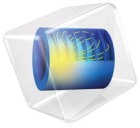


INTRODUCTION TO LiveLink™ *for* MATLAB®

6.2

COMSOL

Introduction to LiveLink™ *for* MATLAB®

Version: COMSOL 6.2

Contact Information

Visit the Contact COMSOL page at www.comsol.com/contact to submit general inquiries or search for an address and phone number. You can also visit the Worldwide Sales Offices page at www.comsol.com/contact/offices for address and contact information.

If you need to contact Support, an online request form is located on the COMSOL Access page at www.comsol.com/support/case. Other useful links include:

- Support Center: www.comsol.com/support
- Product Download: www.comsol.com/product-download
- Product Updates: www.comsol.com/product-update
- COMSOL Blog: www.comsol.com/blogs
- Discussion Forum: www.comsol.com/forum
- Events: www.comsol.com/events
- COMSOL Video Gallery: www.comsol.com/videos
- Support Knowledge Base: www.comsol.com/support/knowledgebase

Part number: CM020010

Contents

Introduction 5

Starting COMSOL Multiphysics® with MATLAB® 7

A Thorough Example: The Busbar 9

Extracting Results at the MATLAB® Command Line. 29

Automating with MATLAB® Scripts. 36

Using External MATLAB® Functions 46

Introduction

LiveLink™ *for* MATLAB® connects COMSOL Multiphysics® to the MATLAB® scripting environment. Using this functionality, you can do the following:

- Set up models from a script. LiveLink™ *for* MATLAB® includes the COMSOL® application programming interface (API), which has all the necessary functions and methods to implement models from scratch. For each operation performed in the COMSOL Desktop® there is a corresponding command that is entered at the MATLAB prompt. It is a simplified syntax based on the Java® programming language and does not require any Java knowledge.
- Use MATLAB functions in model settings. Use LiveLink™ to set model properties with a MATLAB function. For example, define material properties or boundary conditions as a MATLAB routine that is evaluated while the model is solved.
- Interactive modeling between the COMSOL Desktop and MATLAB sharing the same model. Every modification performed at the MATLAB prompt is simultaneously updated in the COMSOL Desktop.
- Leverage MATLAB functionality for program flow. Use the COMSOL API syntax together with MATLAB functionality to control the flow of your programs. For example, implement nested loops using `for` or `while` commands, implement conditional model settings with `if` or `switch` statements, or handle exceptions using `try` and `catch`.
- Analyze results in MATLAB. The API wrapper functions included make it easy to extract data at the command line. Functions are available to access results at node points or arbitrary locations. You can also get low-level information about the extended mesh, such as finite element mesh coordinates, and connection information between the elements and nodes. Extracted data are available as MATLAB variables ready to be used with any MATLAB function.
- Connect to a Model Manager database and load, save and search for model. Although most if the interactions with models stored in a Model Manager database still should be performed with the Model Manager user interface some tasks can be automated using the COMSOL API as well as some useful MATLAB function.
- Create custom interfaces for models. Use the *MATLAB Guide* or the *App Designer* functionality to create a user-defined graphical interface that is combined with a COMSOL model. Make your models available to others by

creating graphical user interfaces tailored to expose the settings and parameters of your choice.

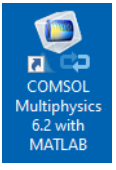
- LiveLink™ *for* MATLAB® can connect to COMSOL Server™ as well as COMSOL Multiphysics server. This means that MATLAB scripts and GUIs that utilize COMSOL functionality can be distributed to and used by any user that has access to COMSOL Server™.

The examples in this guide take you through the process of setting up a COMSOL model and explain how to use COMSOL Multiphysics within the MATLAB scripting environment.

Starting COMSOL Multiphysics® with MATLAB®

Starting on Windows®

To start COMSOL Multiphysics with MATLAB, double-click the COMSOL with MATLAB icon available on the desktop.



This opens the MATLAB desktop together with the COMSOL Multiphysics server, which is represented by the command window appearing in the background.

Starting on macOS

Navigate to Applications>COMSOL 6.2>COMSOL 6.2 with MATLAB.

Starting on Linux®

Start a terminal prompt and run the `comsol` command, which is located inside the bin folder in the COMSOL installation directory:

```
comsol mphserver matlab
```

The COMSOL Client–Server Connection

LiveLink™ *for* MATLAB® provides an interface between COMSOL Multiphysics and MATLAB based on the COMSOL client–server architecture. A COMSOL thin client is running inside MATLAB and has access to the COMSOL API through the MATLAB Java interface. Model information is stored in a *model object* available on the COMSOL Multiphysics server. The thin client communicates with the COMSOL Multiphysics server, enabling you to generate, modify, and solve COMSOL model objects at the MATLAB prompt.

When starting COMSOL with MATLAB, you open both a COMSOL Multiphysics server and the MATLAB user interface. The first time you start a COMSOL Multiphysics server, you are asked to supply a new username and password that will be associated with the client–server mode of operation. These credentials are stored in the user preferences and reused for future connections. Once this information is entered, the client–server communication is established.

Normally, both the COMSOL Multiphysics server and the MATLAB user interface run on the same computer. For computations requiring more memory, you can connect to a remote COMSOL Multiphysics server, but this configuration requires a floating network license.

Note that the COMSOL Desktop is not necessary when running COMSOL Multiphysics with MATLAB. However, you can connect a COMSOL Desktop to the COMSOL Multiphysics server and import the model available in the latter. This way the model is updated simultaneously at the MATLAB prompt and in the COMSOL Desktop.

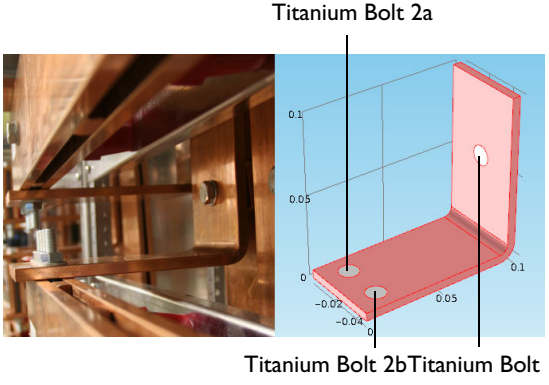
A Thorough Example: The Busbar

This tutorial familiarizes you with the COMSOL model object and the COMSOL API syntax. In this section, learn how to:

- Create a geometry
- Set up a mesh and apply physics properties
- Solve the problem
- Generate results for analysis
- Exchange the model between the scripting interface of MATLAB® and the COMSOL Desktop.

The model you are building at the MATLAB command line is the same model described in the *Introduction to COMSOL Multiphysics*. The difference is that, in this guide, you will use a COMSOL model object instead of the COMSOL Desktop.

This multiphysics example describes electrical heating in a busbar. The busbar is used to conduct direct current from a transformer to an electrical device and is made of copper with titanium bolts, as shown in the figure below.



Note: The step-by-step instructions below are designed to be carried out in a sequence. Skipping any of the sections might result in data being unavailable for the following sections. Start with About The Model Object and work through the sections until reaching the last section, Saving the Model.

About Compact Notation

This example uses *compact notation* to shorten the commands that are entered at the MATLAB command line. The compact notation uses MATLAB variables as links to provide direct access to COMSOL model object features.

For example, to create a block using the compact notation, enter:

```
blk = geom.feature.create('blk1', 'Block');
blk.setIndex('size', '2', 0);
blk.setIndex('size', '3', 1);
```

This creates a block, then changes its width to 2 and its depth to 3.

Compared to above, the commands using a full notation are:

```
geom.feature.create('blk1', 'Block');
geom.feature('blk1').setIndex('size', '2', 0);
geom.feature('blk1').setIndex('size', '3', 1);
```

When a model object is saved in the M-file format, the full notation is always used.

Note: `'blk1'` is the block geometry tag defined inside the COMSOL model object. The variable `blk`, defined only in MATLAB, is the link to the block feature. By linking MATLAB variables to features, you can directly access and modify features of a COMSOL model object in an efficient manner.

Wrapper Functions

LiveLink™ *for* MATLAB® comes with a suite of functions to help you modify the model object. These functions are mainly for retrieving information and data from a model, but there are also functions for updating data. The wrapper functions are meant as an aid in working with the COMSOL API and cover the most common scenarios where MATLAB is used to work with COMSOL models.

To get the complete list of the wrapper function, once you have started COMSOL with MATLAB, enter:

```
help mli
```

When working with wrapper functions, you benefit from autocompletion to enter valid arguments and properties.

For example, to load a model and show a plot in MATLAB you can do:

```
mphopen busbar
mphplot(model,'pg1')
```

In order to observe how autocompletion works, you can write

```
mphplot(
```

and press the Tab key. Then the model variable (usually `model`) will appear. If you then enter a comma and press the Tab key again, the plot group tags in the model will appear. You can use Tab completion to further enter property names and values where applicable.

About The Model Object

The model object contains all the information about a model, from the geometry to the results. The model object is defined on the COMSOL Multiphysics server and can be accessed at the MATLAB command line using a link in MATLAB.

1 Start COMSOL with MATLAB

2 Start modeling by creating a model object on the COMSOL Multiphysics server. This is done by entering the following command on the MATLAB command line:

```
model = ModelUtil.create('Model');
```

The model object on the COMSOL Multiphysics server has the tag `Model`, while the variable `model` is its link in MATLAB.

To access and modify the model object, use the COMSOL API syntax. You can get the documentation of a specific API command with the function `mphdoc`.

3 To get the documentation of the node model from the *COMSOL Programming Reference Manual*, enter:

```
mphdoc(model)
```

4 To get the documentation of the geometry feature WorkPlane, enter:

```
mphdoc(model.geom,'WorkPlane')
```

Connecting the Model in the COMSOL Desktop®

It is possible to connect the model you are working with from the MATLAB prompt to a COMSOL Desktop. This is a convenient way to monitor the modifications that are being made to the model object.

1 At the MATLAB prompt, enter:

```
mphlaunch
```

The model is now accessible from both the MATLAB prompt and the COMSOL Desktop. Every command entered at the prompt updates the model on the COMSOL Multiphysics server and in the COMSOL Desktop. At this stage there

is nothing in the model but later as you are following the instruction you will see how the model is populated during the modeling process.

`mphlaunch` works without arguments if there is only one model loaded on the server. If there is more than one model loaded on the server, you need to specify the tag of the model when using `mphlaunch` or the command will return an error and a list of the loaded models. You can use the command `mphtags -show` to get a list of loaded models before using `mphlaunch`.

About Global Parameters

If you plan to solve your model for several different parameter values, it is convenient to define them in the Parameters node and make use of the parametric sweep functionality in COMSOL Multiphysics. Global parameters can be used in expressions during all stages of model setup, like creating geometry, applying physics, or defining the mesh.

1 At the MATLAB prompt, define the parameters for the busbar model:

```
mphsetparam(model, 'L', '9[cm]', 'Length of the busbar');
mphsetparam(model, 'rad_1', '6[mm]', 'Radius of the fillet');
mphsetparam(model, 'tbb', '5[mm]', 'Thickness');
mphsetparam(model, 'wbb', '5[cm]', 'Width');
mphsetparam(model, 'mh', '6[mm]', 'Maximum element size');
mphsetparam(model, 'htc', '5[W/m^2/K]', 'Heat transfer coefficient');
mphsetparam(model, 'Vtot', '20[mV]', 'Applied electric potential');
```

Geometry

2 You need first to create a component for this model:

```
comp1 = model.component.create('comp1', true);
```

3 In this component, create a 3D geometry node:

```
geom1 = comp1.geom.create('geom1', 3);
```

The create method of the geometry node requires, as input, a tag for the geometry name (`'geom1'`) and the geometry space dimension (`3` for 3D).

4 The initial geometry is obtained by extruding a 2D drawing. Create a work plane tagged `'wp1'`, and set it as the *xz*-plane of the 3D geometry:

```
wp1 = geom1.feature.create('wp1', 'WorkPlane');
wp1.set('quickplane', 'xz');
```

5 In this work plane, create a rectangle and set the width to `L+2*tbb` and the height to `0.1`:

```
r1 = wp1.geom.feature.create('r1', 'Rectangle');
r1.set('size', {'L+2*tbb' '0.1'});
```

Note: When the `size` properties of the rectangle are set as string values (using single quotes `''`), it indicates that the variables `L` and `tbb` are defined within the model object. In this case, they are defined in the Parameters node.

6 Create a second rectangle and set the width to `L+tbb` and the height to `0.1-tbb`. Then change the rectangle position to `(0;tbb)`:

```
r2 = wp1.geom.feature.create('r2', 'Rectangle');
r2.set('size', {'L+tbb' '0.1-tbb'});
r2.set('pos', {'0' 'tbb'});
```

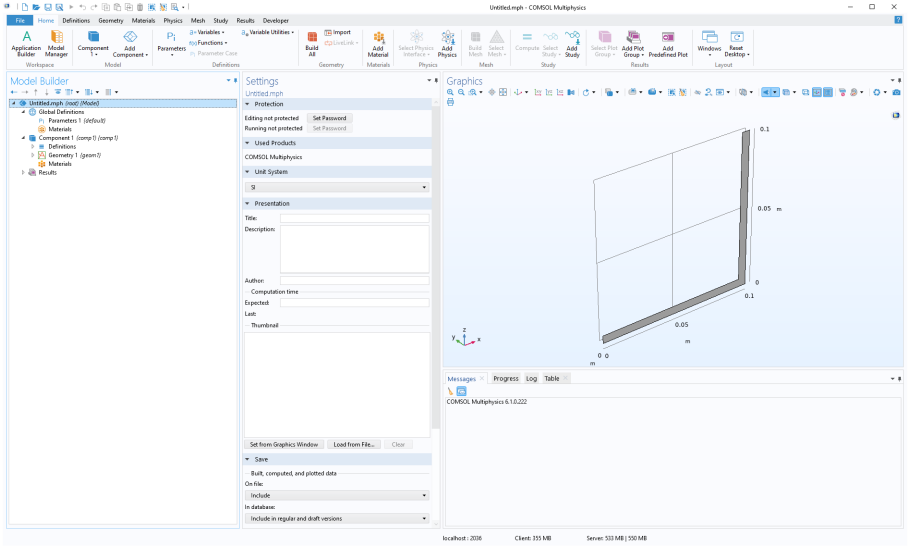
7 Subtract rectangle `r2` from rectangle `r1`, by creating a `Difference` feature with the '`input`' property set to `r1` and the '`input2`' property set to `r2`:

```
dif = wp1.geom.feature.create('dif', 'Difference');
dif.selection('input').set({'r1'});
dif.selection('input2').set({'r2'});
```

To display the current geometry in the COMSOL Desktop, you need to build the geometry node. At the MATLAB prompt, enter:

```
mphrun(model,'geom');
```

In the COMSOL Desktop, you can visualize the current geometry built at the MATLAB prompt.



8 Round the inner corner by creating a `Fillet` feature and set point `3` in the `selection` property. Then set the `radius` to `tbb`:

```
fil1 = wp1.geom.feature.create('fil1', 'Fillet');
```

```
fil1.selection('point').set('dif(1)', 3);
fil1.set('radius', 'tbb');
```

9 Round the outer corner by creating a new `Fillet` feature, then select point `6` and set the `radius` to `2*tbb`:

```
fil2 = wp1.geom.feature.create('fil2', 'Fillet');
fil2.selection('point').set('fil1(1)', 6);
fil2.set('radius', '2*tbb');
```

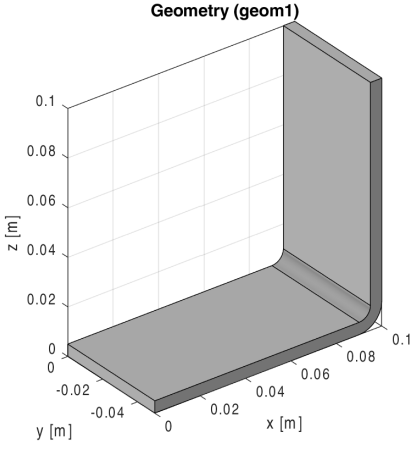
For geometry operations, the names of the resulting geometry objects are formed by appending a numeral within parentheses to the tag of the geometry operation. Above, `'fil1(1)'` is the result of the `'fil1'` operation.

10 Extrude the geometry objects in the work plane. Create an `Extrude` feature, set the work plane `wp1` as `input` and the `distance` to `wbb`:

```
ext1 = geom1.feature.create('ext1', 'Extrude');
ext1.selection('input').set({'wp1'});
ext1.set('distance', {'wbb'});
```

To plot the current geometry in a MATLAB figure, enter:

```
mphgeom(model)
```



Note that `mphgeom` automatically builds the geometry node and updates the COMSOL Desktop.

The busbar shape is now generated. Next, create the cylinders that represent the bolts connecting the busbar to the external frame (not represented in the model).

1 Create a new work plane and set the `planetype` property to `faceparallel`. Then set the `selection` to boundary `8`:

```
wp2 = geom1.feature.create('wp2', 'WorkPlane');
wp2.set('planetype', 'faceparallel');
wp2.selection('face').set('ext1(1)', 8);
```

2 Create a circle and set the radius to `rad_1`:

```
c1 = wp2.geom.feature.create('c1', 'Circle');
c1.set('r', 'rad_1');
```

3 Create an `Extrude` node, select the second work plane `wp2` as `input`, and then set the extrusion distance to `-2*tbb`:

```
ext2 = geom1.feature.create('ext2', 'Extrude');
ext2.selection('input').set({'wp2'});
ext2.set('distance', {'-2*tbb'});
```

4 Create a new workplane, set `planetype` to `faceparallel`, then set the selection to boundary `4`:

```
wp3 = geom1.feature.create('wp3', 'WorkPlane');
wp3.set('planetype', 'faceparallel');
wp3.selection('face').set('ext1(1)', 4);
```

5 Create a circle, then set the radius to `rad_1` and set the position of the center to `(-L/2+1.5e-2;-wbb/4)`:

```
c2 = wp3.geom.feature.create('c2', 'Circle');
c2.set('r', 'rad_1');
c2.set('pos', {'-L/2+1.5e-2' '-wbb/4'});
```

6 Create a second circle in the work plane by copying the previous circle and displacing it a distance `wbb/2` in the *y* direction:

```
copy = wp3.geom.feature.create('copy', 'Copy');
copy.selection('input').set({'c2'});
copy.set('disply', 'wbb/2');
```

7 Extrude the circles `c2` and `copy1` from the work plane `wp3` a distance `-2*tbb`:

```
ext3 = geom1.feature.create('ext3', 'Extrude');
ext3.selection('input').set({'wp3.c2' 'wp3.copy'});
ext3.set('distance', {'-2*tbb'});
```

8 Build the entire geometry sequence, including the finalize node using the `run` method:

```
mphrun(model,'geom');
```

Note: `mphrun` only builds features which need rebuilding or have not yet been built, including the finalize node.

This ends the geometry building. In the COMSOL Desktop, you can now see the final model geometry.

Selections

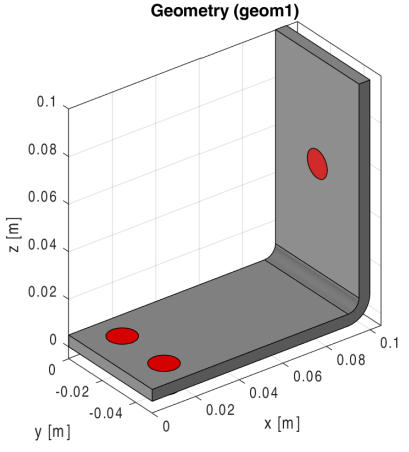
You can create selections of geometric entities such as domains, boundaries, edges, or points. These selections can be accessed during the modeling process with the advantage of not having to select the same entities several times.

1 To create a domain selection corresponding to the titanium bolts, named `Ti bolts`, enter:

```
sel1 = comp1.selection.create('sel1');
sel1.set([2 3 4 5 6 7]);
sel1.label('Ti bolts');
```

2 To visualize the selection in a MATLAB figure, enter:

```
mphviewselection(model,'sel1');
```



Material Properties

The Material node of the model object stores the material properties. In this example, the Joule Heating interface is used to include both an electric current and a heat balance. Thus, the electrical conductivity, heat capacity, relative permittivity, density, and thermal conductivity of the materials all need to be defined.

The busbar is made of copper and the bolts are made of titanium. The properties you need for these two materials are listed in the table below:

PROPERTY	COPPER	TITANIUM
Electrical conductivity	5.998e7[S/m]	7.407e5[S/m]
Heat capacity	385[J/(kg*K)]	710[J/(kg*K)]
Relative permittivity	1	1
Density	8700[kg/m^3]	4940[kg/m^3]
Thermal conductivity	400[W/(m*K)]	7.5[W/(m*K)]

To define material properties using the COMSOL API, you can either create the material from scratch or import the material data from another model MPH-file. In this example you will first create the Copper material node from scratch and then import the Titanium material data from an existing model from the Application library.

1 Create the first material, copper:

```
mat1 = comp1.material.create('mat1');
```

2 Set the properties for the 'electricconductivity', 'heatcapacity', 'relpermittivity', 'density' and 'thermalconductivity' according to the table above:

```
mat1.materialModel('def').set('electricconductivity', {'5.998e7[S/m]'});
mat1.materialModel('def').set('heatcapacity', '385[J/(kg*K)]');
mat1.materialModel('def').set('relpermittivity', {'1'});
mat1.materialModel('def').set('density', '8700[kg/m^3]');
mat1.materialModel('def').set('thermalconductivity', {'400[W/(m*K)]'});
```

3 Set the name of the material to 'Copper'. By default the first material is assigned to all domains:

```
mat1.label('Copper');
```

Now, import Titanium material data from the busbar model MPH-file, which is available in the COMSOL Multiphysics Application Library.

4 Get the list of the material nodes defined in the model busbar.mph:

```
modelRef = '<COMSOL_path>\applications\COMSOL_Multiphysics\
    Multiphysics\busbar.mph';
ModelUtil.scanModel(modelRef,'Material','op')
```

where *<COMSOL_path>* is the path of your COMSOL installation root directory. From the output, one can see that the material node contains two materials. The second material mat2 is the one defining Titanium beta-21S data and that is the one you will import in the model in the steps below.

5 Import material `mat2` from the model `busbar.mph` to the material node of the current model:

```
comp1.material.insert(modelRef,'mat2','');
```

6 Assign the newly created material to the previously created selection `'sel1'` (corresponding to the bolt domains):

```
comp1.material('mat2').selection.named('sel1');
```

Only one material per domain can be assigned. This means that the last operation automatically removes the bolts from the selection of the copper material.

Physics Interface

The Physics node contains the settings of the physics interfaces, including the domain and the boundary settings. Settings are grouped together according to the physics interface they belong to. To model the electrothermal interaction of this example, add the `ConductiveMedia` and `HeatTransfer` interfaces to the model. Apply a fixed electric potential to the upper bolt and ground the two lower bolts. In addition, assume that the device is cooled by convection, approximated by a heat flux with a defined heat transfer coefficient on all outer faces, except where the bolts are connected.

1 Create the `HeatTransfer` interface on the `geom1` geometry:

```
ht = comp1.physics.create('ht', 'HeatTransfer', 'geom1');
```

2 Add a heat flux boundary condition to the physics interface and set the type to `InwardHeatFlux`:

```
hf1 = ht.feature.create('hf1', 'HeatFluxBoundary', 2);
hf1.set('HeatFluxType', 'InwardHeatFlux');
```

Note: The third argument of the create method, the space dimension `sdim`, indicates which geometry level (domain, boundary, edge, or point) the feature should be applied to. In the above commands, the `'HeatFluxBoundary'` feature applies to boundaries, which have the space dimension 2.

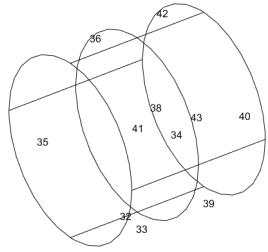
3 Now apply cooling to all exterior boundaries 1–43, except the bolt connection boundaries 8, 15, and 43. An `InwardHeatFlux` boundary condition requires a heat transfer coefficient. Set its value to the previously defined parameter `htc`:

```
hf1.selection.set([1:7 9:14 16:42]);
hf1.set('h', 'htc');
```

4 As you may have noticed, defining selections requires you to know the entity indices. To view the boundary indices, display the geometry with face labels:

```
mphgeom(model,'geom1','facemode','off','facelabels','on')
```

You can use the controls in the window to zoom and pan to read off the indices.



Note: Alternative methods for obtaining entity indices are described in the *LiveLink™ for MATLAB® User's Guide*. These include the use of point coordinates, selection boxes, or adjacency information.

5 Create the `ConductiveMedia` interface on the `geom1` geometry:

```
ec = comp1.physics.create('ec', 'ConductiveMedia', 'geom1');
```

6 Now create an electric potential boundary condition and set the selection to boundary `43`; then set the electric potential to `Vtot`:

```
pot1 = ec.feature.create('pot1', 'ElectricPotential', 2);
pot1.selection.set(43);
pot1.set('V0', 'Vtot');
```

7 Apply a ground boundary condition to boundaries `8` and `15`:

```
gnd1 = ec.feature.create('gnd1', 'Ground', 2);
gnd1.selection.set([8 15]);
```

The model object includes default properties so you do not need to set properties for all boundaries. For example, the Conductive Media interface uses a current insulation as a default boundary condition for the current balance.

Multiphysics Interface

The Multiphysics Couplings node contains the possible couplings between the physics interfaces used in the model. To model the electrothermal interaction in this example, add the Electromagnetic Heating coupling to the model. This node is included by default when modeling electromagnetic heating in the COMSOL Desktop.

1 Create the `ElectromagneticHeating` coupling on the `comp1` component and set the selection to all domains:

```
comp1.multiphysics.create('emh','ElectromagneticHeating');
```

The Heat Transfer and the Conductive Media interfaces are automatically included in the coupling.

2 Set the Electromagnetic Heating multiphysics interface to all domains:

```
comp1.multiphysics('emh').selection.all;
```

Mesh

The mesh sequence is stored in the Mesh node. Several mesh sequences, also called mesh cases, can be created in the same model object.

1 Create a new `mesh` case for `comp1`:

```
mesh = comp1.mesh.create('mesh');
```

2 By default, the mesh sequence contains at least a `size` feature, which applies to all subsequent meshing operations. First create a link to the existing `size` feature. Then set the maximum element size `hmax` to `mh` and the minimum element size `hmin` to `mh-mh/3`. Set the curvature factor `hcurve` to `0.2`:

```
size = mesh.feature('size');
size.set('hmax', 'mh');
size.set('hmin', 'mh-mh/3');
size.set('hcurve', '0.2');
```

3 Add a mesh feature that generates the tetrahedral mesh:

```
mesh.feature.create('ftet', 'FreeTet');
```

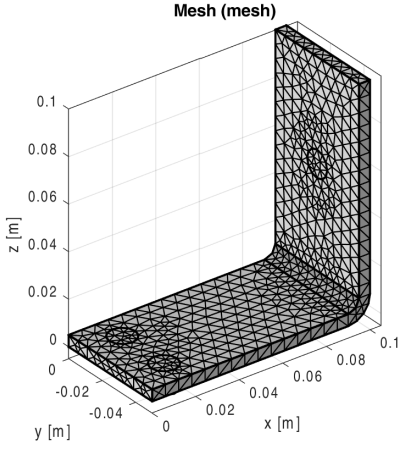
4 Build the mesh:

```
mphrun(model,'mesh');
```

Now look in the COMSOL Desktop to see the resulting mesh in the graphics window.

5 To visualize the mesh in a MATLAB figure, use the command `mphmesh`:

```
mphmesh(model)
```



Study

In order to solve the model, you need to create a Study node where the analysis type is set for the solver.

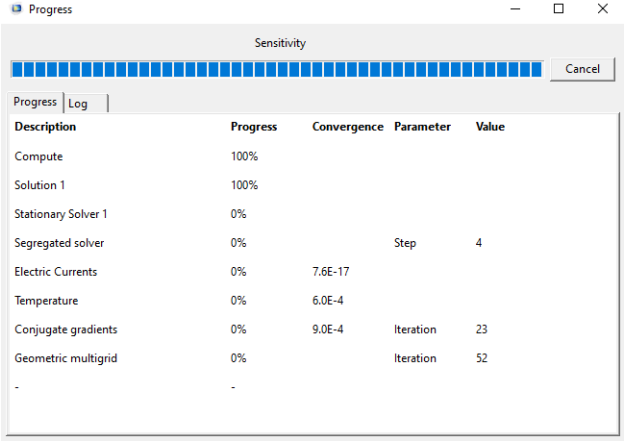
1 Create a `study` node and add a stationary study step:

```
std = model.study.create('std');
std.feature.create('stat', 'Stationary');
```

2 Solve the model, run the study with the command:

```
mphrun(model,'study');
```

During the computation, a window opens to display the progress information and solver log.

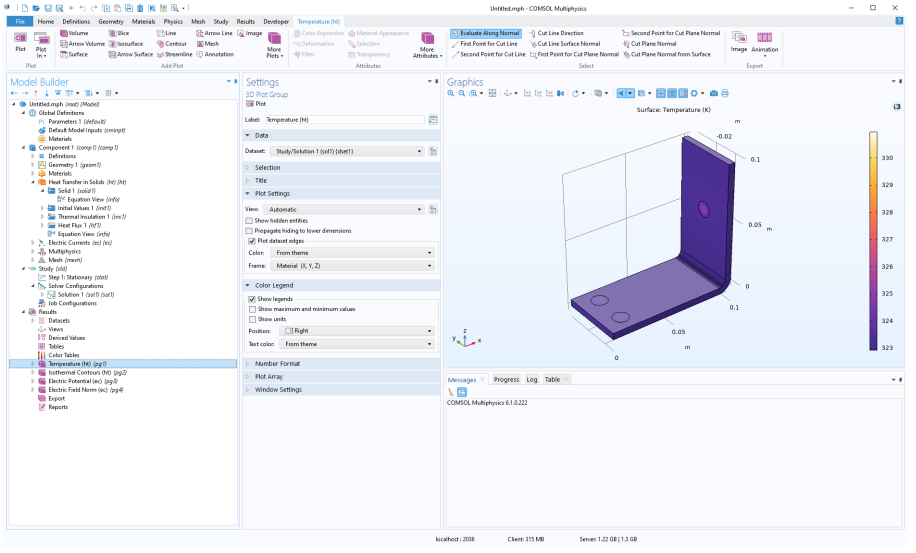


Note: The progress bar is not available on macOS.

In this example, no solver related settings were necessary since the study node automatically built the solver sequence based on the study type, physics interface, and the space dimension of the model.

Plotting The Results

One way to analyze the results is to plot the expression of interest. The first time you run the study, default plots are created, which you can visualize in the COMSOL Desktop.



The first plot group Temperature (ht) shows the temperature distribution, you may notice that the maximum temperature region is located in the bolt. To get a better rendering of the temperature distribution within the busbar, you need to edit the plot group.

1 To manipulate a plot group you need to know their tag. Use `mphtags` to get the list of the plot group tags available under the result node, and then the tag of the plot feature in the first plot group:

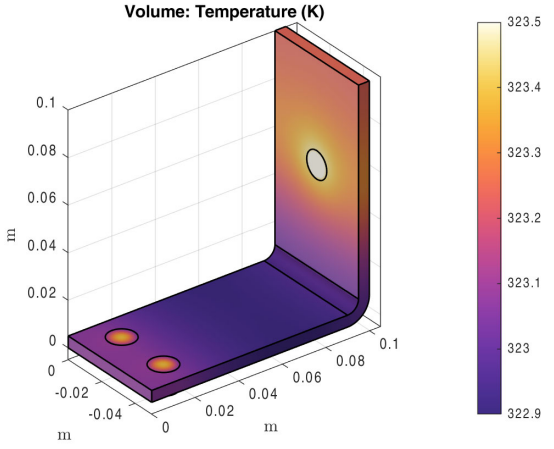
```
tags = mphtags(model, 'result');
feattags = mphtags(model.result(tags{1}));
```

2 Activate a manual color range; set the minimum color value to `322.9` and set the maximum color value to `323.5`:

```
surf = model.result(tags{1}).feature(feattags{1})
surf.set('rangecoloractive', 'on');
surf.set('rangecolormin', '322.9');
surf.set('rangecolormax', '323.5');
```

3 Display the first plot group, including a color bar, use the `mphplot` command:

```
mphplot(model, tags{1}, 'rangenum', 1)
```



Note: If several plot types are available in the same plot group, you can define which color bar to display. The value of the `rangenum` property corresponds to the plot type number in the plot group sequence.

Exporting Results

With LiveLink™ *for* MATLAB®, you can export data either to a text file by using the Export feature available in the model, or directly to the MATLAB workspace by using the COMSOL function suite available in MATLAB. See Extracting Results at the MATLAB® Command Line for more details.

Results can be exported to a text file by using the Export node.

1 Create a data export node for the temperature variable `T`:

```
data = model.result.export.create('data', 'Data');
data.setIndex('expr', 'T', 0);
```

2 Set the filename to `<filepath>\Temperature.txt` where `<filepath>` is replaced with the path to the directory where the file should be saved:

```
data.set('filename','<filepath>\Temperature.txt');
```

3 Perform the export operation:

```
mphrun(model,'data');
```

The above steps extract the temperature at each computational node of the geometry and store the data in a text file in the following format:

```
% Model:              Untitled.mph
```

```
% Version:            COMSOL 6.2.0.250
% Date:               Nov 1 2023, 15:08
% Dimension:          3
% Nodes:              1351
% Expressions:        1
% Description:        Temperature
% Length unit:        m
% X             Y               Z               T (K)
0.095           -0.02200000     0.06019615      323.47243290
0.095           -0.01803113     0.06159800      323.43024700
0.0975          -0.02023335     0.06305422      323.44003680
0.09767380      -0.01977740     0.05795373      323.49327862
```

Saving the Model

1 To save the model MPH-file with the `mphsave` method, enter:

```
mphsave(model,'<path>/busbar');
```

In the above command, replace `<path>` with the directory where you would like to save your model. If a path is not defined, the model is saved in the current working directory. You can use the command `pwd` to get the current working directory in MATLAB.

The default save format is the COMSOL binary format with the `mph` extension. To save the model as an M-file use the command:

```
mphsave(model,'<path>/busbar.m');
```

Note: `mphsave` only save model to file. To save a model in Model Manager use the command `mmsave`.

2 In case you have created a database using Model Manager, first get the branch address where you would like to save your model:

```
branch = mmgetbranch('<branchname>','<repositoryname>','<databasename>');
```

where `'<branchname>'`, `'<repositoryname>'`, `'<databasename>'` are the name of the branch, the repository, and the database where to save the model respectively.

3 To save the model as a version at `branch`, enter:

```
mmsave(model,'version','Commit message for the busbar model',branch);
```

Code for Use with MATLAB®

```
model = ModelUtil.create('Model');
mphsetparam(model, 'L', '9[cm]', 'Length of the busbar');
mphsetparam(model, 'rad_1', '6[mm]', 'Radius of the fillet');
mphsetparam(model, 'tbb', '5[mm]', 'Thickness');
mphsetparam(model, 'wbb', '5[cm]', 'Width');
mphsetparam(model, 'mh', '6[mm]', 'Maximum element size');
```

```
mphsetparam(model, 'htc', '5[W/m^2/K]', 'Heat transfer coefficient');
mphsetparam(model, 'Vtot', '20[mV]', 'Applied electric potential');
comp1 = model.component.create('comp1', true);
geom1 = comp1.geom.create('geom1', 3);
wp1 = geom1.feature.create('wp1', 'WorkPlane');
wp1.set('quickplane', 'xz');
r1 = wp1.geom.feature.create('r1', 'Rectangle');
r1.set('size', {'L+2*tbb' '0.1'});
r2 = wp1.geom.feature.create('r2', 'Rectangle');
r2.set('size', {'L+tbb' '0.1-tbb'});
r2.set('pos', {'0' 'tbb'});
dif = wp1.geom.feature.create('dif', 'Difference');
dif.selection('input').set({'r1'});
dif.selection('input2').set({'r2'});
mphrun(model,'geom');
fil1 = wp1.geom.feature.create('fil1', 'Fillet');
fil1.selection('point').set('dif(1)', 3);
fil1.set('radius', 'tbb');
fil2 = wp1.geom.feature.create('fil2', 'Fillet');
fil2.selection('point').set('fil1(1)', 6);
fil2.set('radius', '2*tbb');
ext1 = geom1.feature.create('ext1', 'Extrude');
ext1.selection('input').set({'wp1'});
ext1.set('distance', {'wbb'});
mphgeom(model)
wp2 = geom1.feature.create('wp2', 'WorkPlane');
wp2.set('planetype', 'faceparallel');
wp2.selection('face').set('ext1(1)', 8);
c1 = wp2.geom.feature.create('c1', 'Circle');
c1.set('r', 'rad_1');
ext2 = geom1.feature.create('ext2', 'Extrude');
ext2.selection('input').set({'wp2'});
ext2.set('distance', {'-2*tbb'});
wp3 = geom1.feature.create('wp3', 'WorkPlane');
wp3.set('planetype', 'faceparallel');
wp3.selection('face').set('ext1(1)', 4);
c2 = wp3.geom.feature.create('c2', 'Circle');
c2.set('r', 'rad_1');
c2.set('pos', {'-L/2+1.5e-2' '-wbb/4'});
copy = wp3.geom.feature.create('copy', 'Copy');
copy.selection('input').set({'c2'});
copy.set('disply', 'wbb/2');
ext3 = geom1.feature.create('ext3', 'Extrude');
ext3.selection('input').set({'wp3.c2' 'wp3.copy'});
ext3.set('distance', {'-2*tbb'});
mphrun(model,'geom');
sel1 = comp1.selection.create('sel1');
sel1.set([2 3 4 5 6 7]);
sel1.label('Ti bolts');
mphviewselection(model,'sel1');
mat1 = comp1.material.create('mat1');
mat1.materialModel('def').set('electricconductivity', {'5.998e7[S/m]'});
mat1.materialModel('def').set('heatcapacity','385[J/(kg*K)]');
```

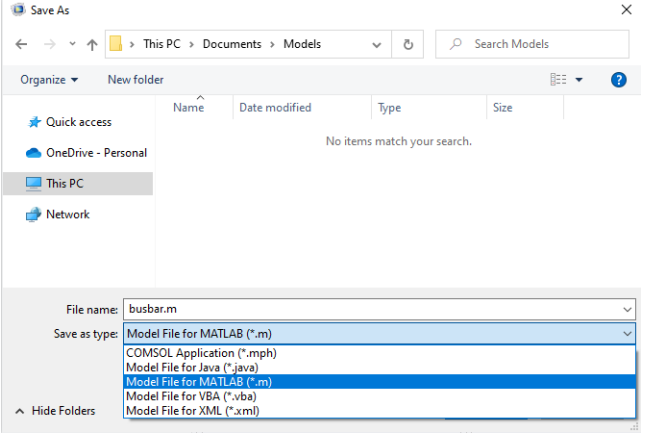
```
mat1.materialModel('def').set('relpermittivity', {'1'});
mat1.materialModel('def').set('density','8700[kg/m^3]');
mat1.materialModel('def').set('thermalconductivity',{'400[W/(m*K)]'});
mat1.label('Copper');
ModelUtil.scanModel('<COMSOL_path>\applications\COMSOL_Multiphysics\
Multiphysics\busbar.mph', 'Material','op')
comp1.material.insert('<COMSOL_path>\applications\COMSOL_Multiphysics\
Multiphysics\busbar.mph', 'mat2', '');
comp1.material('mat2').selection.named('sel1');
ht = comp1.physics.create('ht', 'HeatTransfer', 'geom1');
hf1 = ht.feature.create('hf1', 'HeatFluxBoundary', 2);
hf1.set('HeatFluxType', 'InwardHeatFlux');
hf1.selection.set([1:7 9:14 16:42]);
hf1.set('h', 'htc');
mphgeom(model,'geom1','facemode','off','facelabels','on')
ec = comp1.physics.create('ec','ConductiveMedia', 'geom1');
pot1 = ec.feature.create('pot1','ElectricPotential', 2);
pot1.selection.set(43);
pot1.set('V0', 'Vtot');
gnd1 = ec.feature.create('gnd1', 'Ground', 2);
gnd1.selection.set([8 15]);
comp1.multiphysics.create('emh','ElectromagneticHeatSource');
comp1.multiphysics('emh').selection.all;
mesh = comp1.mesh.create('mesh');
size = mesh.feature('size');
size.set('hmax', 'mh');
size.set('hmin', 'mh-mh/3');
size.set('hcurve', '0.2');
mesh.feature.create('ftet', 'FreeTet');
mphrun(model, 'mesh');
mphmesh(model)
std = model.study.create('std');
std.feature.create('stat', 'Stationary');
mphrun(model, 'study');
tags = mphtags(model,'result');
feattags = mphtags(model.result(tags{1}));
surf = model.result(tags{1}).feature(feattags{1});
surf.set('rangecoloractive', 'on');
surf.set('rangecolormin', '322.9');
surf.set('rangecolormax', '323.5');
mphplot(model, tags{1}, 'rangenum', 1)
data = model.result.export.create('data', 'Data');
data.setIndex('expr', 'T', 0);
data.set('filename','<filepath>\Temperature.txt');
mphrun(model, 'data');
mphsave(model,'<path>\busbar.m');
```

Saving and Running a Model M-File

The easiest way to learn the commands of the COMSOL API is to save a model from the COMSOL Desktop as an M-file. The M-file contains the sequence of commands that create the model. You can open the M-file in a text editor and make modifications to it, as shown in the following example.

Note: By default, the M-file contains the entire command history of the model following the operation done to the model. This means that the model M-file can include settings that are no longer part of the model. In order to include only settings that are part of the current model, compact the model history before saving the M-file. If you want to get the specific command for the last operation that was performed, do not compact the model history.

1. Start COMSOL with MATLAB and a COMSOL Desktop.
2. In the COMSOL Desktop, go to the Application Libraries window.
3. In the Application Libraries window, expand the COMSOL Multiphysics folder and then the Multiphysics folder. Select the busbar model and click Open.
4. Compact the model history in order to get only the commands corresponding to the current state of the model. In the COMSOL Desktop, choose File>Compact History ( ).
5. Go to File>Save As. In the Save window, locate the Save as type list and select Model file for MATLAB (*.m). Name the file `busbar` and choose the directory to save it to. Click OK.



6. Open the saved M-file with the MATLAB editor or any text editor and search for the lines below:

```
model.result('pg4').feature('surf1').set('expr', 'ec.normJ');
```

```
model.result('pg4').feature('surf1').set('descr', 'Current density norm');
model.result('pg4').feature('surf1').set('rangecolormax', '1e6');
```

These commands define the plot group `pg4`, which includes a surface plot of the expression `ec.normJ` with the maximum color range set to `1e6`.

7 To modify the plot group `pg4` so that it displays the total heat `jh.Qtot` with the maximum color range set to `1e4`, add the following lines to the M-file, just before the last command (output of the function):

```
model.result('pg4').feature('surf1').set('expr','ht.Qtot');
model.result('pg4').feature('surf1').set('descr', 'Total heat source');
model.result('pg4').feature('surf1').set('rangecolormax','1e4');
```

These commands modify plot group `pg4` so that it displays the total heat `ht.Qtot` with a new setting for the maximum value for the color range. You can also directly modify the original lines corresponding to the plot settings to achieve the same thing, but this way you can easily revert to the old version.

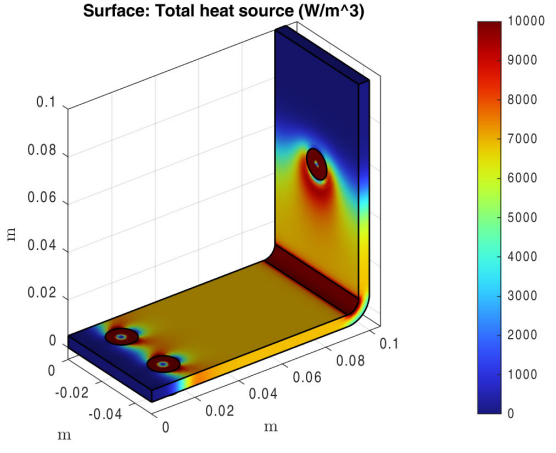
8 Save the modified M-file.

9 Run the model at the MATLAB prompt with the command:

```
model = busbar;
```

10 Display the plot group `pg4`:

```
mphplot(model,'pg4')
```



Another way to modify a model object in MATLAB is to load the model MPH-file in MATLAB, and then enter the commands in step 7 directly at the MATLAB command line. The benefit of this latter approach is that it does not require running the entire model. See An Example Using Nested Loops, where this technique is applied.

Extracting Results at the MATLAB® Command Line

LiveLink™ *for* MATLAB® packages several functions to make it easy to access results directly from the MATLAB command line. The most commonly used functions are:

- `mphinterp`, to evaluate expressions at arbitrary location.
- `mpheval`, to evaluate expressions on all node points of a given domain selection.
- `mphglobal`, to evaluate global expressions.
- `mphint2`, to integrate the value of expressions on selected domains.
- `mphmax`, `mphmin`, and `mphmean`, to evaluate the maximum, minimum, and average, respectively, of an expression.
- `mphgetexpressions`, to get the model variables and model parameters expressions.
- `mphevaluate`, to evaluate parameter expressions (constants) in models.

Note: For a complete list of the available functions see the section Summary of Commands in the *LiveLink for MATLAB User's Guide*.

Load the `busbar` model from the application library directly at the MATLAB command line.

1 Start COMSOL with MATLAB.

2 Load the busbar example model from the application library. Enter the command:

```
mphopen busbar
```

The `mphopen` command automatically searches the MATLAB path as well as the path for the application libraries to find the requested file. You can also specify the path in the filename. When specifying the mphopen command on the MATLAB command line you can enter `mphopen` and space. Then press the TAB-key to get a list of the MPH-files in the current directory.

Evaluating Data at Arbitrary Points

If you want to get data at a specific location not necessarily defined by node points, use the command `mphinterp` to interpolate the results. The interpolation method uses the shape function of the elements.

1 To extract the total heat source at `[5e-2;-2e-2;1e-3]` enter:

```
Qtot = mphinterp(model,'ht.Qtot','coord',[5e-2;-2e-2;1e-3])
```

This returns:

```
Qtot =
  6.9417e+003
```

2 You can specify the unit for evaluation, for instance extract the temperature in Fahrenheit with the command:

```
[Temp, unit]= mphinterp(model,'T','coord',[5e-2;-2e-2;1e-3],'unit','degF')
```

This returns:

```
Temp =
  120.6804

unit =
  'degF'
```

3 A grid of points can also be evaluated. Enter the following lines:

```
x0=0:5e-2/4:5e-2;
y0=0:-2.5e-2/4:-2.5e-2;
z0=[0 5e-3];
[x,y,z]=meshgrid(x0,y0,z0);
xx=[x(:),y(:),z(:)]';
Qtot = mphinterp(model,'ht.Qtot','coord',xx);
```

4 Reshape the variable `Qtot` for a better visualization:

```
Qtot = reshape(Qtot,length(x0),length(y0),length(z0))
```

This results in:

```
Qtot(:,:,1) =
1.0e+05 *
    0.0000    0.0278    0.0739    0.0697    0.0694
    0.0007    0.3955    0.0736    0.0694    0.0694
    0.0000    3.1413    0.0672    0.0692    0.0694
    0.0007    0.2908    0.0736    0.0694    0.0694
    0.0000    0.0278    0.0738    0.0697    0.0694

Qtot(:,:,2) =
1.0e+03 *
    0.0000    2.8296    7.3588    6.9707    6.9434
    0.0678    5.2452    7.0008    6.9426    6.9422
    0.0000    8.7617    5.5528    6.9160    6.9407
    0.0639    5.2439    7.0061    6.9413    6.9414
    0.0000    2.8245    7.3524    6.9672    6.9421
```

Evaluating Data at Node Points

Use the `mpheval` function to evaluate the electric potential, which is one of the dependent variables in the busbar model. `mpheval` returns the value of the

expression evaluated using the nodes of a simplex mesh. By default, the simplex mesh corresponds to the active mesh in the model object.

1 Enter at the command line:

```
data = mpheval(model,'V')
```

A MATLAB structure is returned according to:

```
data =
    expr: {'V'}
      d1: [1x7334 double]
       p: [3x7334 double]
       t: [4x30916 int32]
      ve: [7334x1 int32]
    unit: {'V'}
```

`mpheval` returns a MATLAB structure defined with the following fields:

- `expr` contains the list of the evaluated expression,
- `d1` contains the data of the evaluated expression,
- `p` contains the coordinates of the evaluation points,
- `t` contains the indices to columns in the `p` field. Each column in the `t` field corresponds to an element of the mesh used for the evaluation,
- `ve` contains the indices of the mesh elements at each evaluation point,
- `unit` contains the unit of the evaluated expression.

2 Multiple variables can be evaluated at once, for example, both the temperature and the electric potential. Enter:

```
data2 = mpheval(model,{'T' 'V'})
```

This command returns this structure:

```
data2 =
    expr: {'T'  'V'}
      d1: [1x7334 double]
      d2: [1x7334 double]
       p: [3x7334 double]
       t: [4x30916 int32]
      ve: [7334x1 int32]
    unit: {'K'  'V'}
```

The fields `data2.d1` and `data2.d2` contain the values for the temperature and electric potential, respectively.

3 Now evaluate the temperature `T`, including at the element midpoints as defined by a quadratic shape function. Use the `refine` property and set the property value to `2`, which corresponds to the discretization order:

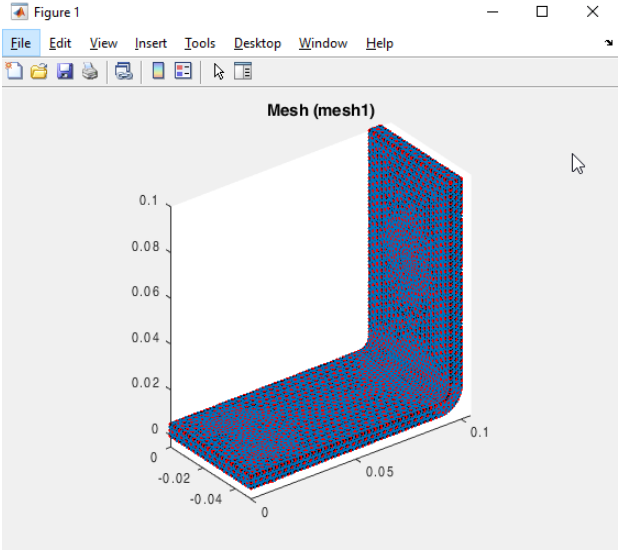
```
data3 = mpheval(model,'T','refine',2)
```

This command returns this structure:

```
data3 =
    expr: {'T'}
      d1: [1x49794 double]
       p: [3x49794 double]
       t: [4x247328 int32]
      ve: [49794x1 int32]
    unit: {'K'}
```

To visualize the evaluation location for both `data2` and `data3` together with the mesh, enter the commands:

```
mphmesh(model)
hold on;
plot3(data3.p(1,:),data3.p(2,:),data3.p(3,:),'.');
plot3(data2.p(1,:),data2.p(2,:),data2.p(3,:),'.r');
```



4 This step demonstrates how to use the `edim` property in combination with the `selection` property. `edim` allows you to determine the space dimension of the evaluation and `selection` allows you to set the entity number indicating where to evaluate the expression.

To evaluate data on a specific domain, such as the norm of the current density at boundary number 43, enter:

```
data4 = mpheval(model,'ec.normJ',...
     'edim','boundary','selection',43)
```

This command returns this structure:

```
data4 =
    expr: {'ec.normJ'}
      d1: [1x65 double]
```

```
      p: [3x65 double]
      t: [3x108 int32]
     ve: [65x1 int32]
   unit: {'A/m^2'}
```

Global Evaluation and Integration

To get the value of an expression defined with a global scope, use the function `mphglobal`.

1 Evaluate the total applied voltage, defined with the parameter Vtot:

```
Vtot = mphglobal(model,'Vtot')

Vtot =
  0.0200
```

2 To calculate the total heat produced in the busbar, you can integrate the total heat source using the command `mphint2`. Enter:

```
Q = mphint2(model,'ec.Qh','volume','selection',1)
```

The total heat is:

```
Q =
  0.2419
```

3 Evaluate the total current passing through boundary 43 and retrieve the unit of the integral:

```
[I,unit] = mphint2(model,'ec.normJ','surface','selection',43)

I =
  161.5116

unit =
  'A'
```

4 Evaluate the maximum of the temperature in the busbar:

```
maxT = mphmax(model,'T','volume','selection',1)

maxT =
  323.0784
```

5 Evaluate the maximum of the temperature on boundary 43:

```
maxT = mphmax(model,'T','surface','selection',43)

maxT =
  330.4062
```

Evaluating Expressions

If you want to evaluate the parameters, variables, or expressions in the model, use the function `mphgetexpressions`.

1 Evaluate the expressions of the parameters defined in the model:

```
expr = mphgetexpressions(model.param)
```

```
expr =

  {'L'    }  {'9[cm]'     }  {'Length'         }  {[0.0900]}  {'m'         }
  {'rad_1'}  {'6[mm]'     }  {'Bolt radius'    }  {[0.0060]}  {'m'         }
  {'tbb'  }  {'5[mm]'     }  {'Thickness'      }  {[0.0050]}  {'m'         }
  {'wbb'  }  {'5[cm]'     }  {'Width'          }  {[0.0500]}  {'m'         }
  {'mh'   }  {'6[mm]'     }  {'Maximum eleme…'}  {[0.0060]}  {'m'         }
  {'htc'  }  {'5[W/m^2/K]'}  {'Heat transfer…'}  {[     5]}  {'W/(m^2*K)'}
  {'Vtot' }  {'20[mV]'    }  {'Applied volta…'}  {[0.0200]}  {'V'         }
```

2 You can evaluate a specific parameter expression by using the function `mphevaluate`. To evaluate the length of the busbar, L, in inches:

```
L = mphevaluate(model,'L','in')
```

```
L =

  3.5433
```

Note: The evaluation does not require an existing solution dataset in the model.

Code for Use with MATLAB®

```
mphopen busbar
```

```
% Evaluating Data at Arbitrary Points
Qtot = mphinterp(model,'ht.Qtot','coord',[5e-2;-2e-2;1e-3])
[Temp, unit]= mphinterp(model,'T','coord',[5e-2;-2e-2;1e-3],'unit','degF')
x0=0:5e-2/4:5e-2;
y0=0:-2.5e-2/4:-2.5e-2;
z0=[0 5e-3];
[x,y,z]=meshgrid(x0,y0,z0);
xx=[x(:),y(:),z(:)]';
Qtot = mphinterp(model,'ht.Qtot','coord',xx);
Qtot = reshape(Qtot,length(x0),length(y0),length(z0))
```

```
% Evaluating Data at Node Points
data = mpheval(model,'V')
data2 = mpheval(model,{'T' 'V'})
data3 = mpheval(model,'T','refine',2)
hold on;
plot3(data3.p(1,:),data3.p(2,:),data3.p(3,:),'.');
plot3(data2.p(1,:),data2.p(2,:),data2.p(3,:),'.r');
mphmesh(model)
```

```
data4 = mpheval(model,{'ec.normJ'},...
     'edim',2,'selection',43)
```

```
% Global Evaluation and Integration
```

```
Vtot = mphglobal(model,'Vtot')
Q = mphint2(model,'ec.Qh','volume','selection',1)
[I unit] = mphint2(model,'ec.normJ','surface','selection',43)
maxT = mphmax(model,'T','volume','selection',1)
maxT = mphmax(model,'T','surface','selection',43)
```

Automating with MATLAB® Scripts

LiveLink™ *for* MATLAB® enables you to combine the MATLAB programming tools with a COMSOL model object. A benefit of this integration is the ability to access results from the MATLAB workspace. Another benefit is that you can integrate COMSOL modeling commands into MATLAB scripts, taking full advantage of all available tools for controlling the flow of your code. This section explains how to do this efficiently, by introducing MATLAB variables into your COMSOL model and updating only the affected parts of your model object.

Obtaining Model Information

Modifying an existing model object requires you to know how the model is set up. Some functions help you to access this information so that you can add or remove a model feature, or modify the value of an existing property. The function `mphnavigator` helps you to retrieve model object information. Calling mphnavigator at the MATLAB prompt displays a summary of the features and their properties for the model object available in the MATLAB workspace.

1 Start COMSOL with MATLAB.

2 Load the busbar example model:

```
mphopen busbar
```

Alternatively, use this more advanced form of the command if you wish to use another variable name than `model`:

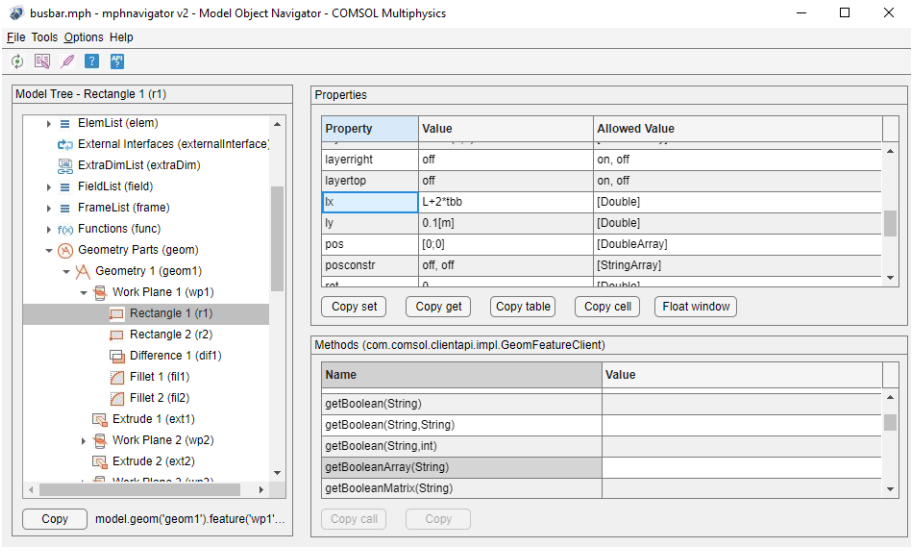
```
model = mphopen('busbar');
```

3 To access the model information, enter the following command at the MATLAB prompt:

```
mphnavigator
```

In the `mphnavigator` graphical user interface, the model nodes are listed under the Model Tree section. The nodes can be expanded to access the subnodes. When a node is selected in the Model Tree, its properties are listed in the Properties section. The Methods section lists the methods available for the selected node. Just above the Methods section, you can see the COMSOL API syntax that lead to the feature. The Copy button copies the API syntax of the command to the system clipboard.

4 In the Model Tree, expand the model nodes down to geom>geom1>wp1>r1 and select the node r1.



5 To get the command to access Rectangle 1 in MATLAB, under the Model Tree click Copy. Then select the MATLAB prompt and press CTRL+V keys.

6 In the Model Object Navigator, Properties section, you can see the that the lx property of the geometry feature r1 is set to `L+2*tbb`, which represents the length of the rectangle. The width of the rectangle, represented with the property ly, is set to `0.1`.

7 To get the command to set the value of a property, for instance lx, click on the property in the Model Object Navigator and then click Copy set.

8 Go back to MATLAB and press CTRL+V keys. In the newly pasted command, you can now change the value of the property by replacing the expression `L+2*tbb` by the desired value.

Updating Model Settings

The COMSOL model object allows you to modify parameters, features, or feature properties, using the `set` method. Once a property value is updated, you can run the appropriate sequence, depending on where the changes were introduced into the model — for example, in the geometry or the mesh sequence. Running the solver sequence automatically runs all sequences where modified settings are detected.

Starting with the busbar model described in A Thorough Example: The Busbar, this section modifies a parameter value and re-solves the model. For this purpose, the geometry of the model is prepared using parameters such as `L` for the length of the busbar, and `tbb` for the thickness of the busbar.

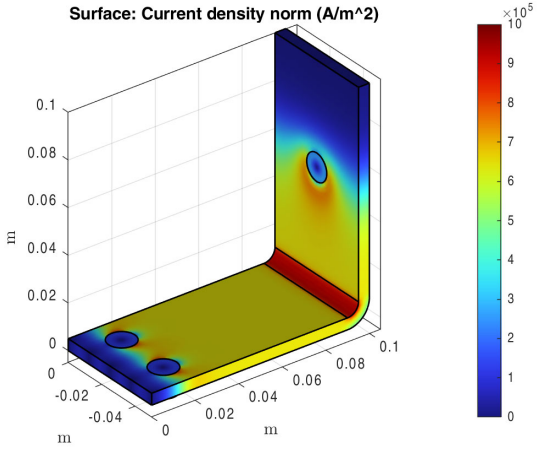
1 Start COMSOL with MATLAB (if you have not done so already).

2 Load the model from the COMSOL application library:

```
mphopen busbar
```

3 Plot the current solution:

```
mphplot(model,'pg4');
```



4 Change the parameter `L` (length of the busbar):

```
mphsetparam(model,'L','18[cm]');
```

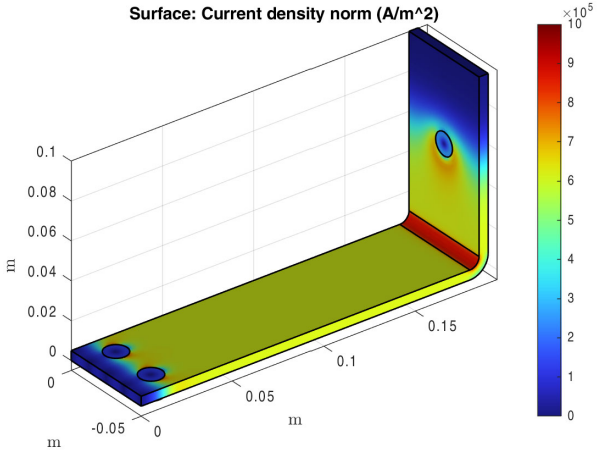
5 Run the solver sequence and compute the new solution:

```
mphrun(model,'study');
```

Note: The solver node automatically detects all modifications in the model, and runs the geometry or mesh sequences when needed. The new value for the parameter `L` induces a change in the geometry, requiring a new mesh. Full associativity is also ensured, making sure that all physics settings remain applied as in the original model.

6 Plot the new solution with the updated geometry:

```
mphplot(model,'pg4');
```



Code for Use with MATLAB®

```
mphopen busbar
mphplot(model,'pg4');
mphsetparam(model,'L','18[cm]');
mphrun(model,'study');
mphplot(model,'pg4');
```

Using MATLAB® Variables in the COMSOL Model

Use the `set` method described in the previous section to define the feature properties using MATLAB variables. It is important to make sure that the value of the MATLAB variable is consistent with the units used in the model. Before beginning, make sure that the busbar model used in the previous section is loaded into MATLAB.

First, change the length of the busbar, described by the parameter `L`.

1 Create a MATLAB variable `L0` equal to `9e-2`:

```
L0 = 9e-2;
```

2 Update the value of the parameter `L` with the variable `L0`:

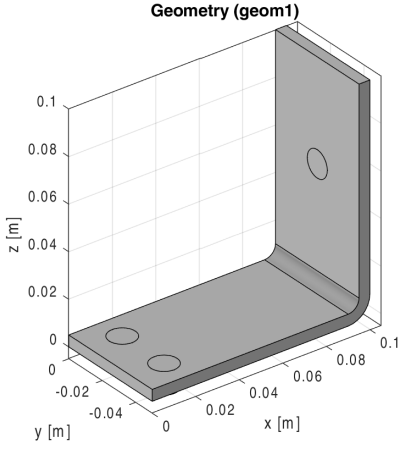
```
mphsetparam(model,'L',L0);
```

The parameter `L` (in the COMSOL model) is now assigned with the value of `L0`.

Note: In contrast to this command, setting a value using an expression from within the model object requires the second argument to be a string expression, as shown in Updating Model Settings.

3 To display the new geometry enter:

```
mphgeom(model)
```



Working with parameters is convenient, since there is only one place where you need to go to view or modify the current configuration of the model. The next step uses a different method, where a feature node is edited to modify its properties. For example, to modify the busbar geometry, the rectangles `r1` and `r2` generated in the work plane `wp1` are edited.

4 Create links to the geometry features `r1` and `r2` by entering the commands:

```
r1 = model.geom('geom1').feature('wp1').geom.feature('r1');
r2 = model.geom('geom1').feature('wp1').geom.feature('r2');
```

Note: The function `mphnavigator` can be used to get a list of properties and values for feature nodes of a model, as well as its API command. See Obtaining Model Information for more details.

5 Enter the following command to define MATLAB variables:

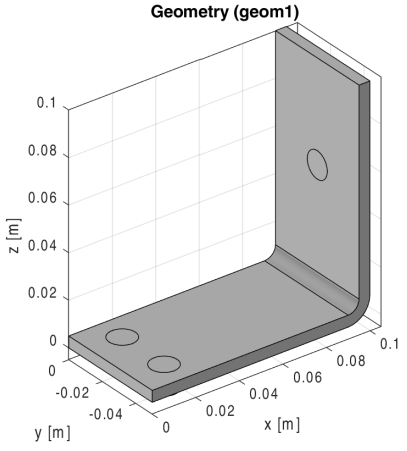
```
H1 = 0.2;
H2 = 0.195;
```

6 Set the height of rectangles `r1` and `r2` using the variables `H1` and `H2`, respectively.

```
r1.set('ly', H1);
r2.set('ly', H2);
```

7 Enter this command to plot the geometry:

```
mphgeom(model);
```



There is a third option when defining property values for features, which is to combine MATLAB variables with parameters or variables defined within the COMSOL model object. To do this, the MATLAB variable needs to be converted to a string.

These final steps demonstrate the method by changing the heights of rectangles `r1` and `r2` to `L0` and `L0-tbb`, respectively, where `L0` is a MATLAB variable.

8 Define the variable `L0`:

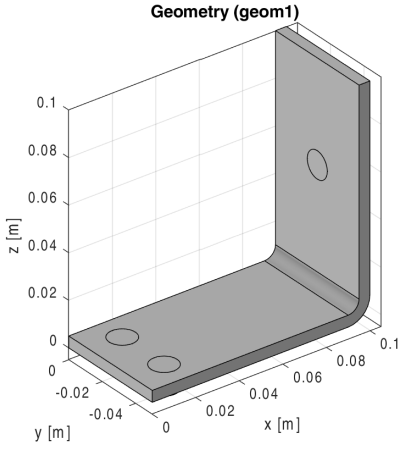
```
L0 = 0.1;
```

9 Modify the `ly` property of rectangles `r1` and `r2`:

```
r1.set('ly',L0);
r2.set('ly',[num2str(L0) '-tbb']);
```

10 Update and plot the geometry:

```
mphgeom(model);
```



Code for Use with MATLAB®

```
mphopen busbar
L0 = 9e-2;
mphsetparam(model,'L',L0);
mphgeom(model)

r1 = model.geom('geom1').feature('wp1').geom.feature('r1');
r2 = model.geom('geom1').feature('wp1').geom.feature('r2');
H1 = 0.2;
H2 = 0.195;
r1.set('ly', H1);
r2.set('ly', H2);
mphgeom(model)

L0 = 0.1;
r1.set('ly',L0);
r2.set('ly',[num2str(L0) '-tbb']);
mphgeom(model)
```

An Example Using Nested Loops

In this example, a MATLAB function is created that solves the busbar model for different values of the length and thickness of the busbar in addition to the input electric potential. The function evaluates and returns the maximum temperature in the busbar, the generated heat, and the current through the busbar. For each variation of the model, the function stores these results in a file and saves the model object in an MPH-file. To make it easy to identify the name of the

MPH-file, it contains the values of the different parameters used to set up the model.

The input argument of the M-function is the busbar model object and the path to the directory where the data and models are saved.

1. Open a text editor, for example, the MATLAB text editor.
2. At the first line of a new M-file, enter the following function header:

```
function modelParam(model,filepath)
```

   The function first creates a file and then sets the header of the output file format.

3. In the text editor, continue to enter the following lines:

```
filename = fullfile(filepath,'results.txt');
fid=fopen(filename,'wt');
fprintf(fid,'*** run parametric study ***\n');
fprintf(fid,'L[m] | tbb[m] | Vtot[V] | ');
fprintf(fid,'MaxT[K] | TotQ[W] | Current[A]\n');
```

4. When running a large number of iterations, the model history can be disabled to prevent the memory usage from increasing with each iteration due to the model history stored in the model. Enter the lines:

```
model.hist.disable;
```

5. Now start a `for` loop to parameterize the parameter `L` and set the corresponding parameter in the model object. Enter the commands:

```
for L = [9e-2 15e-2]
model.param.set('L',L);
```

6. Continue by creating a second `for` loop, this time to parameterize the parameter `tbb` and to modify its value in the model object:

```
for tbb = [5e-3 10e-3]
model.param.set('tbb',tbb);
```

7. Define the last `for` loop to parameterize the applied potential `Vtot`:

```
for Vtot = [20e-3 40e-3]
model.param.set('Vtot',Vtot);
```

8. For each version of the model, write the parameter values to the output file:

```
fprintf(fid,[num2str(L),' | ',num2str(tbb),...
' | ',num2str(Vtot),' | ']);
```

9. To solve the model, add the line below:

```
mphrun(model,'study');
```

10. The following lines evaluate the results:

```
MaxT = mphmax(model,'T',3,'selection',1);
TotQ = mphint2(model,'ht.Qtot',3,'selection',1);
Current = mphint2(model,'ec.normJ','surface','selection',43);
```

11. Save the results to the output text file:

```
fprintf(fid,[num2str(MaxT),' | ',num2str(TotQ),...
' | ',num2str(Current),' \n']);
```

12 Name and save the model:

```
modelName = fullfile(filepath,['busbar_L=',num2str(L),...
'_tbb=',num2str(tbb),...
'_Vtot=',num2str(Vtot),'.mph']);
mphsave(model,modelName);
```

13 Terminate the `for` loops:

```
end
end
end
```

14 Close the output text file:

```
fclose(fid);
```

The script in the text editor should now look like this text:

```
function modelParam(model,filepath)
filename = fullfile(filepath,'results.txt');
fid=fopen(filename,'wt');
fprintf(fid,'*** run parametric study ***\n');
fprintf(fid,'L[m] | tbb[m] | Vtot[V] | ');
fprintf(fid,'MaxT[K] | TotQ[W] | Current[A]\n');
model.hist.disable;
for L = [9e-2 15e-2]
  model.param.set('L',L);
  for tbb = [5e-3 10e-3]
    model.param.set('tbb',tbb);
    for Vtot = [20e-3 40e-3]
      model.param.set('Vtot',Vtot);
      fprintf(fid,[num2str(L),' | ',num2str(tbb),' | ',...
        num2str(Vtot),' | ']);
      mphrun(model,'study');
      MaxT = mphmax(model,'T',3,'selection',1);
      TotQ = mphint2(model,'ht.Qtot',3,'selection',1);
      Current = mphint2(model,'ec.normJ','surface','selection',43);
      fprintf(fid,[num2str(MaxT),' | ',num2str(TotQ), ' | ',...
        num2str(Current),' \n']);
      modelName = fullfile(filepath,...
        ['busbar_L=',num2str(L),'_tbb=',num2str(tbb),...
        '_Vtot=',num2str(Vtot),'.mph']);
      mphsave(model,modelName);
    end
  end
end
fclose(fid);
```

15 Save the M-file as `modelParam.m` in a directory known by MATLAB.

16 If COMSOL with MATLAB is not already running, start it now and load the busbar model from the application library:

```
model = mphopen('busbar');
```

17 Run the function and substitute `'filepath'` with the directory of your choice:

```
modelParam(model,'filepath')
```

18 Open the file `results.txt` in a text editor to look at a summary of the results:

```
*** run parametric study ***
L[m] | tbb[m] | Vtot[V] |  MaxT[K]  | TotQ[W]  | Current[A]
0.09 | 0.005  | 0.02    | 323.0784  | 0.2419   | 161.5116
0.09 | 0.005  | 0.04    | 413.0169  | 0.9676   | 323.0232
0.09 | 0.01   | 0.02    | 308.5303  | 0.0472   |  95.761
0.09 | 0.01   | 0.04    | 354.7501  | 0.1887   | 191.5219
0.15 | 0.005  | 0.02    | 315.7229  | 0.3249   | 156.3937
0.15 | 0.005  | 0.04    | 383.5576  | 1.2999   | 312.7875
0.15 | 0.01   | 0.02    | 305.1014  | 0.0643   |  94.84
0.15 | 0.01   | 0.04    | 340.9558  | 0.2571   | 189.6799
```

Although this example duplicates functionality that is readily accessible in the COMSOL Desktop®, you can use a similar technique to couple a COMSOL Multiphysics model to your customized optimization script.

Another advantage of using LiveLink™ *for* MATLAB® is the ability to call a MATLAB® function directly from within the COMSOL Desktop®. You can use a function to define, within a COMSOL® model object, properties such as material definitions or physics settings. COMSOL Multiphysics automatically detects the external function and starts the MATLAB engine to evaluate it. The function output is then applied to the corresponding property.

To use this functionality, just start the COMSOL Desktop and MATLAB automatically starts in the background when needed.

Note: To run MATLAB functions with the model object, the preference Allow external MATLAB® functions needs to be set to either Yes or Ask. This preference can be found on the Security page of the Preferences window, available from the File (or Options) menu.

Note: On Linux® operating systems, specify the MATLAB root directory `MLROOT` and load the gcc library when you start the COMSOL Desktop: `comsol -mlroot MLROOT -forcegcc`.

The example in this section illustrates how to use external MATLAB functions in a COMSOL model. The model computes the temperature distribution in a material subjected to an external heat flux. Both the thermal conductivity of the material and the applied heat flux are defined by MATLAB functions.

Creating the MATLAB® Functions

Assume that the material in the example is characterized by a random thermal conductivity. To define the thermal conductivity, create a MATLAB function with the x-axis position as an input argument and let the conductivity vary randomly within 200 W/(m·K) around a constant value of 400 W/(m·K).

Define a MATLAB function for the external heat flux to be a function of the following arguments:

- `x` and `y`, the location coordinates.
- `x0` and `y0`, the origin of the heat source.
- `Q0`, the maximum heat source.
- `scale`, a parameter that scales the period of the heat flux.

`x` and `y` are dependent variables of the model. `x0`, `y0`, `Q0`, and `scale` are defined with constant value in this exercise.

The functions must adhere to a certain structure. In order for the calls from COMSOL to MATLAB to be as efficient as possible the arguments are transferred to MATLAB in blocks. This means that all the input arguments for the functions are vectors, and that the functions need to return vectors with the same length as the input arguments. It is highly recommended to test the functions on the MATLAB command line with some reasonable vector arguments.

1 Open a text editor, for example the MATLAB text editor.

2 In a new file enter:

```
function out = conductivity(x)
out = 400+5*randn(size(x));
```

3 Save the file in the user Documents folder under MATLAB as `conductivity.m`. Saving the file in this location (/Documents/MATLAB) ensures that MATLAB can find the function when COMSOL calls it for evaluation.

Note: If you want to save the function in a specific directory, saving the COMSOL model at the same location as the external MATLAB functions ensures that the function path is known by MATLAB. Other alternatives include setting the function path directly in MATLAB before running the model or setting the environmental variable COMSOL_MATLAB_PATH with the directory path of the functions.

4 Create a new file and enter:

```
function out = heatflux(x,y,x0,y0,Q0,scale)
radius = sqrt((x-x0).^2+(y-y0).^2);
out = Q0/5+Q0/2.*sin(scale*pi.*radius)./(scale*pi.*radius);
```

5 Save the file as `heatflux.m`, in the same location as the previous file.

Model Wizard

Note: These instructions are for the user interface on Windows®, but apply with minor differences also to Linux® and macOS.

1 To start the software, double-click the COMSOL Multiphysics icon on the desktop.

2 Select Create a model guided by the Model Wizard .

3 In the space dimension, click 3D .

4 In the Select Physics tree under Heat Transfer , select Heat Transfer in Solids(ht) .

5 Click Add, then click Study .

6 In the Select Study window under General Studies, select Stationary .

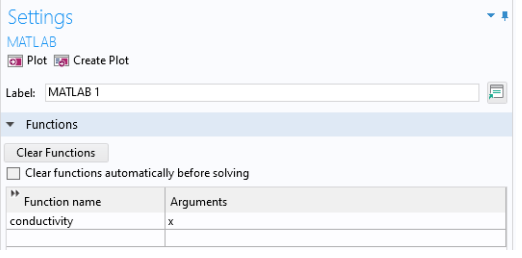
7 Click Done .

Defining External MATLAB® Functions in a Model

Start by defining a MATLAB function in a model so that COMSOL Multiphysics can recognize it as an external function to be evaluated in the MATLAB engine.

1 In the Home ribbon, click Functions . Under the Global section, select MATLAB . On Linux and macOS, this can be found in the Model Toolbar.

2 On the Settings page, under the Functions section, enter `conductivity` in the Function name field, and enter `x` in the Arguments field.



By defining the function here, COMSOL knows that conductivity is a MATLAB function. COMSOL automatically starts a MATLAB engine to evaluate the function when necessary.

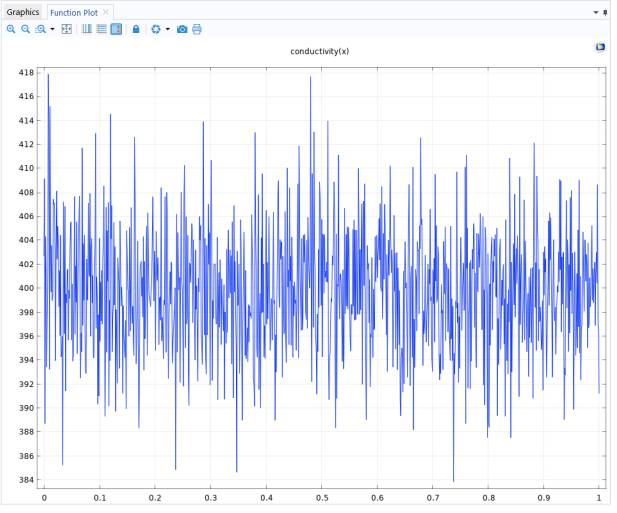
In case you did not save the function in a directory that is in the MATLAB path you need to save first the model MPH-file.

3 In the COMSOL Desktop, from the File menu (Windows users) or from the Options menu (Mac and Linux users), click Save As . Select COMSOL Application (`*.mph`) in the Save as type list, and browse to the directory where the files `conductivity.m` and `heatflux.m` are stored.

You can display the value of the defined function by first defining the plot limit for the input arguments.

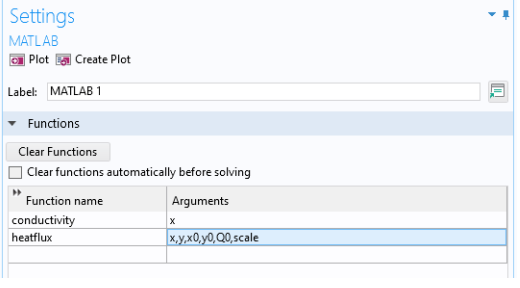
4 Expand the Plot parameters section. In the associated table enter `0` as the Lower limit and `1` as the Upper limit.

5 Click the Plot button . This will start the MATLAB engine where the function is evaluated and produce the following plot.



You can now define a second MATLAB function that returns the heat flux condition.

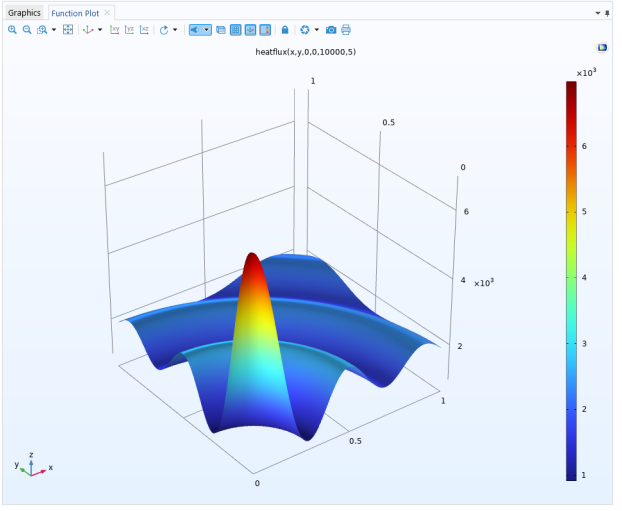
6 Repeating the above procedure, enter `heatflux` in the Function name field and enter `x,y,x0,y0,Q0,scale` in the Arguments field.



7 To display the function, enter the following in the Plot parameters table:

LOWER LIMIT	UPPER LIMIT
0	1
0	1
0	0
0	0
1e4	1e4
5	5

8 Click the Plot button .



Before you continue with the model settings, you need to manually specify the function derivative with respect to all function arguments. If this is not done, the solver will return a warning message. For this model, the function arguments are not defined using the temperature (the dependent variable) and the numerical problem can be considered linear. For this reason, you can set the derivative to be 0 for all input arguments.

9 Expand the Derivatives section and complete the table as shown below:

FUNCTION NAME	ARGUMENT	PARTIAL DERIVATIVE
`conductivity`	`x`	`0`
`heatflux`	`x`	`0`
`heatflux`	`y`	`0`
`heatflux`	`x0`	`0`
`heatflux`	`y0`	`0`
`heatflux`	`Q0`	`0`
`heatflux`	`scale`	`0`

Note: For nonlinear problems, it is necessary to specify the partial derivatives. The Partial Derivative column can be set with an expression or another MATLAB function.

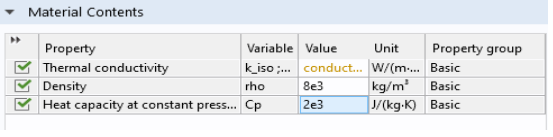
Geometry and Material Definition

1 In the ribbon Geometry tab, click Block .

2 In the Settings window for Block, click Build All Objects .

3 In the ribbon Materials tab, click Blank Material .

4 In the Settings window for Materials, under Material Contents, set the value of the Thermal conductivity to `conductivity(x)`, the Density to `8e3` and the Heat Capacity at Constant Pressure to `2e3`.



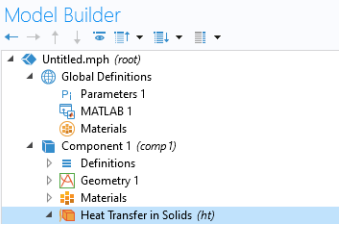
Note: The expression `conductivity(x)` displays in orange as the MATLAB function is dimensionless.

Heat Transfer in Solids

1 In the Model Builder under Component 1, click Heat Transfer in Solids .

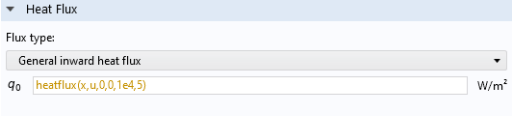
2 In the ribbon Physics tab, click Boundaries and choose Temperature .

3 In the Settings window for Temperature, select boundaries 3, 5, and 6 (You can also use the Paste Selection button to directly enter the selection).



4 To add a boundary heat flux, in the ribbon Physics tab, click Boundaries and select Heat Flux .

5 In the Settings window for Heat Flux, select boundary 4 and set the General inward heat flux expression to `heatflux(x,y,0,0,1e4,5)`.

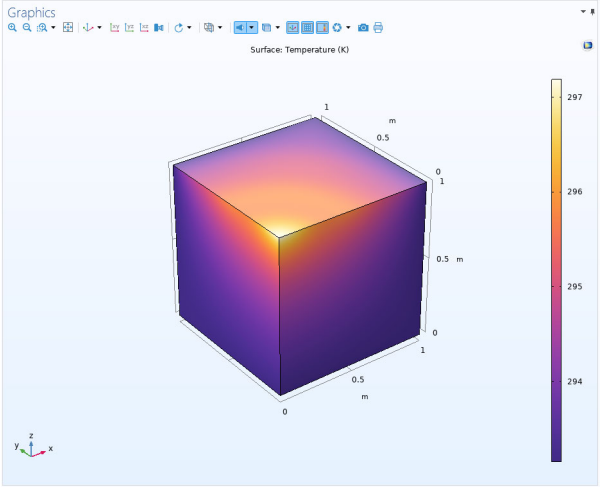


Computing and Plotting the Results

Once the model is set up, the solution can be computed. A default mesh is automatically generated.

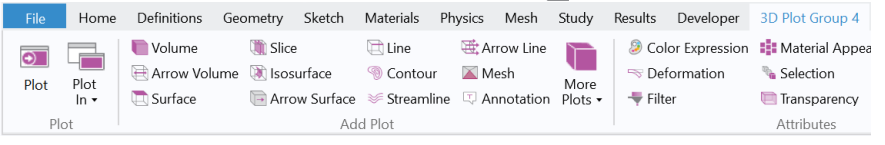
1 In the Study tab in the ribbon, click Compute .

2 Click the Temperature (ht) node to display the temperature field at the geometry surface.



3 Visualize the material thermal conductivity on the geometry by adding a new 3D Plot group. In the Results tab, select 3D Plot Group . An additional tab containing Plot Tools for the 3D Plot Group 2 appears when the 3D Plot Group 2 is selected in the Model Builder.

4 In the 3D Plot Group 2 tab, click Isosurface .

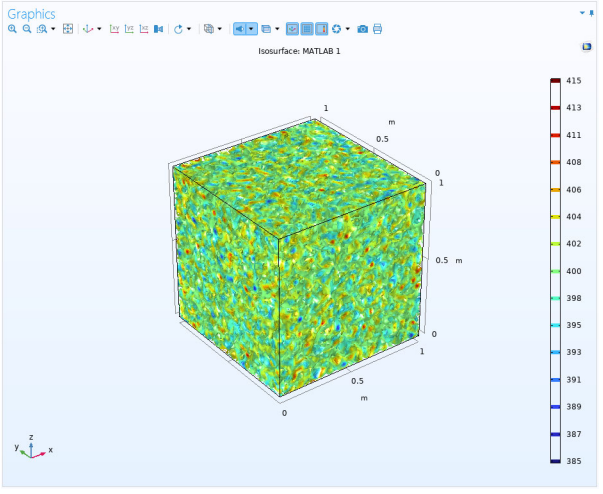


5 In the Model Builder, under the 3D Plot group 2 node, select the Isosurface 1 node.

6 On the associated Settings page, under the Expression section, enter `conductivity(x)` in the Expression field.

7 Under the Levels section, change the Total levels to `15`.

8 Click the Plot button .



Note: The conductivity function has been evaluated again in MATLAB to generate the above plot. As the function is using a random distribution, the plot does not exactly represent the distribution of the thermal conductivity when the solution has been computed.