QSA Q-Bat - Tutorial 2

QuickerSim Automotive Ltd

Contents

1. Introduction

This tutorial presents how to create and simulate heat transfer in a more complicated battery geometry, consisting of four groups of cells and four cooling plates. The geometry can be seen as two identical battery islands, each containing two groups of cells and two different cooling plates. The geometry is presented below.

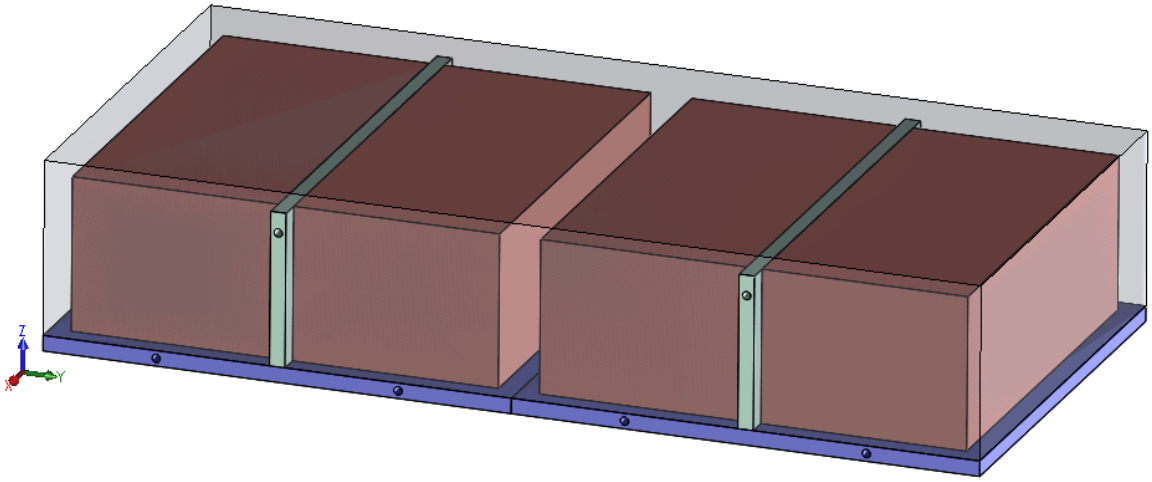

Figure 1: Battery assembly

All of the cooling plates are coupled hydraulically, as shown on Figure 2.

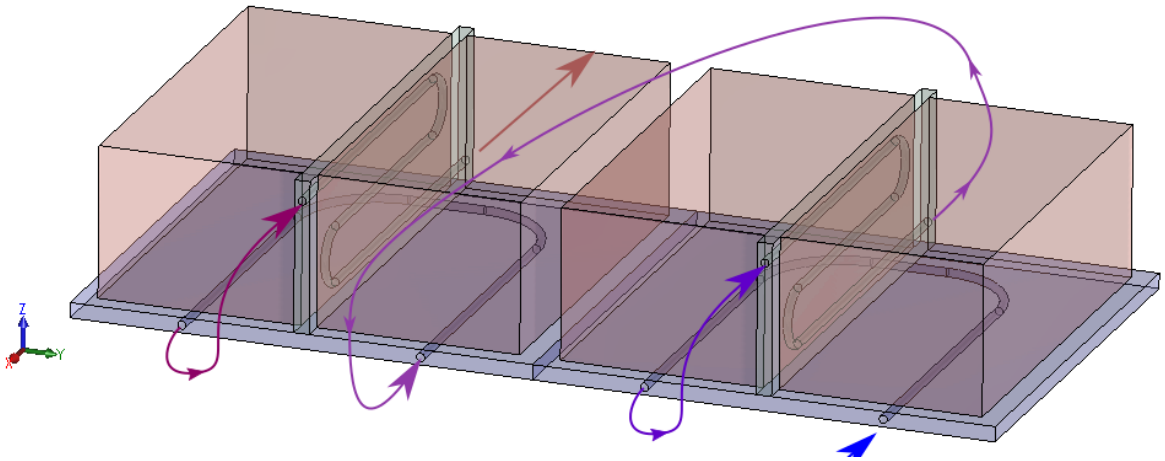

Figure 2: Battery assembly - hydraulic couplings

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
```

If you have parallel toolbox installed go ahead and start it (if not - skip this step):

```
gcp
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 6).
```

2.1 Cell

Now we can begin creating components of our assembly. First we will create the cells, which in our assembly will be represented as four larger prisms, depicted on Figure 3.

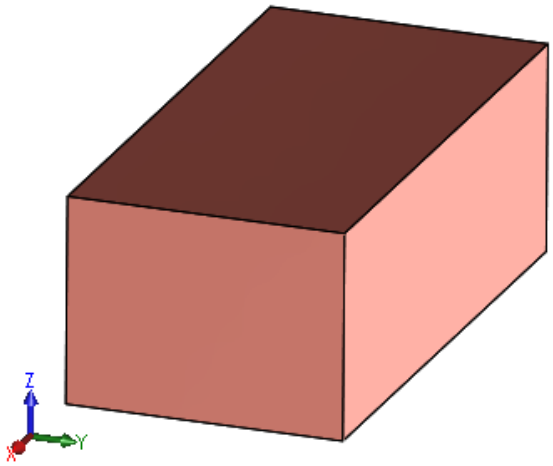

Figure 3: Cell

Firstly 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 have to 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 the 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, 0.4, 0.2, 0.15], 'mesh_size', 0.6, 'data',...
    'tutorials/tutorial2/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 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 4: Cell properties

In this tutorial we also specified the mesh size as a parameter. This is not a required (it can be used if the User wants to coarsen or densify the mesh) and is by default set to 1.

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

```
cell_prototype.plot
```

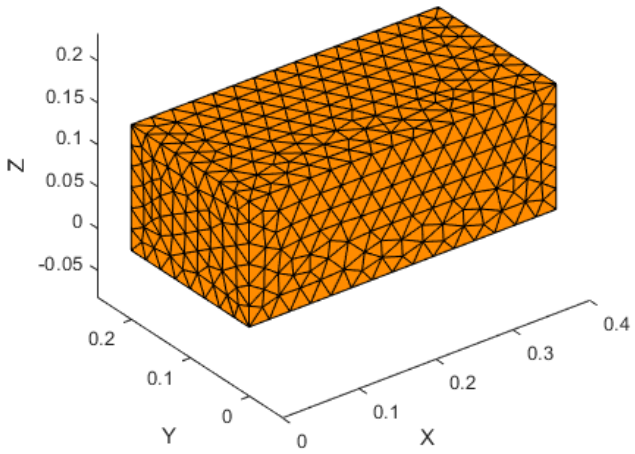

Figure 5: Cell mesh

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

```
cell_prototype.saveComponent('tutorials\tutorial2\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 two cells (we could also use the *instantiateinPattern* function and specify the type of pattern and its parameters). We must define the prototype which we want to use, which in this case is the *cell_prototype*.

```
cells = instantiateInLocation('prototype', cell_prototype, ...
    'location_matrix', [-0.42, 0.02, 0; -0.42, 0.235, 0]);
```

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

```
current = readtable("tutorials/tutorial2/data/electric_current.xlsx");
current.Properties.VariableNames = {'t','current'};
```

	A	B
1	time	current
2	1	600
3	10	600
4	20	400
5	30	340
6	40	800
7	50	800
8	60	540
9	70	560
10	80	580
11	90	600
12	100	600
13	110	600
14	120	500
15	130	460

Figure 6: Electric current profile data

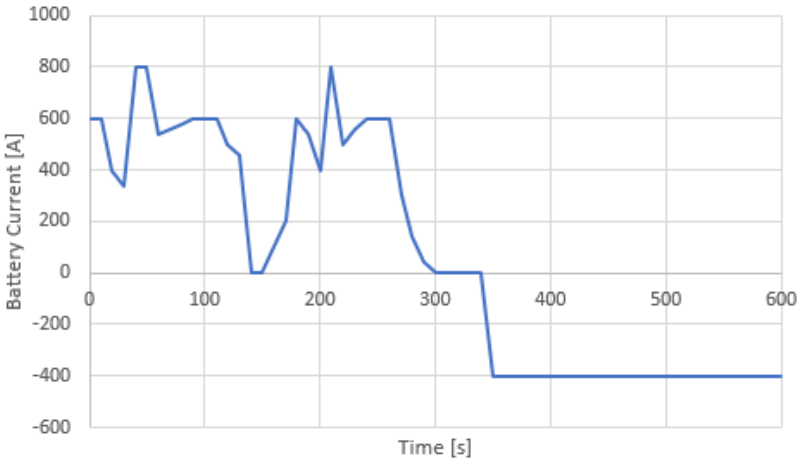

Figure 7: Electric current profile plot

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

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

We have created an array of two cells (what in our nomenclature is defined as a cell casket), with a full set of data. Aggregating them in a cell casket will allow us to append them to a battery module.

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

Cell caskets can also append heat components (which is not the case in this tutorial). A cooling plate will be added between the cells, but this is not supported at the level of cell casket creation (this will be done later, while defining a battery module).

2.2 Cooling plate #1

After finishing the cell, we can move on to other components of the assembly. Next we will create the larger cooling plate, on which the cells stand (shown on Figure 8).

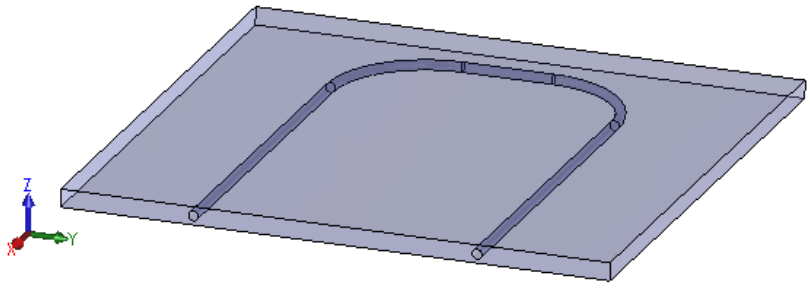

Figure 8: Cooling plate #1

Once again, we have to define the cooling plate prototype first. As cooling plates often have complicated geometries, it is recommended to mesh them in Gmsh software and import the msh file. Apart from the msh file path, we also have to define the pipe radius, mass flow rate, inlet temperature and material properties of the cooling plate and cooling fluid. Additionally, we define the *pipe_discretization* parameter to obtain a denser discretization of the pipe, due to its significant length (this is not a required parameter, by default it is set to 30).

```
cooling_plate_prototype_1 = defineCoolingPlatePrototype('mesh_path', ...
    'tutorials/tutorial2/msh/cp1.msh', 'pipe_radius', 0.004, ...
    'pipe_mass_flow_rate', 0.02, 'pipe_inlet_temperature', 10, ...
    'cooling_plate_data','tutorials/tutorial2/data/cp_data.xlsx', ...
    'coolant_data', 'tutorials/tutorial2/data/coolant_data.xlsx', ...
    'pipe_discretization', 100);
```

```
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 mi depends on T.
Material data nu depends on T.
Material data beta depends on .
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
```

The Excel spreadsheets used in this step are shown below. The cooling fluid's parameters depend on its temperature - what is clearly visible in the table.

	A	B	C	D
1				
2				
3		rho	lambda	cp
4		2700	130	870
5				
6				

Figure 9: Cooling plate #1 - 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	mi	nu		beta
4			-20	1100		-20	3320		-20	0.4		-20	0.0035	3.00E-06		0.003534
5			0	1070		0	3320		0	0.45		0	0.0035	3.00E-06		
6			100	1050		100	3320		100	0.5		100	0.0035	3.00E-06		
7																
8																

Figure 10: Cooling plate #1 - cooling fluid properties

Automatically, a figure of our cooling plate mesh opens. At this point the software allows us to manually choose the pipe walls. Walls can be selected by left-clicking on them and an action can be chosen from the drop-down menu (visible to the User after a right-click). The walls can either be hidden (to display the ones that are underneath) or selected as pipe walls. Make sure that you select all the walls that constitute a pipe.

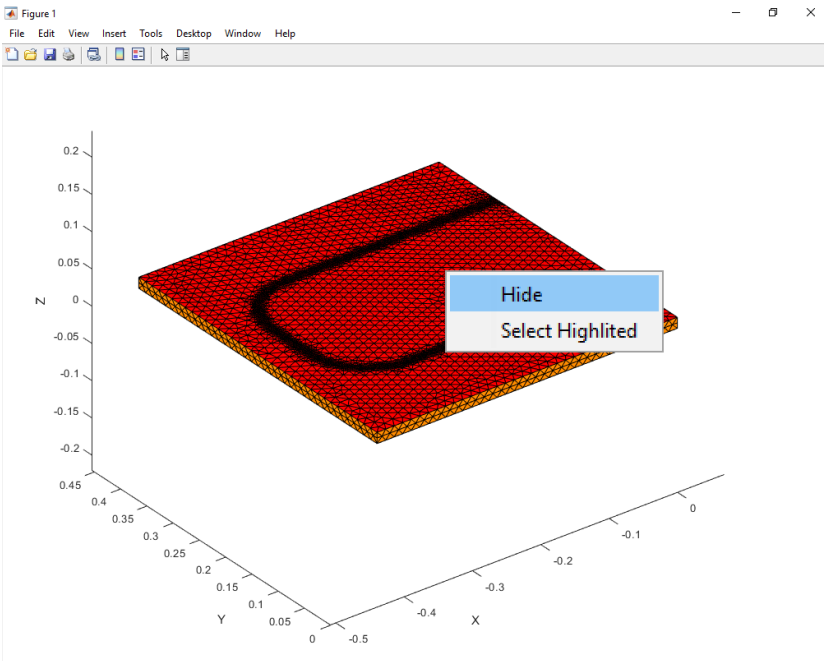

Figure 11: Hide surface

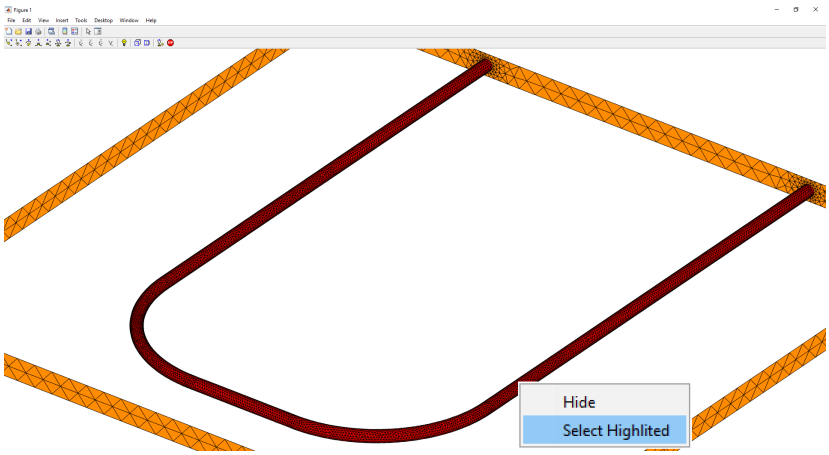

Figure 12: Select highlighted surfaces

After selecting the pipe walls, the figure closes automatically and another one opens. Now we can choose the pipe inlet and outlet. Select the edges of our inlet, right-click and select "Mark selection as Inlet" from the drop down list. The same has to be done with the outlet. In the end right-click anywhere on the figure and select "Done" to end the selection process.

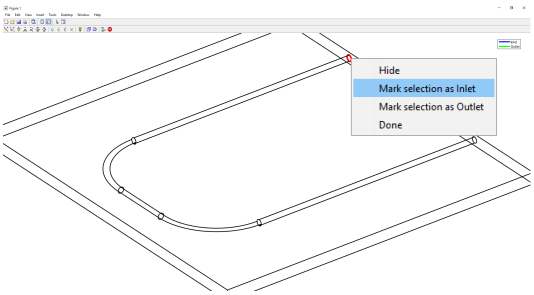

Figure 13: Select pipe inlet

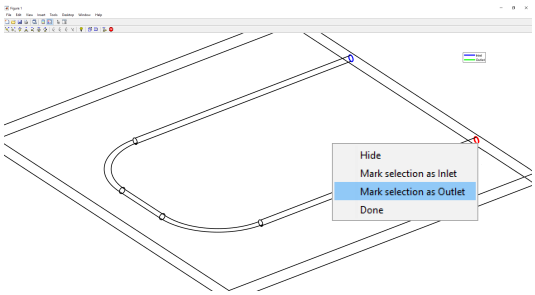

Figure 14: Select pipe outlet

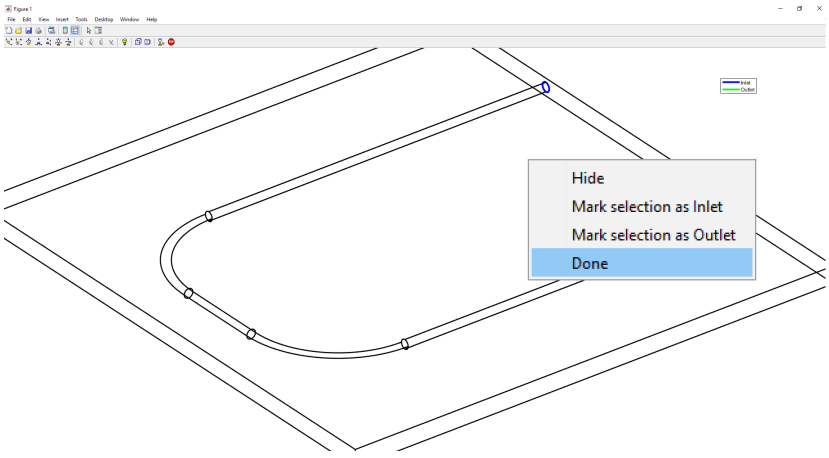

Figure 15: End selection process

The accuracy of our choices can be inspected by calling the *showPipeWall* function.

```
cooling_plate_prototype_1.showPipeWall
```

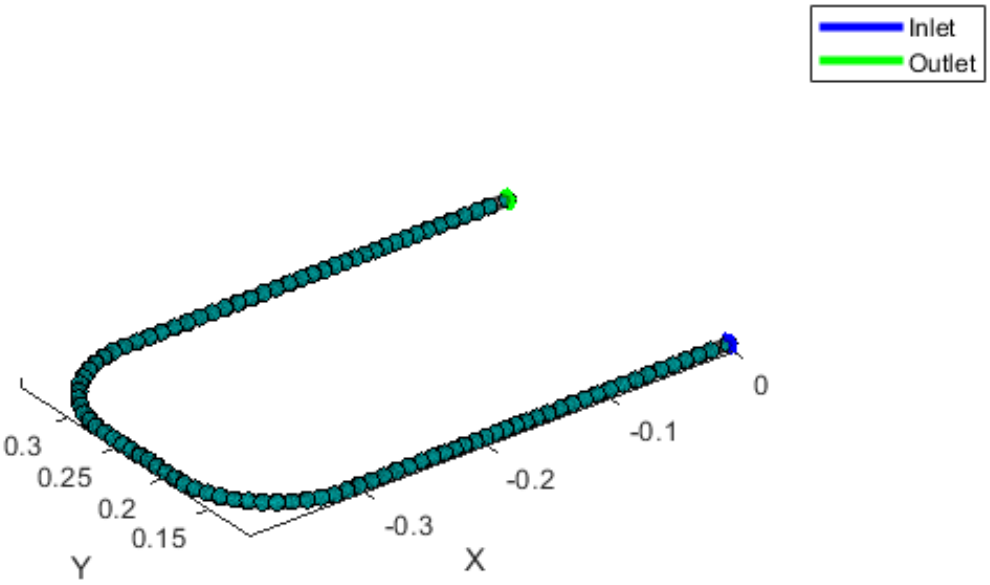

Figure 16: Cooling plate #1 - pipe walls

Remember to save the cooling plate prototype:

```
cooling_plate_prototype_1.saveComponent('tutorials\tutorial2\cp1_proto')
```

If the pipe is displayed incorrectly at this point, the selections can be modified by the following functions:

- *cooling_plate_prototype_1.choosePipeWall*
- *cooling_plate_prototype_1.choosePipeInletOutlet*
- *cooling_plate_prototype_1.discretizePipe*

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

```
cooling_plate_1 = instantiateInLocation('prototype', ...
    cooling_plate_prototype_1, 'location_matrix', [0, 0, -0.015]);
```

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

```
ui.utils.ComponentDisplayer.displayComponent(cooling_plate_1)
```

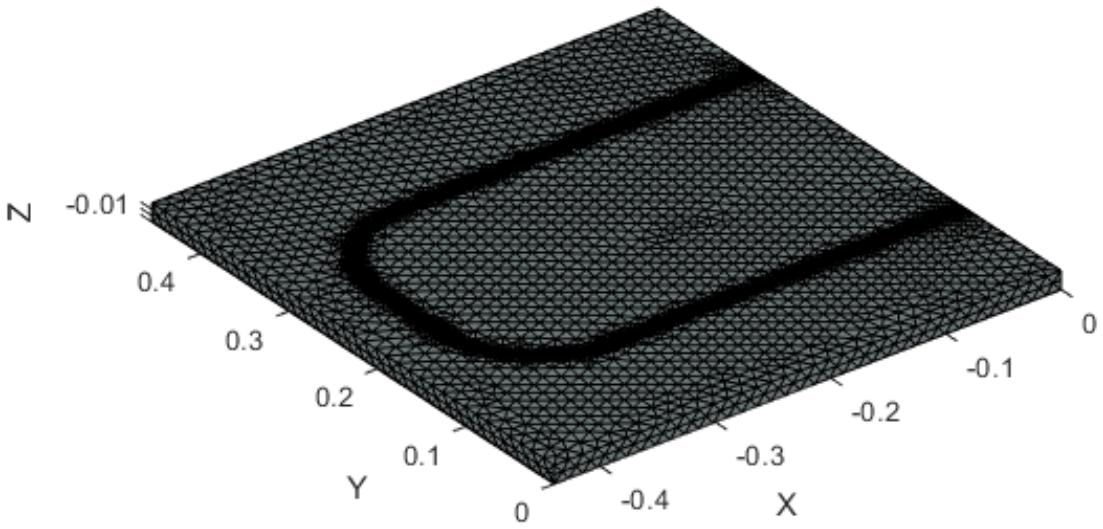

Figure 17: Cooling plate #1 mesh

2.3 Cooling plate #2

The next step will be creating the second cooling plate (shown on Figure 18) similarly as the first one.

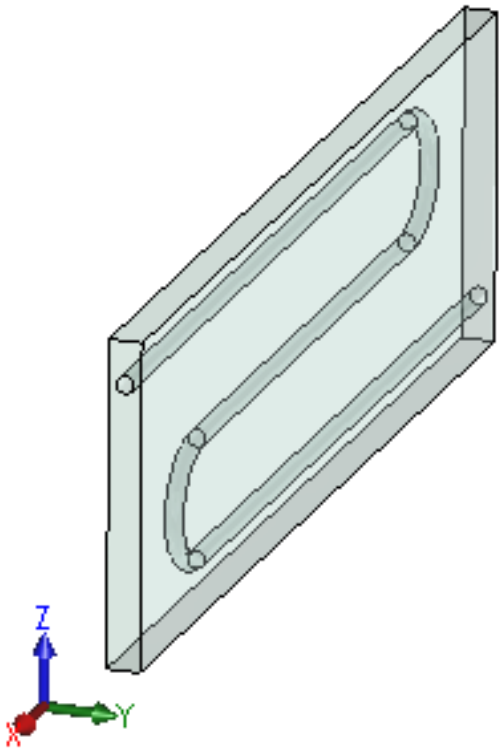

Figure 18: Cooling plate #2

This time the ids of the pipe walls, and inlet and outlet curves are known and can be specified as function arguments. The material properties are identical to those used before, although they could be changed (using different Excel spreadsheets).

```
cooling_plate_prototype_2 = defineCoolingPlatePrototype('mesh_path', ...
    'tutorials/tutorial2/msh/cp2.msh', 'pipe_radius', 0.004, ...
    'pipe_mass_flow_rate',0.02, 'pipe_inlet_temperature', 10, ...
    'cooling_plate_data','tutorials/tutorial2/data/cp_data.xlsx', ...
    'coolant_data', 'tutorials/tutorial2/data/coolant_data.xlsx', ...
    'pipe_wall_ids', [3, 12, 1, 13, 4, 14, 15, 2, 5, 16], ...
    'pipe_discretization', 100, 'pipe_inlet_ids', [10, 19], ...
    'pipe_outlet_ids', [18, 27]);
```

```
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 mi depends on T.
Material data nu depends on T.
```

```
Material data beta depends on .
Reading mesh from Gmsh, version = 4.1
Reading section: Entites...  Done!
Reading section: Nodes... Done!
Reading section: Elements...  Done!
Mesh read succesfuly.
Pipe wall selected
Pipe inlet and outlet selected
```

As a good habit, we can display the pipe walls, to make sure they have been selected properly.

```
cooling_plate_prototype_2.showPipeWall
```

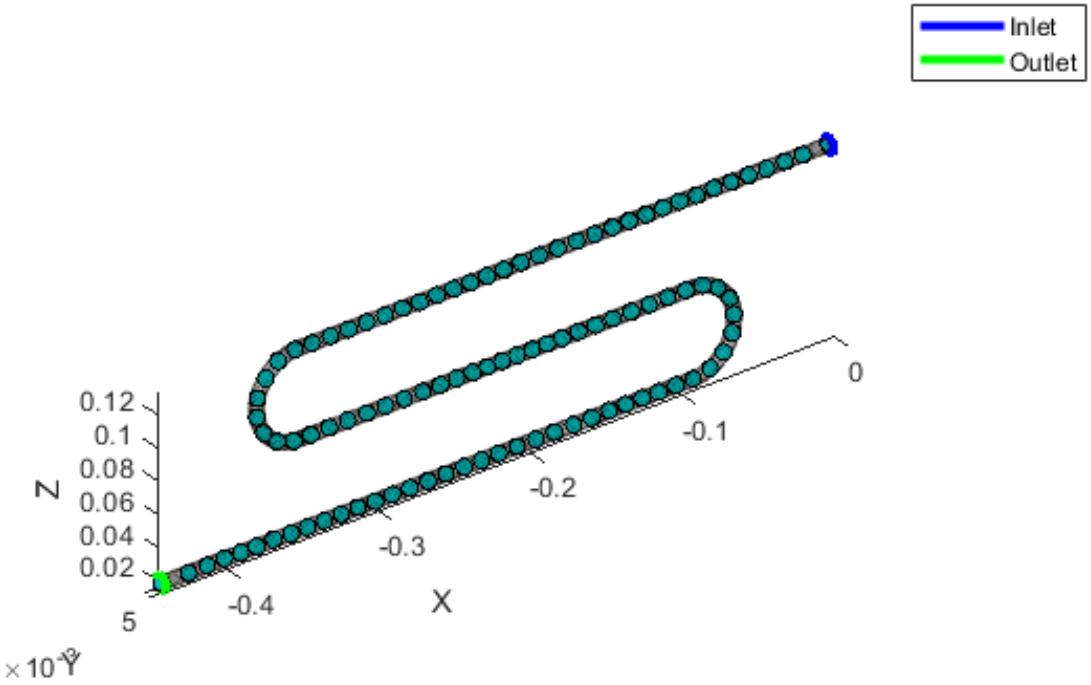

Figure 19: Cooling plate #2 - pipe walls

Remember to save the cooling plate prototype:

```
cooling_plate_prototype_2.saveComponent('tutorials\tutorial2\cp2_proto')
```

The cooling plate has to be inserted into the simulation domain. This component can now be viewed using the *displayComponent* function.

```
cooling_plate_2 = instantiateInLocation('prototype', ...
```

```
    cooling_plate_prototype_2, 'location_matrix', [0, 0.22, 0]);

ui.utils.ComponentDisplayer.displayComponent(cooling_plate_2)
```

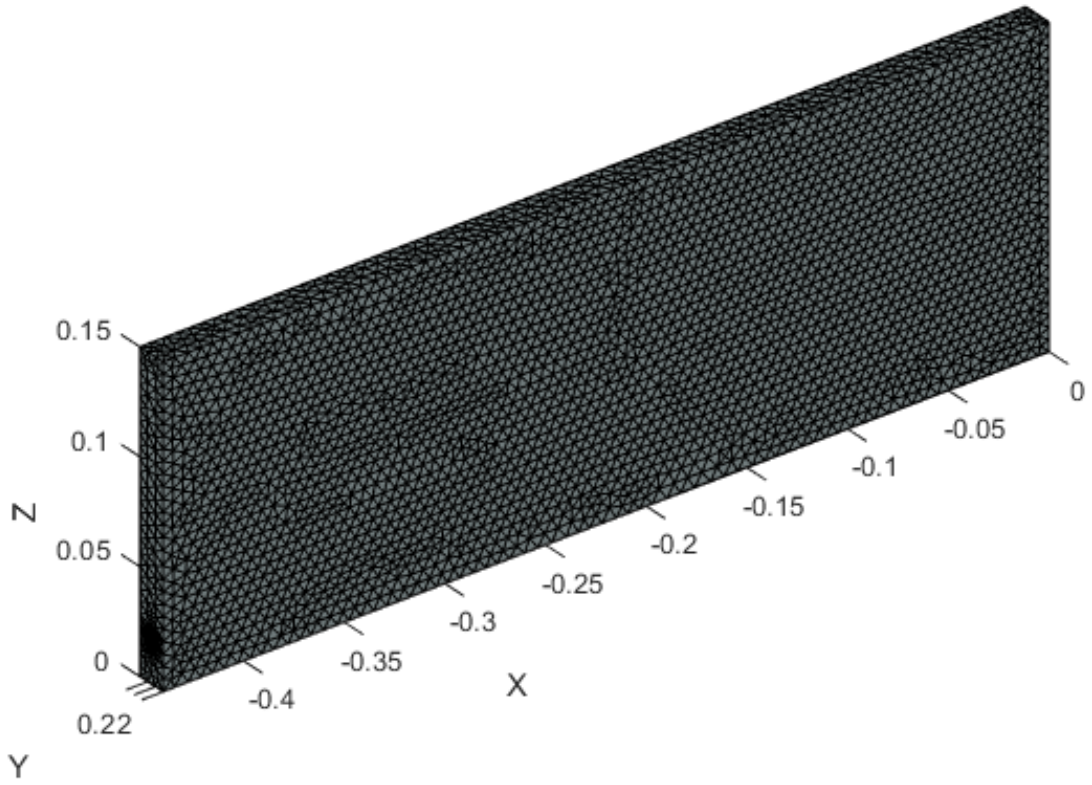

Figure 20: Cooling plate #2 mesh

2.4 Battery module

All of the components have now been created. We can start to aggregate them, to eventually obtain a full battery model.

Firstly we will create a battery module, which consist of two cells (the cell casket we created before) with the second cooling plate between them. Specify the cell caskets and cooling plates we want to use and add contacts inside the module. As we want to simulate thermal pads between the cells and cooling plate, we specify the conductivity of the contacts as $1 \frac{\mathrm{W}}{\mathrm{m \cdot K}}$ and the thickness as 0.001 m.

```
battery_module = defineBatteryModule('cell_caskets', cell_casket, ...
    'cooling_plates', cooling_plate_2, 'all_contacts', true, ...
    'conductivity', 1, 'contacts_thickness', 1e-3);
```

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

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

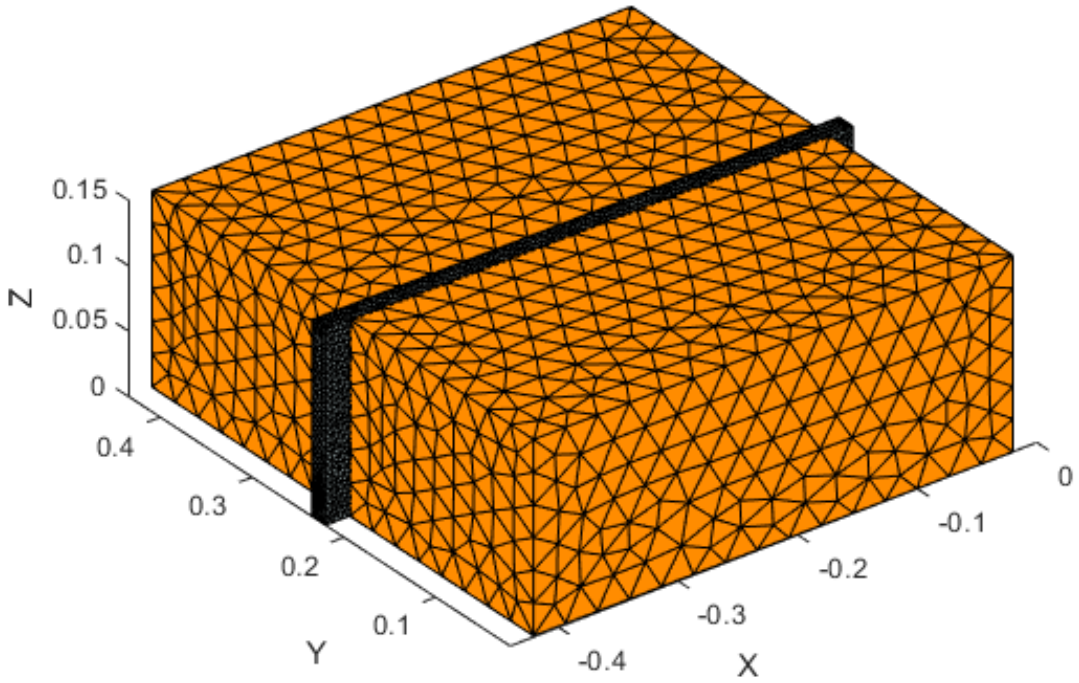

Figure 21: Battery module

2.5 Battery island

To add our first cooling plate to the assembly, we need to create a battery island. This is done in a similar manner, by defining the battery modules, additional cooling plates and contacts appearing in this assembly.

```
battery_island = defineBatteryIsland('battery_modules', battery_module, ...
    'cooling_plates', cooling_plate_1, 'all_contacts', true, ...
    'conductivity', 1, 'contacts_thickness', 1e-3);

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

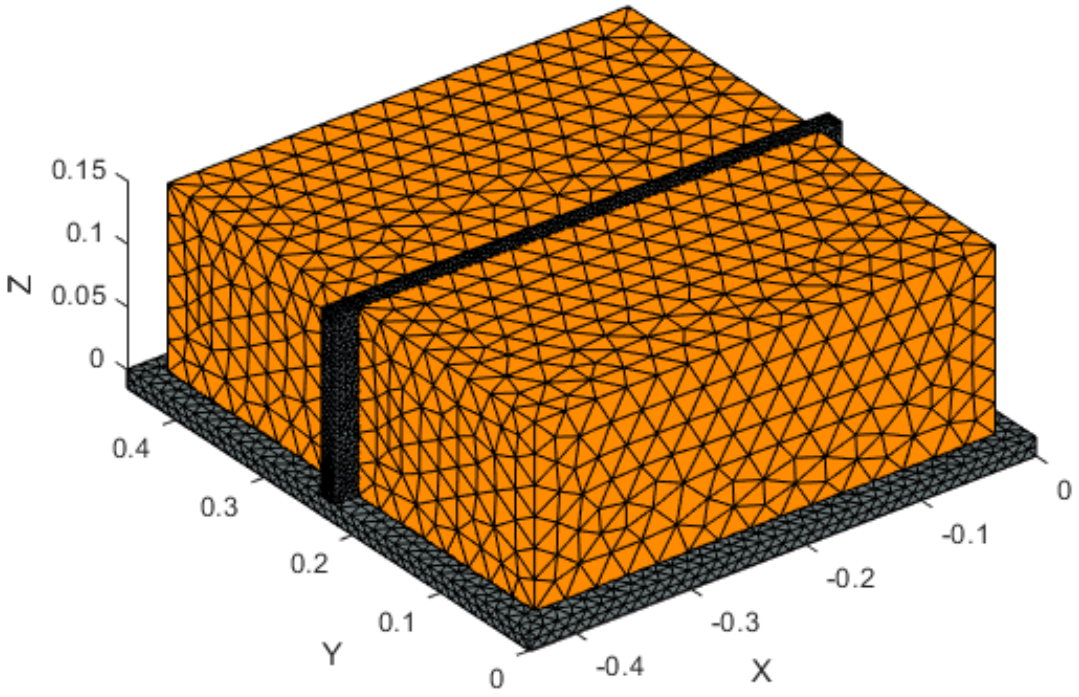

Figure 22: Battery island

Next, the battery island must be copied, as we want two of them side by side in our battery. Create an array of two battery islands by specifying their location matrix.

```
battery_islands = copyInstanceInLocation('instance', battery_island, ...
    'location_matrix', [0, 0, 0; 0, 0.455, 0]);
```

2.6 Battery

Finally we can create our battery by simply defining its battery islands. Add contacts between the components - in this case it will be the contact between two adhering battery islands (sides of their cooling plates). This time there are no thermal pads present, so a high value of summary conductance was chosen (ratio of thermal conductivity and contact thickness equal to 10^6).

```
battery = defineBattery('battery_islands', battery_islands, ...
    'all_contacts', true, 'conductivity', 1e3, 'contacts_thickness', 1e-3);
```

We set the initial conditional (temperature in degrees Celsius) of the battery and display the prepared model.

```
battery.setInitialCondition(20);

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

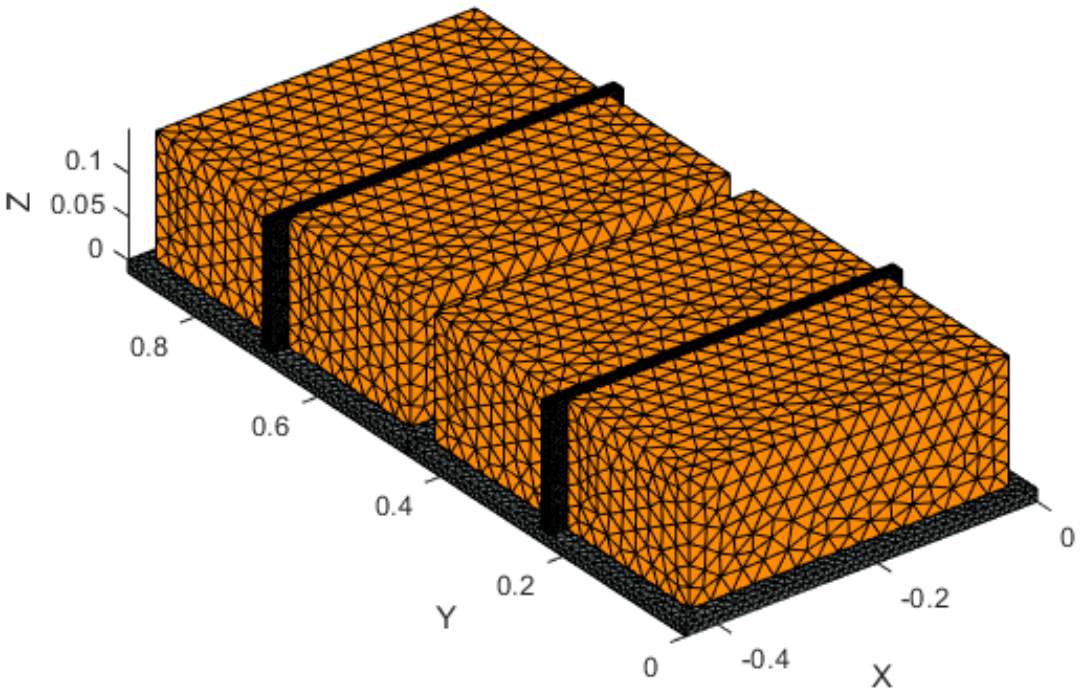

Figure 23: Battery

The last step will consist of coupling the existing cooling plates hydraulically. In order to achieve that, we need to specify all of the cooling plates in the model. We define *bi1_cp* and *bi2_cp* as the cooling plates inside respectively the first and second battery island (this will

be cooling plates #1, as they are not aggregated inside the battery modules).

```
bi = battery.getBatteryIslands();
bi1_cp = bi(1).getCoolingPlates();
bi2_cp = bi(2).getCoolingPlates();
```

Later we define *bi1_bm* and *bi2_bm* as the battery modules inside our battery islands.

```
bi1_bm = bi(1).getBatteryModules();
bi2_bm = bi(2).getBatteryModules();
```

When we have the battery modules specified, we can get the cooling plates within them (this will be cooling plates #2).

```
bi1_bm1_cp = bi1_bm(1).getCoolingPlates();
bi2_bm1_cp = bi2_bm(1).getCoolingPlates();
```

Once we specified all of the cooling plates, we can connect them, using the *connectCooling-PlatesPipes* function. The connections are always made between the outlet of the first cooling plate to the inlet of the second one and etc.

```
connectCoolingPlatesPipes('cooling_plates', ...
    [bi2_cp, bi2_bm1_cp,bi1_cp, bi1_bm1_cp]);
```

Our battery 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.

```
m = assembleModel(battery)
```

During the assembly and reduction various messages will be displayed in the command window. It is advised to read them, to assure the accuracy of the 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 2 thermal couples in Battery:BatteryIsland:BatteryModule: between
CellCasket and CoolingPlate.
Found 3 thermal couples in Battery:BatteryIsland: between BatteryModule
and CoolingPlate.
Found 2 thermal couples in Battery:BatteryIsland:BatteryModule: between
CellCasket and CoolingPlate.
Found 3 thermal couples in Battery:BatteryIsland: between BatteryModule
and CoolingPlate.
Found 1 thermal couples in Battery: between BatteryIsland and BatteryIsland.
Assembling domain 1. Number of nodes: 157206. Number of elements: 92433 ...
Elapsed time: 2.8525 s.
Assembling domain 2. Number of nodes: 162114. Number of elements: 95892 ...
Elapsed time: 2.9848 s.
Assembling domain 3. Number of nodes: 6175. Number of elements: 3619 ...
Elapsed time: 0.15499 s.
Parallel couple assemblation no. 1.
Parallel couple assemblation no. 3.
Parallel couple assemblation no. 5.
Parallel couple assemblation no. 7.
Parallel couple assemblation no. 9.
Parallel couple assemblation no. 11.
Elapsed time: 1.8547 s.
Elapsed time: 7.7456 s.
Parallel couple assemblation no. 10.
Elapsed time: 7.056 s.
Elapsed time: 23.0704 s.
Elapsed time: 22.568 s.
Elapsed time: 25.5443 s.
Elapsed time: 25.3054 s.
Parallel couple assemblation no. 4.
Elapsed time: 19.5607 s.
Parallel couple assemblation no. 8.
Elapsed time: 16.2499 s.
Parallel couple assemblation no. 2.
Elapsed time: 18.4956 s.
Parallel couple assemblation no. 6.
Elapsed time: 18.0184 s.
```

```
Assembling thpipe couple 1
Elapsed time: 14.3992 s.
Assembling thpipe couple 2
Elapsed time: 17.171 s.
Assembling thpipe couple 3
Elapsed time: 14.3096 s.
Assembling thpipe couple 4
Elapsed time: 17.2343 s.
Assembling electro grids 1
Elapsed time: 0.029133 s.
Reducing domain 1. Number of nodes: 157206. Number of elements: 92433 ...
Elapsed time: 56.806 s.
Reducing domain 2. Number of nodes: 162114. Number of elements: 95892 ...
Elapsed time: 49.505 s.
Reducing domain 3. Number of nodes: 6175. Number of elements: 3619 ...
Elapsed time: 1.1985 s.
Assembling and reducing nonlinear domain 1 out of 3
Elapsed time: 0.0001481 s.
Assembling and reducing nonlinear domain 2 out of 3
Elapsed time: 0.0001246 s.
Assembling and reducing nonlinear domain 3 out of 3
Elapsed time: 9e-05 s.
m =
  Model with properties:

                      full: 0
                    bodies: [1x8 engine.components.Body]
                     grids: [1x3 engine.components.Grid]
                   couples: [1x11 engine.components.Couple]
                   thpipes: [1x4 engine.components.thsolver.THPipe]
            thpipes_couples: [1x4 engine.components.thsolver.Couple]
              pipes_couples: [1x3 engine.components.thsolver.PipeCouple]
              electro_cells: [1x4 engine.components.electro.ElectroCell]
              electro_grids: [1x1 engine.components.electro.ElectroGrid]
                        dim: 0
          current_time_step: []
                    nGRDofs: 900
                 nGRSignals: 0
                    signals: [1x1 engine.components.Signal]
                  freeDOFs: [900x1 logical]
                  lockDOFs: [900x1 logical]
                        ur: []
                       ur0: [900x1 double]
                        tr: [1x0 double]
            electro_state: []
           electro_state0: [0 1 0 1 0 1 0 1]
               electro_tr: [1x0 double]
          electro_signals: [1x4 engine.components.Signal]
     electro_signal_value: []
  n_global_electro_states: 8
```

```
                    Fg: []
                    Mg: []
                    Kg: []
                   Fgs: [900x0 double]
               Fg_stat: [900x1 double]
               Kg_stat: [900x900 double]
special_fields_struct: [1x1 struct]
                    eM: []
                    eK: []
           source_val: []
      bodies_to_surtr: [1x1 struct]
```

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. We can additionally specify the maximum residual and maximum number of subiterations per time step (these parameters are not required).

```
m.runReducedMerged(10, 60, 1, 'maxRes', 0.1, 'maxSubIter', 20);
```

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.

```
cla
plotComponentsMinTempOverTime(battery, m, 'cell', ax)
```

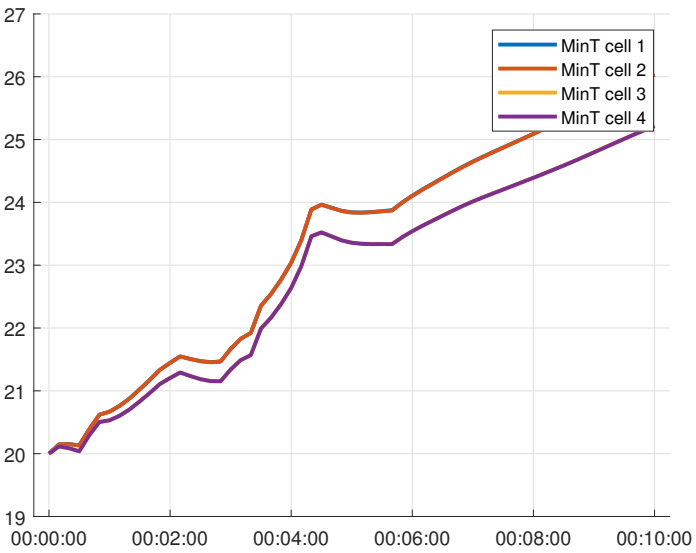

Figure 24: Minimum temperatures in cells

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

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

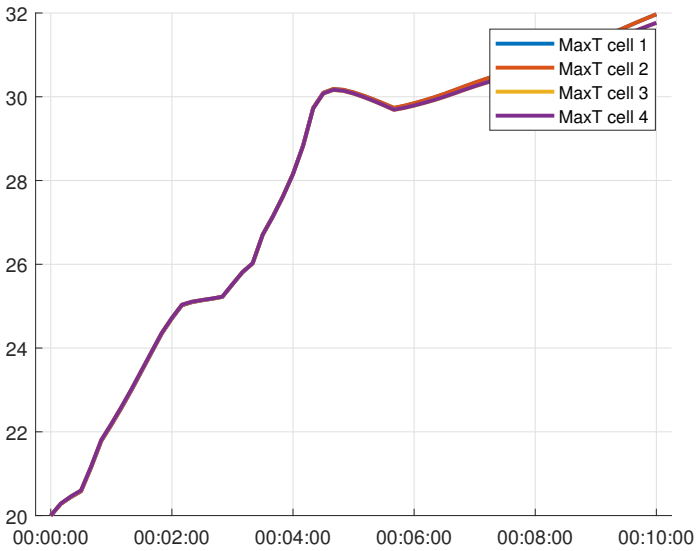

Figure 25: Maximum temperatures in cells

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

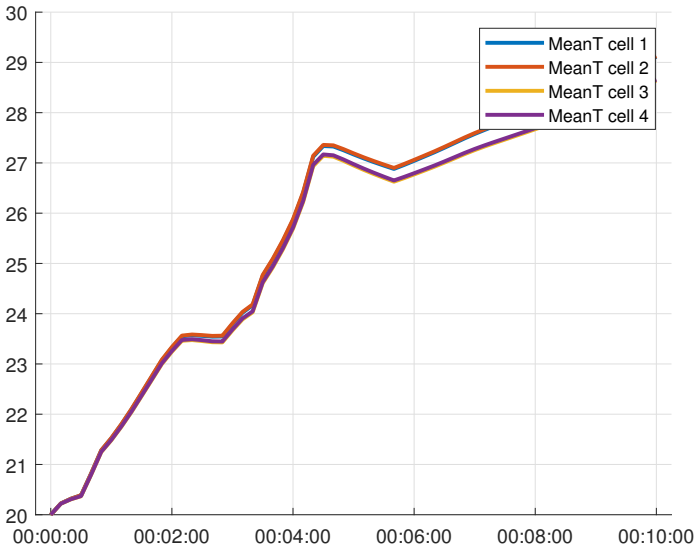

Figure 26: Mean temperatures in cells

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(bi1_bm1_cp, m, ax)
```

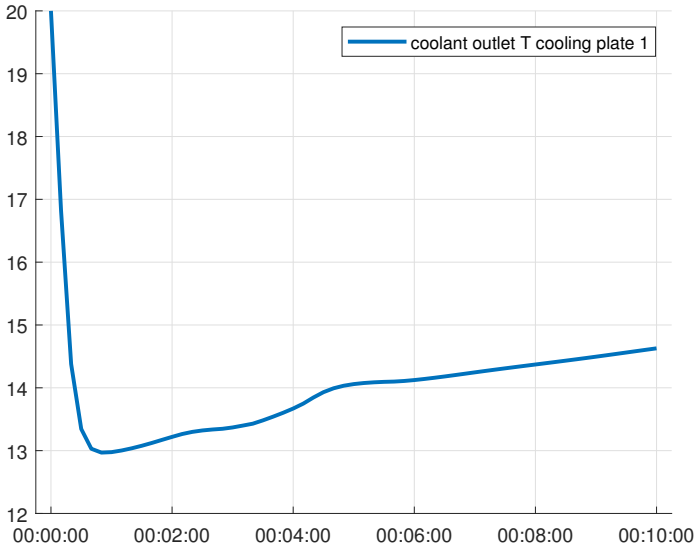

Figure 27: Cooling fluid outlet temperature in cooling plate *bi1_bm1_cp*

```
cla
plotCoolantTempProfile(bi1_bm1_cp, m, 60, ax)
```

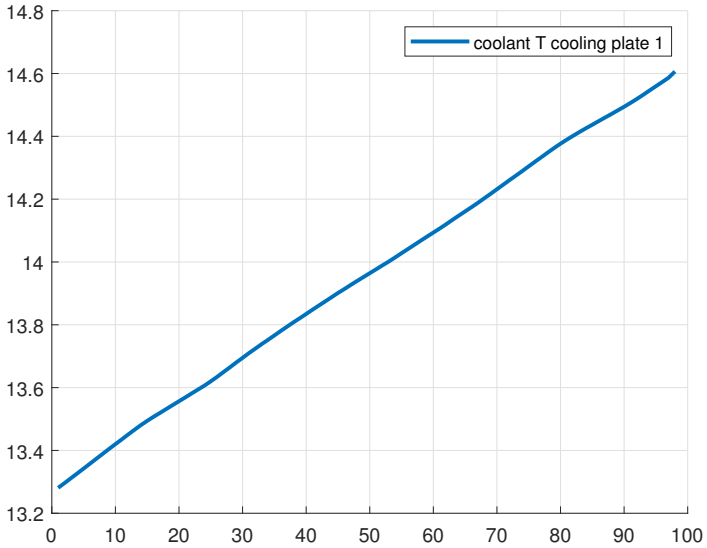

Figure 28: Cooling fluid temperature in cooling plate *bi1_bm1_cp* in the 60th step

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

```
plotSolution(battery, m, 60)
```

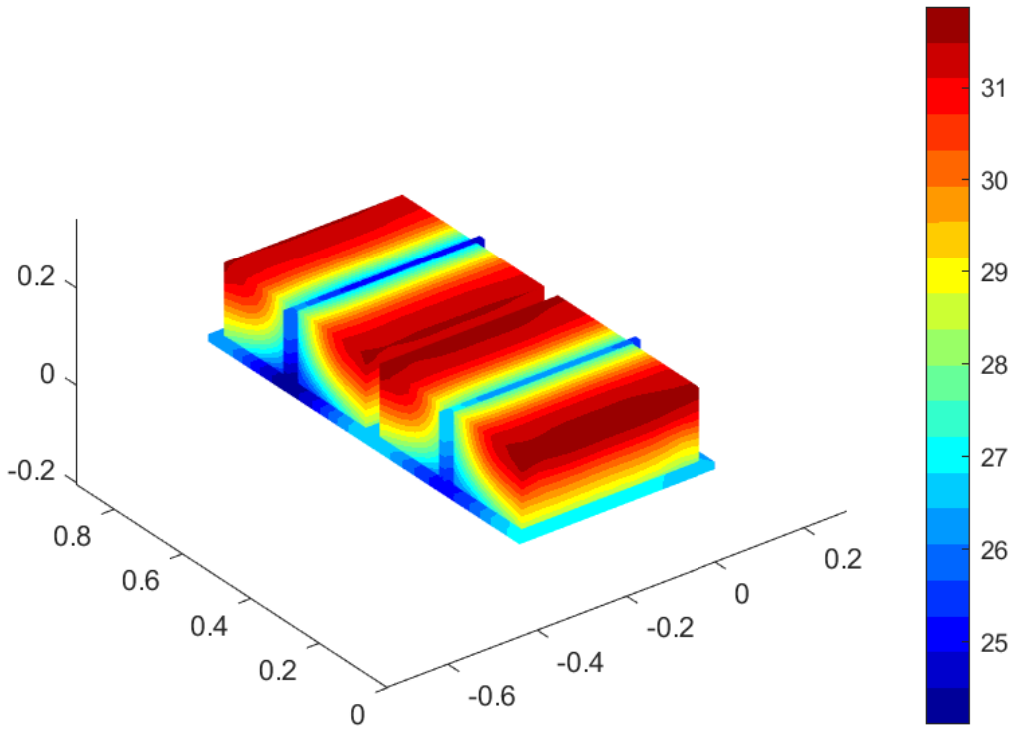

Figure 29: Solution in step 60

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

```
exportSolutionToVTK(battery, m, ...
    'tutorials/tutorial2/tutorial2_vtk');
exportSolutionToCSV(battery, m, ...
    'tutorials/tutorial2/tutorial2_solution.xls');
```