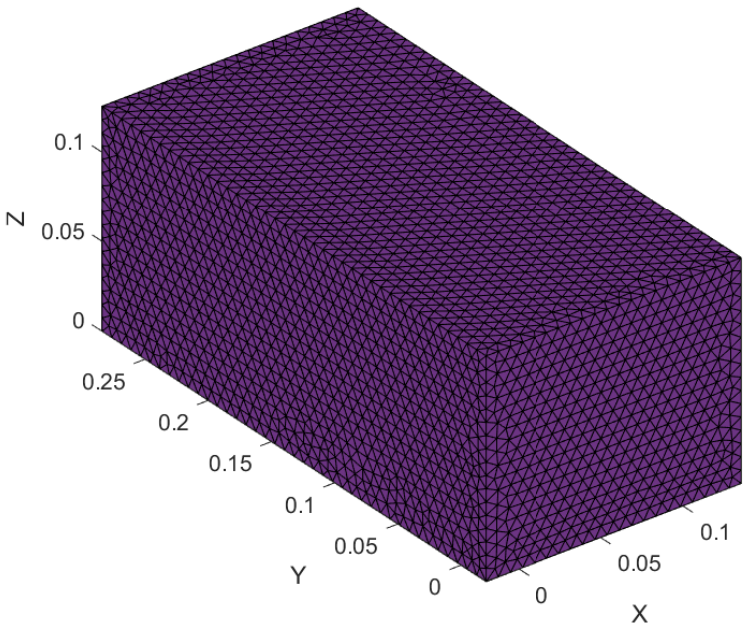

Figure 14: Air mesh

2.4 Battery module

All of the components have now been created. We can combine them, to obtain the full battery module.

Specify the cell caskets, cooling plates and heat components we want to use. Contacts inside the module will be added separately, as their values are not all equal.

```
battery_module = defineBatteryModule('cell_caskets', cell_casket, ...
    'cooling_plates', cooling_plate, 'heat_components', air);
```

Contacts are specified by stating the bodies between which the contact occurs, and the thermal conductivity and thickness. We want to simulate a thermal pad between the cells and cooling plate, with conductivity equal to $3\frac{\mathrm{W}}{\mathrm{m}\cdot\mathrm{K}}$ and 0.001 m thickness. The parameters of contact between the cell and air (heat component) have been adjusted to describe natural convection. To simulate contact between the cooling plate and air (no thermal pad) a high value of summary conductance was chosen (ratio of thermal conductivity and contact thickness equal to 10^{12}).

```
battery_module.setCouplingBetweenCellCasketsAndCoolingPlate(3,1e-3);
battery_module.setCouplingBetweenCellCasketsAndHeatComponents(5,1);
battery_module.setCouplingBetweenCoolingPlatesAndHeatComponents(1e6,1e-6);
```

Using the *displayComponent* function we can display our newly created model.

```
ui.utils.ComponentDisplayer.displayComponent(battery_module)
```

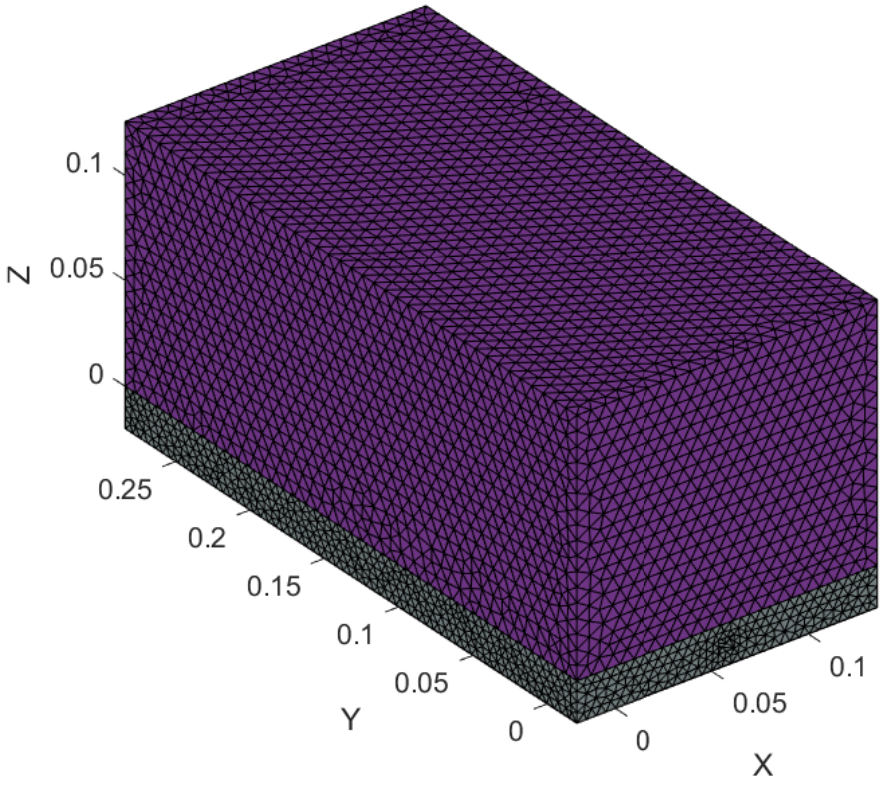

Figure 15: Battery module

Lastly, we set an initial temperature, that will be applied to the whole battery module.

```
battery_module.setInitialCondition(20);
```

Our battery module is done! We can now compile the model and run our simulation.

3. Calculations

3.1 Model assembly and reduction

The model is reduced and assembled using the *assembleModel* function. Adding *tic* and *toc* will allow to measure the assembly time.

```
t1 = tic;
m = assembleModel(battery_module);
t2 = toc(t1);
```

During the process various messages will be displayed in the command window. It is advised to read them, to assure the accuracy of the created model. It is specifically important to check if the number of contacts found by the software corresponds to the actual number of contacts in your model.

```
Found 1 thermal couples in BatteryModule: between CellCasket and
CoolingPlate.
Found 5 thermal couples in BatteryModule: between CellCasket and
HeatComponent.
Found 1 thermal couples in BatteryModule: between CoolingPlate and
HeatComponent.
Assembling domain 1. Number of nodes: 69701. Number of elements: 42839 ...
Elapsed time: 1.1155 s.
Assembling domain 2. Number of nodes: 53000. Number of elements: 30829 ...
Elapsed time: 0.8594 s.
Assembling domain 3. Number of nodes: 1807. Number of elements: 987 ...
Elapsed time: 0.055218 s.
Assembling couple 1
Elapsed time: 9.5447 s.
Assembling couple 2
Elapsed time: 3.4464 s.
Assembling couple 3
Elapsed time: 3.2848 s.
Assembling couple 4
Elapsed time: 1.4064 s.
Assembling couple 5
Elapsed time: 1.4159 s.
Assembling couple 6
Elapsed time: 3.6949 s.
Assembling couple 7
Elapsed time: 26.163 s.
Assembling thpipe couple 1
Elapsed time: 0.25507 s.
Assembling electro grids 1
Elapsed time: 0.0088236 s.
Reducing domain 1. Number of nodes: 69701. Number of elements: 42839 ...
Elapsed time: 13.151 s.
Reducing domain 2. Number of nodes: 53000. Number of elements: 30829 ...
Elapsed time: 26.3072 s.
```

```
Reducing domain 3. Number of nodes: 1807. Number of elements: 987 ...
Elapsed time: 0.23489 s.
Assembling and reducing nonlinear domain 1 out of 3
Elapsed time: 0.0041584 s.
Assembling and reducing nonlinear domain 2 out of 3
Elapsed time: 0.0003852 s.
Assembling and reducing nonlinear domain 3 out of 3
Elapsed time: 0.0003505 s.
```

3.2 Running the calculations

Once the model is compiled we can run the simulation. ***As the first parameter we specify the time step in seconds, next the number of time steps we want to simulate, and then the time step of the electrical calculation.***

```
m.runReducedMerged(10,60,1);
```

Congratulations, your simulation is done!

3.3 Post-processing and export

To have a clear space for post-processing close all the figures and define the axes ratio.

```
close all
cla
ax = gca;
```

Now using different post processing functions, we can plot temperatures in chosen bodies. Four parameters can be plotted:

- ***Minimum temperature over time***
- ***Maximum temperature over time***
- ***Mean temperature over time***
- ***Current over time***

Let's plot out the minimum temperatures in the cells of our battery. The functions take in the name of the whole created model, it's name after reduction and the type of components from which we want to display the solution.

```
plotComponentsMinTempOverTime(battery_module, m, 'cell', ax)
```

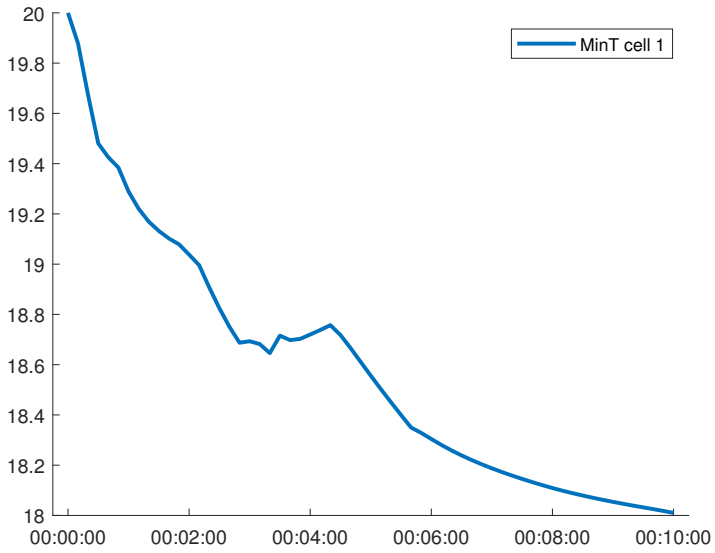

Figure 16: Minimum temperatures in cells

The same can be done with the maximum and mean temperature.

```
cla
plotComponentsMaxTempOverTime(battery_module, m, 'cell', ax)
```

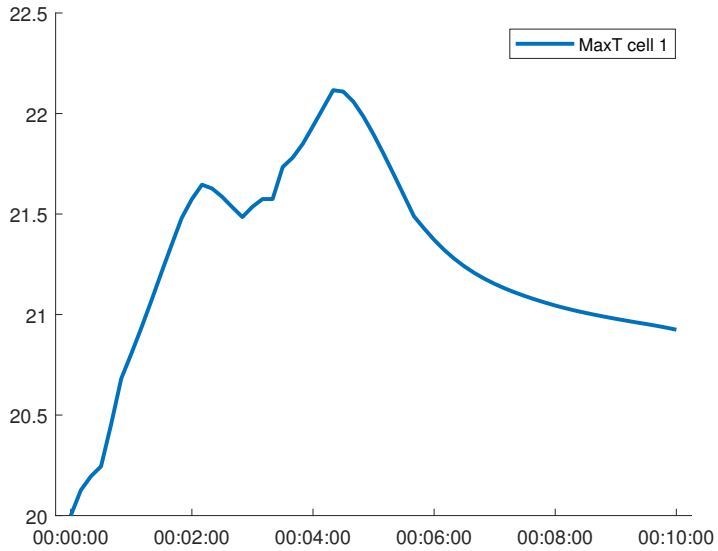

Figure 17: Maximum temperatures in cells

```
cla
plotComponentsMeanTempOverTime(battery_module, m, 'cell', ax)
```

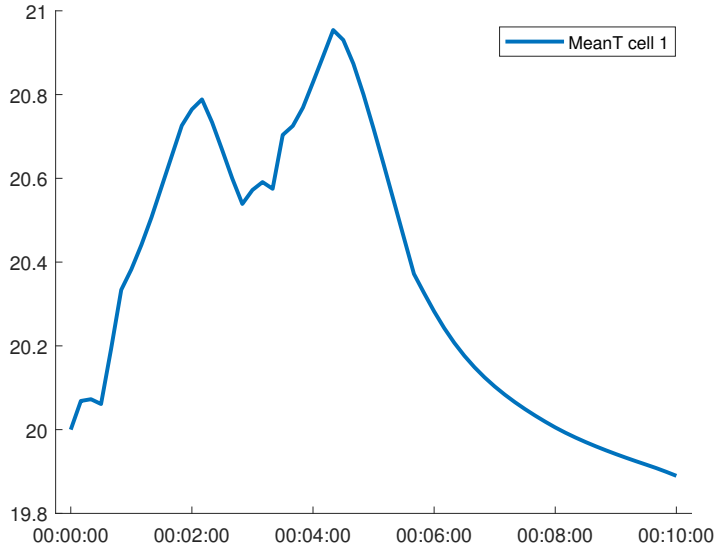

Figure 18: Mean temperatures in cells

Another interesting value can be the outlet temperature of the cooling fluid and its temperature profile throughout the pipe at a given time step. The outlet temperature can be plotted by specifying the cooling plate which is of interest and the reduced model.

```
cla
plotCoolantOutletTempOverTime(cooling_plate, m, ax)
```

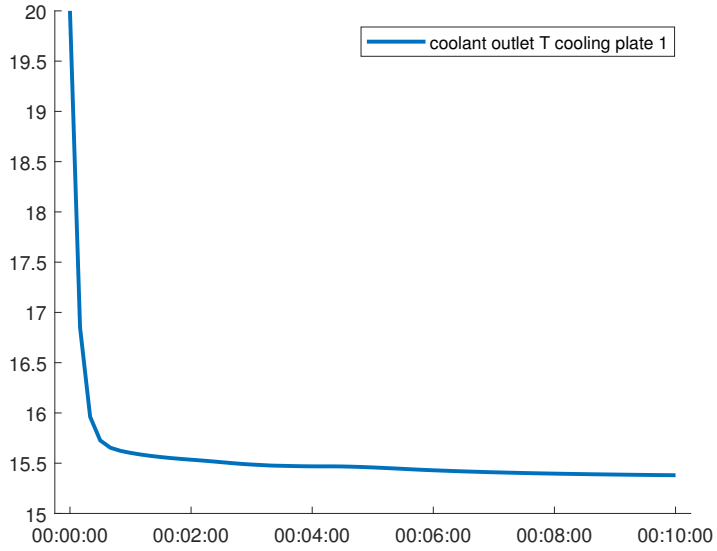

Figure 19: Cooling fluid outlet temperature in the cooling plate

```
cla
plotCoolantTempProfile(cooling_plate, m, 60, ax)
```

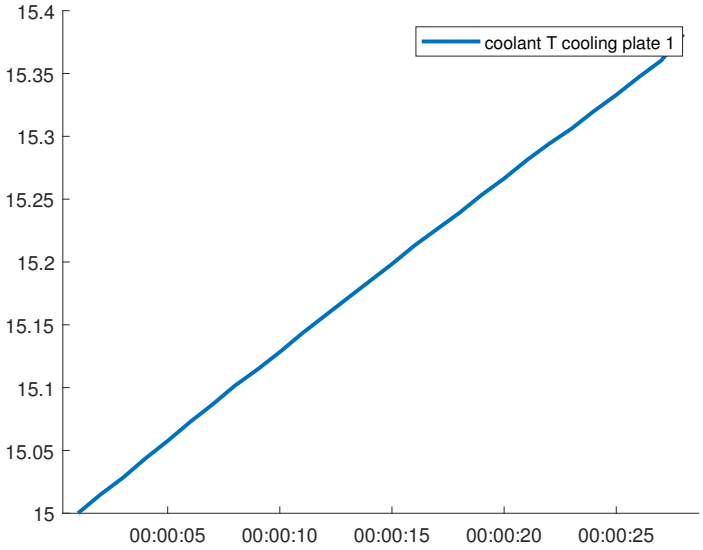

Figure 20: Cooling fluid temperature in the cooling plate

We can also plot the solution for the whole battery for a selected time step:

```
plotSolution(battery_module, m, 60)
```

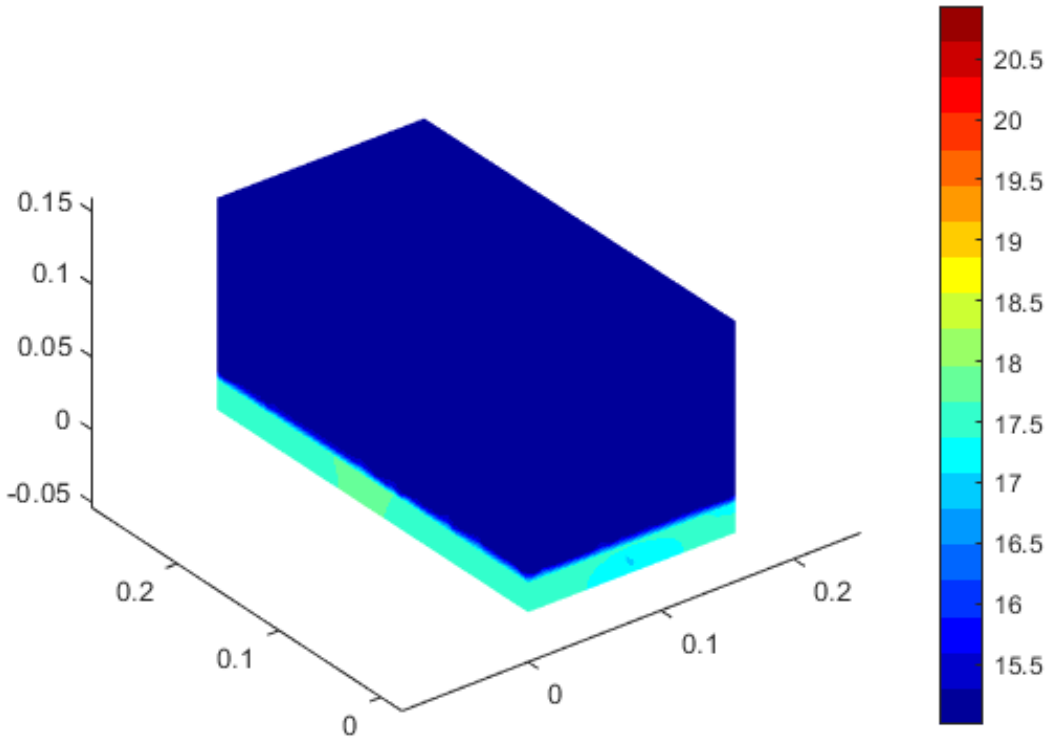

Figure 21: Solution in step 60

Lastly, we will export our solution to extensions which are compatible with other software, such as Paraview or Excel.

```
exportSolutionToVTK(battery_module, m, ...
    'tutorials/tutorial3/tutorial3_vtk');
exportSolutionToCSV(battery_module, m, ...
    'tutorials/tutorial3/tutorial3_solution.xls');
```