Optimization Module

User's Guide

COMSOL

Optimization Module User's Guide

Contact Information

Visit the Contact COMSOL page at www.comsol.com/contact to submit general inquiries, contact Technical Support, 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 at 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/support/updates
- COMSOL Blog: www.comsol.com/blogs
- Discussion Forum: www.comsol.com/community
- Events: www.comsol.com/events
- COMSOL Video Gallery: www.comsol.com/video
- Support Knowledge Base: www.comsol.com/support/knowledgebase

Contents

Chapter 1: Introduction

Chapter 2: Optimization and Sensitivity Theory

Chapter 3: The Optimization Interface

Chapter 4: The Sensitivity Interface

Chapter 5: The Optimization Solvers

Chapter 6: Glossary

1

Introduction

Welcome to the *Optimization Module User's Guide*. The capabilities of the Optimization Module can be used in conjunction with any combination of other COMSOL products. This guide is a supplement to the *COMSOL Multiphysics Reference Manual*. In this section is a short Optimization Module Overview.

Optimization Module Overview

What Can the Optimization Module Do?

The Optimization Module can be used throughout the COMSOL product family —
it provides a general interface for calculating optimal solutions to engineering
problems. Any model inputs, be it geometric dimensions, part shapes, material
properties, or material distribution, can be treated as control variables, and any model
output can be an objective function.

Simulation is a powerful tool in science and engineering for predicting the behavior of
physical systems, particularly those governed by partial differential equations. In many
cases a single or a few simulations are not enough to provide sufficient understanding
of a system. Two important classes of problems whose resolution relies on a more
systematic exploratory process are:

- *Design problems* where the problem is to find the values of control variables or
  design variables that yield the best performance of a model, quantified by means of
  an objective function. Problems of this kind arise, for example, in structural
  optimization, antenna design, and process optimization.

- *Inverse problems*, and in particular *parameter estimation* in multiphysics models,
  where the problem is to reliably determine the values of a set of parameters that
  provide simulated data which best matches measured data. Such problems arise in,
  for example, geophysical imaging, nondestructive testing, and biomedical imaging.
  Curve fitting also belongs to this category.

Problems of the above types can often be formulated more generally as *optimization
problems*. The Optimization interface and Optimization study step in COMSOL
Multiphysics are useful for solving design problems as well as inverse problems and
parameter estimation.

OPTIMIZATION ALGORITHMS

There are three optimization algorithms for gradient-based optimization available in
the module. The first algorithm is based on the SNOPT code developed by Philip E.
Gill of the University of California San Diego, and Walter Murray and Michael A.
Saunders of Stanford University. When using SNOPT, the objective function can have
any form and any constraints can be applied. The algorithm uses a gradient-based
optimization technique to find optimal designs and when the underlying PDE is

stationary, frequency- or time-dependent, analytic sensitivities of the objective function with respect to the control variables can be used.

The second algorithm is the MMA solver, which is based on the *globally convergent method of moving asymptotes* by Krister Svanberg of KTH Royal Institute of Technology. The MMA solver can handle objective functions and constraints of the same very general form as SNOPT. It is well suited to handle problems with a large number of control variables, such as topology optimization.

The third algorithm is a Levenberg-Marquardt solver. When this solver is used, the objective function must be of least-squares type. Also, constraints are not supported. Since the Levenberg-Marquardt method is designed to solve problems of least-squares type, it typically converges faster than SNOPT and MMA for such problems.

In addition, the Optimization Module provides a number of gradient-free (derivative-free) optimization algorithms. Currently Nelder-Mead, BOBYQA, COBYLA, and a coordinate search are supported. These methods can optimize a model with respect to design parameters (model parameters) such as parameters which control the geometry sequence that defines the model's geometry. There is also a Monte Carlo method, useful for exploring the design space.

All optimization solvers are accessible from the same Optimization study step, which contains the ordinary solver sequence over which the optimization method iterates. The gradient-free methods can contain any other study sequence, while the gradient-based methods are limited to optimizing over a single study step of a type supporting computation of analytic sensitivities: currently Stationary, Time Dependent and Frequency Domain studies.

The Physics Interfaces and Building a COMSOL Multiphysics Model in the *COMSOL Multiphysics Reference Manual*

Where Do I Access the Documentation and Application Libraries?

Optimization model examples which do not require any other COMSOL add-on modules are located in the Optimization Module folder in the Application Libraries window. There are also other optimization models found in other folders for different modules. You can find all related examples entering `optimization` in the **Search** field.

A number of internet resources have more information about COMSOL, including licensing and technical information. The electronic documentation, topic-based (or context-based) help, and the application libraries are all accessed through the COMSOL Desktop.

> **!** If you are reading the documentation as a PDF file on your computer, the blue links do not work to open an application or content referenced in a different guide. However, if you are using the Help system in COMSOL Multiphysics, these links work to open other modules, application examples, and documentation sets.

THE DOCUMENTATION AND ONLINE HELP

The *COMSOL Multiphysics Reference Manual* describes the core physics interfaces and functionality included with the COMSOL Multiphysics license. This book also has instructions about how to use COMSOL Multiphysics and how to access the electronic Documentation and Help content.

Opening Topic-Based Help

The Help window is useful as it is connected to the features in the COMSOL Desktop. To learn more about a node in the Model Builder, or a window on the Desktop, click to highlight a node or window, then press F1 to open the Help window, which then displays information about that feature (or click a node in the Model Builder followed by the **Help** button (**?**). This is called *topic-based* (or *context) help*.

> **Win** To open the **Help** window:
>
> - In the **Model Builder**, **Application Builder**, or **Physics Builder** click a node or window and then press F1.
>
> - On any toolbar (for example, **Home**, **Definitions**, or **Geometry**), hover the mouse over a button (for example, **Add Physics** or **Build All**) and then press F1.
>
> - From the **File** menu, click **Help** (**?**).
>
> - In the upper-right corner of the COMSOL Desktop, click the **Help** (**?**) button.

<table>
<tr><td>Mac
Linux</td><td>To open the **Help** window:

• In the **Model Builder** or **Physics Builder** click a node or window and then press F1.

• On the main toolbar, click the **Help** (?) button.

• From the main menu, select **Help>Help**.</td></tr>
</table>

Opening the Documentation Window

<table>
<tr><td>Win</td><td>To open the **Documentation** window:

• Press Ctrl+F1.

• From the **File** menu select **Help>Documentation** (▮).</td></tr>
</table>

<table>
<tr><td>Mac
Linux</td><td>To open the **Documentation** window:

• Press Ctrl+F1.

• On the main toolbar, click the **Documentation** (▮) button.

• From the main menu, select **Help>Documentation**.</td></tr>
</table>

THE APPLICATION LIBRARIES WINDOW

Each model or application includes documentation with the theoretical background and step-by-step instructions to create a model or app. The models and applications are available in COMSOL Multiphysics as MPH files that you can open for further investigation. You can use the step-by-step instructions and the actual models as templates for your own modeling. In most models, SI units are used to describe the relevant properties, parameters, and dimensions, but other unit systems are available.

Once the Application Libraries window is opened, you can search by name or browse under a module folder name. Click to view a summary of the model or application and its properties, including options to open it or its associated PDF document.

<table>
<tr><td>🔍</td><td>The Application Libraries Window in the *COMSOL Multiphysics Reference Manual*.</td></tr>
</table>

Opening the Application Libraries Window

To open the **Application Libraries** window ( ):

Win	• From the **Home** toolbar, **Windows** menu, click ( ) **Applications Libraries**.
	• From the **File** menu select **Application Libraries**.
	To include the latest versions of model examples, from the **File>Help** menu, select ( ) **Update COMSOL Application Library**.

Mac	Select **Application Libraries** from the main **File>** or **Windows>** menus.
Linux	To include the latest versions of model examples, from the **Help** menu select ( ) **Update COMSOL Application Library**.

CONTACTING COMSOL BY EMAIL

For general product information, contact COMSOL at info@comsol.com.

COMSOL ACCESS AND TECHNICAL SUPPORT

To receive technical support from COMSOL for the COMSOL products, please contact your local COMSOL representative or send your questions to support@comsol.com. An automatic notification and a case number are sent to you by email. You can also access technical support, software updates, license information, and other resources by registering for a COMSOL Access account.

COMSOL ONLINE RESOURCES

COMSOL website	www.comsol.com
Contact COMSOL	www.comsol.com/contact
COMSOL Access	www.comsol.com/access
Support Center	www.comsol.com/support
Product Download	www.comsol.com/product-download
Product Updates	www.comsol.com/support/updates
COMSOL Blog	www.comsol.com/blogs
Discussion Forum	www.comsol.com/community
Events	www.comsol.com/events

COMSOL Video Gallery	www.comsol.com/video
Support Knowledge Base	www.comsol.com/support/knowledgebase

2

Optimization and Sensitivity Theory

This chapter discusses the theory for optimization and sensitivity. In this chapter:

- Optimization Theory
- Theory for the Sensitivity Interface
- Theory for Topology Optimization

Optimization Theory

This section contains theory useful for understanding and applying The Optimization Interface and The Optimization Study. Topics explained in this section are:

- Basic Optimization Concepts
- Optimization Problem Formulation
- PDE-Constrained Optimization

Basic Optimization Concepts

In general there are three fundamental parts of an optimization problem — the *control variables*, the *objective function* and, optionally, *constraints*.

The optimization problem is to find the value of the control variables that minimizes (or maximizes) the objective function, subject to a number of constraints. The constraints collectively define a set, the *feasible set*, of permissible values for the control variables.

The Optimization Study together with The Optimization Interface provide a framework for specifying and solving general optimization problems. The objective function and constraints can depend indirectly on the control variables via the solution of a multiphysics model. See PDE-Constrained Optimization.

Optimization Problem Formulation

The Optimization Module is built around a general single-objective minimization problem formulation. The Optimization Study node transforms maximization as well as multi-objective minimax and maximin problems internally to the canonical minimization form.

THE GENERAL OPTIMIZATION PROBLEM

The most general formulation of an optimization problem can be written as

$$\begin{cases} \min_{\xi} Q(\xi) \\ \xi \in C \end{cases} \tag{2-1}$$

Here, the control variables are denoted by ξ, the scalar-valued objective function by Q, and the feasible set is denoted by C. Assuming sufficient continuity, the feasible set can be expressed as a set of — possibly very nonlinear — inequality constraints

$$C = \{\xi : \text{lb} \leq G(\xi) \leq \text{ub}\}$$

where G is a vector-valued function (G is scalar-valued in case of a single constraint).

> For vectorial quantities, the inequality defining C is to be interpreted component-wise and lb and ub are the corresponding vectors containing the upper and lower bounds.

CLASSICAL OPTIMIZATION

In *classical optimization*, Q and G are given explicitly as closed-form expressions of the control variables ξ. However, design problems and parameter estimation problems often result in objective functions Q and constraints G that are not explicitly expressible as closed-form expressions of the control variables ξ.

PDE-Constrained Optimization

In multiphysics modeling, it is often desirable to let control variables parameterize the problem and seek to optimize a function of the PDE solution. The objective function is therefore a function of both the control variables and the PDE solution, which is in turn a function of the control variables. The multiphysics problem is a PDE, which after discretization is represented as a system of equations $L(u(\xi), \xi) = 0$, where u is the PDE solution and ξ the control variables.

The complete PDE-constrained optimization problem to be solved by one of the optimization algorithms in the Optimization Module adds the PDE problem as an equality constraint to the general optimization problem:

$$
\begin{cases}
\min_{\xi} \quad Q(u(\xi), \xi) \\
\\
L(u(\xi), \xi) = 0 \\
\text{lb} \leq G(u(\xi), \xi) \leq \text{ub}
\end{cases}
\tag{2-2}
$$

It is advantageous to separate those constraints in G that are defined as explicit expressions of ξ only (*design constraints*) from those that mix u and ξ (*performance constraints*). The former group can further be divided into *simple bounds*, which set

a lower and upper limit directly on the control variables, and constraints on general expressions of the control variables. Hence, the general constraint formulation $\text{lb} \leq G(u(\xi), \xi) \leq \text{ub}$ above is replaced by three classes of constraints:

$$\text{lb}_P \leq P(\xi, u) \leq \text{ub}_P$$
$$\text{lb}_\Psi \leq \Psi(\xi) \leq \text{ub}_\Psi$$
$$\text{lb}_b \leq \xi \leq \text{ub}_b$$

and the optimization problem in Equation 2-2 can be written as

$$\left\{ \begin{array}{l} \min_{\xi} \quad Q(u(\xi), \xi) \\ L(u(\xi), \xi) = 0 \\ \text{lb}_P \leq P(u(\xi), \xi) \leq \text{ub}_P \\ \text{lb}_\Psi \leq \Psi(\xi) \leq \text{ub}_\Psi \\ \text{lb}_b \leq \xi \leq \text{ub}_b \end{array} \right. \tag{2-3}$$

This is the general form of the optimization problem considered in the Optimization Module. Control variables ξ can be either global model parameters converted into control variables in the Optimization study step, or control variable fields set up in an Optimization interface. You specify the objective function and the constraints in the form of expressions in ξ and u. The relation between u and ξ, which is a system of equations written here compactly as $L(u, \xi) = 0$, is given by the multiphysics model.

SPECIFICATION OF THE OBJECTIVE FUNCTION

The objective function is, in general, a sum of a number of terms:

$$Q(u, \xi) = Q_{\text{global}}(u, \xi) + Q_{\text{probe}}(u, \xi) + \sum_{k=0}^{n} Q_{\text{int}, k}(u, \xi)$$

where n is the space dimension of the multiphysics model and the different contributions in the sum above are defined as follows:

- Q_{global} is the *global contribution* to the objective function Q. It is given as one or more general global expressions, either in an Optimization study step or in a Global Objective node under an Optimization interface.

- Q_{probe} is a *probe contribution* to the objective function Q. It is a *probe objective* so its definition is restricted to a point on a given geometrical entity. You specify the

probe point used for the point evaluation explicitly in a Probe Objective node under an Optimization interface.

- $Q_{\text{int},k}$ is an *integral contribution* to the objective function Q. It is an *integral objective* so its definition is restricted to a set of geometric entities of the same dimension. Use an Integral Objective node under an Optimization interface to specify an integrand and a select a set of domains, boundaries, edges or points over which to integrate. For a point selection, the integration reduces to a summation.

Several global, probe, and integral contributions can be defined in separate nodes under an Optimization interface. In such cases, the total global, probe, and integral contribution is given as the sum of the contributions. If you specify one or more objectives directly in the Optimization study step, these are also added to the sum.

SPECIFICATION OF CONSTRAINTS

The full nonlinear set of constraints $\text{lb} \leq G(u(\xi), \xi) \leq \text{ub}$ in the general PDE-constrained optimization problem, [Equation 2-2](), are separated into three groups:

$$
\begin{aligned}
\text{lb}_P &\leq P(\xi, u) \leq \text{ub}_P \\
\text{lb}_\Psi &\leq \Psi(\xi) \leq \text{ub}_\Psi \\
\text{lb}_b &\leq \xi \leq \text{ub}_b
\end{aligned}
$$

The first row above contains the general *implicit constraints,* or *performance constraints* in the case of a design problem. These are given in terms of expressions involving both the solution variables u and control variables ξ. The second row constitutes the *explicit constraints* — or *design constraints* — which are those

constraints given by explicit expressions only in the control variables ξ. The last row contains the *control variable bounds*

> The reason for this subdivision is computational. Each evaluation of an implicit constraint requires an up-to-date solution of the multiphysics solution u. The gradient-based optimization methods also require a complete sensitivity evaluation, which is computationally demanding, see Choosing a Sensitivity Method in the *COMSOL Multiphysics Reference Manual*.
>
> Explicit constraints, in contrast, can be computed without updating the multiphysics solution. They can, however, be nonlinear, making it difficult for the optimization methods to follow an active constraint. Control variable bounds are the least expensive to handle, since when active, the optimization solver can essentially just exclude the corresponding control variable.

The **Optimization** interface differentiates between the following constraints (in the description that follows, n denotes the dimension of the multiphysics model): control variable bounds, pointwise inequality constraints, integral inequality constraints, and global inequality constraints, each of which are described below

- *Bounds* or *control variable bounds* are inequality constraints setting lower and upper bounds directly on each control variable degree of freedom. Hence, bound constraints correspond to constraints of the form lb ≤ ξ ≤ ub. They are handled efficiently by all solvers that support them and in many cases improve solver stability and efficiency.
- *Pointwise inequality constraints* are inequality constraints involving an explicit expression in terms of the control variables. The constraint sets lower and upper bounds on the expression for node points in a set of geometric entities of the same dimension.
- *Global inequality constraints* set upper and lower bounds on a general global expression, possibly involving both the control variables and the PDE solution. Apart from the specification of bounds, a global inequality constraint is identical to a Global Objective.
- *Integral inequality constraints* set upper and lower bounds on an integral of an expression, possibly involving the PDE solution and control variables, over a set of

geometric entities of the same dimension. For integral inequality constraints on points, the integration reduces to a summation.

Global inequality constraints and integral inequality constraints are structurally similar to the objective function and equally expensive to evaluate.

Theory for the Sensitivity Interface

About Sensitivity Analysis

The Sensitivity interface is special in the sense that it does not contain any physics of its own. Instead, it is a tool for evaluating the sensitivity of a model with respect to almost any variable. The Sensitivity interface is used together with a Sensitivity study step, which in turn controls the Sensitivity solver extension. Simple cases can be handled directly in the Sensitivity study step, while more advanced cases must be set up in a Sensitivity interface prior to solving.

Simulation is a powerful tool for predicting the behavior of physical systems, particularly those that are governed by partial differential equations. However, a single simulation is often not enough to provide sufficient understanding of a system. Hence, a more exploratory process might be needed, such as *sensitivity analysis*, where one is interested in the sensitivity of a specific quantity with respect to variations in certain parameters included in the model. Such an analysis can, for example, be used for estimating modeling errors caused by uncertainties in material properties or for predicting the effect of a geometrical change.

Many times it is possible to reformulate problems of the above type as the problem of calculating derivatives, so differentiation plays a central role in solving such problems. The Sensitivity study step and corresponding physics interface can calculate derivatives of a scalar *objective function* with respect to a specified set of *control variables*. The objective function is in general a function of the solution to a multiphysics problem, which is in turn parameterized by the control variables.

Sensitivity Problem Formulation

Because the Sensitivity interface does not contain any physics, it is not intended for use on its own. When the physics interface is added to a multiphysics model, no new equations are introduced, and the set of solution variables remains the same. Instead,

an objective function and a set of control variables can be specified. The Sensitivity interface can perform these distinct tasks:

- Select control variables and set their values
- Define scalar objective functions

> The control variables are independent variables whose values are not affected by the solution process, but they are also degrees of freedom (DOFs) stored in the solution vector. When defining a control variable, its *initial value* must be supplied. The initial value is used to initialize the control variable DOFs, which remain fixed during the solution step.

The companion Sensitivity study step is responsible for:

- Choosing which objective functions and control variables to solve for
- Selecting a sensitivity evaluation method
- Selecting which study step to compute sensitivities for
- Setting up the Sensitivity solver extension

Theory for Stationary Sensitivity Analysis

Evaluating the sensitivity of a scalar-valued objective function $Q(\xi)$ with respect to the control variables, ξ, at a specific point, ξ_0, can be rephrased as the problem of calculating the derivative $\partial Q/\partial \xi$ at $\xi = \xi_0$. In the context of a multiphysics model, Q is usually not an explicit expression in the control variables ξ alone. Rather, $Q(u(\xi), \xi)$ is also a function of the solution variables u, which are in turn implicitly functions of ξ.

The multiphysics problem is a PDE, which after discretization is represented as a system of equations $L(u(\xi), \xi) = 0$. If the PDE has a unique solution $u = L^{-1}(\xi)$, the sensitivity problem can be informally rewritten using the chain rule as that of finding

$$\frac{d}{d\xi}Q(u(\xi), \xi) = \frac{\partial Q}{\partial \xi} + \frac{\partial Q}{\partial u} \cdot \frac{\partial u}{\partial L} \cdot \frac{\partial L}{\partial \xi}$$

The first term, which is an explicit partial derivative of the objective function with respect to the control variables, is easy to compute using symbolic differentiation. The second term is more difficult. Assuming that the PDE solution has N degrees of

freedom and that there are n control variables ξ_i, $\partial Q/\partial u$ is an N-by-1 matrix, $\partial u/\partial L$ is an N-by-N matrix (because L^{-1} is unique), and $\partial L/\partial \xi$ is an N-by-n matrix.

> The system of equations, L, is here assumed to include any constraints present in the multiphysics model. The number of degrees of freedom, N, therefore in theory includes also Lagrange multipliers for the constraints. In practice, constraints are usually eliminated, which imposes some restrictions on the sensitivity analysis; see The Sensitivity Analysis Algorithm in the *COMSOL Multiphysics Reference Manual*.

The first and last factors, $\partial Q/\partial u$ and $\partial L/\partial \xi$, can be computed directly using symbolic differentiation. The key to evaluating the complete expression lies in noting that the middle factor can be computed as $\partial u/\partial L = (\partial L/\partial u)^{-1}$ and that $\partial L/\partial u$ is the PDE Jacobian at the solution point:

$$\frac{d}{d\xi}Q(u(\xi), \xi) = \frac{\partial Q}{\partial \xi} + \frac{\partial Q}{\partial u} \cdot \left(\frac{\partial L}{\partial u}\right)^{-1} \cdot \frac{\partial L}{\partial \xi} \tag{2-4}$$

Actually evaluating the inverse of the N-by-N Jacobian matrix is too expensive. In order to avoid that step, an auxiliary linear problem can be introduced. This can be done in two different ways, each requiring at least one additional linear solution step (see Forward Sensitivity Methods and Adjoint Sensitivity Method below).

If an incomplete Jacobian has been detected during the sensitivity analysis, an attempt to assemble the complete Jacobian is done. If the assemble succeeds, the complete Jacobian is used in sensitivity computations in the following way:

Assume that the Jacobian $\dfrac{\partial L}{\partial u}$ in Equation 2-4 above is incomplete and denote it by $\left(\dfrac{\partial L}{\partial u}\right)_{incomplete}$.

Let the complete Jacobian be $\dfrac{\partial L}{\partial u}$. Hence, the system to solve is

$$\frac{\partial L}{\partial u} \cdot \frac{\partial u}{\partial \xi_i} = \frac{\partial L}{\partial \xi_i} \tag{2-5}$$

Using

$$\frac{\partial L}{\partial u} = \left(\frac{\partial L}{\partial u}\right)_{incomplete} + \left(\frac{\partial L}{\partial u} - \left(\frac{\partial L}{\partial u}\right)_{incomplete}\right)$$

the previous system becomes

$$\left(\frac{\partial L}{\partial u}\right)_{\text{incomplete}} \cdot \frac{\partial u}{\partial \xi_i} = \frac{\partial L}{\partial \xi_i} + \left(\left(\frac{\partial L}{\partial u}\right)_{\text{incomplete}} - \frac{\partial L}{\partial u}\right) \cdot \frac{\partial u}{\partial \xi_i}$$

Then, the solution to the system in Equation 2-5 is approximated iteratively by

$$\left(\frac{\partial L}{\partial u}\right)_{\text{incomplete}} \cdot \left(\frac{\partial u}{\partial \xi_i}\right)^n = \frac{\partial L}{\partial \xi_i} + \left(\left(\frac{\partial L}{\partial u}\right)_{\text{incomplete}} - \frac{\partial L}{\partial u}\right) \cdot \left(\frac{\partial u}{\partial \xi_i}\right)^{n-1}$$

where n is the iteration number.

The iterations are terminated either when the estimated error is less than the relative tolerance used by the current solver (convergence), or when the number of iterations has reached the maximum number of iterations specified in the Fully Coupled or Segregated attribute node (nonconvergence).

If the previous algorithm does not converge (that is, the estimated error is larger than the given tolerance), the sensitivity computations are repeated using the incomplete Jacobian and the warning *Jacobian is incomplete. No convergence when attempting to use the complete Jacobian* is written.

If the assemble of the complete Jacobian fails, the incomplete Jacobian is used and the warning *Unable to assemble the complete Jacobian. Using incomplete Jacobian for sensitivity analysis* is written.

In the Optimization interface, a warning is written only if the optimization problem does not converge.

FORWARD SENSITIVITY METHODS

To use the *forward sensitivity* methods, introduce the N-by-n matrix of solution sensitivities

$$\frac{\partial u}{\partial \xi} = \left(\frac{\partial L}{\partial u}\right)^{-1} \cdot \frac{\partial L}{\partial \xi}$$

These can be evaluated by solving n linear systems of equations

$$\frac{\partial L}{\partial u} \cdot \frac{\partial u}{\partial \xi_i} = \frac{\partial L}{\partial \xi_i}$$

using the same Jacobian $\partial L/\partial u$, evaluated at $u(\xi_0)$. Inserting the result into Equation 2-4, the desired sensitivities can be easily computed as

$$\frac{d}{d\xi}Q(u(\xi), \xi) = \frac{\partial Q}{\partial \xi} + \frac{\partial Q}{\partial u} \cdot \frac{\partial u}{\partial \xi}$$

ADJOINT SENSITIVITY METHOD

To use the *adjoint sensitivity* method, introduce instead the N-by-1 adjoint solution u^*, which is defined as

$$u^* = \frac{\partial Q}{\partial u} \cdot \left(\frac{\partial L}{\partial u}\right)^{-1}$$

Multiplying this relation from the right with the PDE Jacobian $\partial L/\partial u$ and transposing leads to a single linear system of equations

$$\frac{\partial L}{\partial u}^T \cdot u^* = \frac{\partial Q}{\partial u}$$

using the transpose of the original PDE Jacobian.

Theory for Time-Dependent Sensitivity

FORWARD SENSITIVITY

When you enable sensitivity analysis, the time-dependent solvers can compute—in addition to the basic forward solution—the sensitivity of a functional

$$Q = Q(u_\xi, \xi, T) \tag{2-6}$$

with respect to the control variables ξ evaluated at the final time $t=T$. The forward solution u_ξ is a solution to the parameterized discrete forward problem

$$L(u_\xi, \xi) = N_F \Lambda_\xi \qquad M(u_\xi, \xi) = 0 \tag{2-7}$$

where Λ_ξ are the constraint Lagrange multipliers, or (generalized) reaction forces, corresponding to the constraints M. It is assumed that Q does not explicitly depend on Λ_ξ.

To compute the sensitivity of Q with respect to ξ, first apply the chain rule:

$$\frac{dQ}{d\xi} = \frac{\partial Q}{\partial \xi} + \frac{\partial Q}{\partial u}\frac{\partial u}{\partial \xi} \tag{2-8}$$

In this expression, the sensitivity of the solution with respect to the control variables, $\partial u/\partial\xi$, is still an unknown quantity. Therefore, differentiate the forward problem, Equation 2-7, formally with respect to ξ:

$$D\frac{\partial \dot{u}_\xi}{\partial\xi} + K\frac{\partial u_\xi}{\partial\xi} + N_F\frac{\partial \Lambda_\xi}{\partial\xi} = \frac{\partial L}{\partial\xi} + \frac{\partial N_F}{\partial\xi}\Lambda_\xi \qquad N\frac{\partial u_\xi}{\partial\xi} = \frac{\partial M}{\partial\xi}$$

Here, $D = -\partial L/\partial\dot{u}$ $K = -\partial L/\partial u$, and $N = -\partial M/\partial u$ as usual. Assuming that the constraint force Jacobian N_F is independent of ξ (that is, $\partial N_F/\partial\xi = 0$), you can write the above relations in matrix form

$$\begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix}\begin{pmatrix} \dfrac{\partial \dot{u}_\xi}{\partial\xi} \\ \dfrac{\partial \Lambda_\xi}{\partial\xi} \end{pmatrix} + J\begin{pmatrix} \dfrac{\partial u_\xi}{\partial\xi} \\ \dfrac{\partial \Lambda_\xi}{\partial\xi} \end{pmatrix} = \begin{pmatrix} \dfrac{\partial L}{\partial\xi} \\ \dfrac{\partial M}{\partial\xi} \end{pmatrix} \qquad J = \begin{bmatrix} K & N_F \\ N & 0 \end{bmatrix} \qquad (2\text{-}9)$$

solve for the sensitivities $\partial u_p/\partial\xi$ and $\partial\Lambda_p/\partial\xi$, with initial conditions $\partial u_{0\xi}/\partial\xi$ and $\partial\Lambda_{0\xi}/\partial\xi$, respectively, and plug them back for evaluation at $t=T$ into Equation 2-8.

If the number of individual control variables, ξ_j, is small, Equation 2-9 can be solved for each right-hand side $[\partial L/\partial\xi_j\ \partial M/\partial\xi_j]^T$ with corresponding initial conditions and the solution inserted into Equation 2-8. This is the *forward method*, which in addition to the sensitivity $dQ/d\xi$ returns the sensitivity of the solution, $\partial u_\xi/\partial\xi$. As an alternative the right-hand side of Equation 2-9 can be calculated by finite differences using the *forward numeric method*.

If there are many control variables and the sensitivity of the solution itself, $\partial u_\xi/\partial\xi$, is not required, the adjoint method is more efficient.

ADJOINT SENSITIVITY

The adjoint sensitivity method is based on using solution variables u^* and U^* known as the *adjoint solution*, to rewrite Equation 2-8:

$$\frac{dQ}{d\xi} = \left(\frac{\partial Q}{\partial\xi} - U^*\frac{\partial L}{\partial\xi}\right)\bigg|_{t=T} - u^*D\frac{\partial u}{\partial\xi}\bigg|_{t=0} - \int_0^T u^*\frac{\partial L}{\partial\xi}dt$$

$$\frac{d}{dt}(u^*D) - u^*K = 0$$

Note that it has been assumed that

$$\frac{d}{dT}\langle U^*D\frac{\partial u}{\partial \xi}\rangle = 0$$

The homogeneous adjoint equations are solved backward in time and requires "final" conditions for initialization. The final conditions for U^* and $u^\star$ are computed as:

$$U^*D\big|_{t = T} = 0$$

$$u^*D\big|_{t = T} = \frac{\partial Q}{\partial u} - U^*K\bigg|_{t = T}$$

On this form, only one forward and one backward (adjoint) problem must be solved regardless of the number of control variables, followed by an evaluation of the gradient for each variable. Obviously, this is much faster than the forward method if the number of variables is large with the drawback that the forward solution must be available at all times during the backward solution of the adjoint. To reduce the memory requirements for this, a checkpointing strategy is employed. This means that at a number of checkpoints the forward solution is stored in memory such that a hot start of the time-dependent solver can be performed to produce the forward solution in higher resolution between checkpoints when needed. This reduces the memory requirement at the cost of one additional forward solution.

Specification of the Objective Function

The objective function can in general be a sum of a number of terms:

$$Q(u, \xi) = Q_{\text{global}}(u, \xi) + Q_{\text{probe}}(u, \xi) + \sum_{k = 0}^{n} Q_{\text{int}, k}(u, \xi)$$

where n is the space dimension of the multiphysics model and the different contributions in the sum above are defined as follows:

- Q_{global} is the *global contribution* to the objective function Q. It is given as one or more general global expressions.

- Q_{probe} is a *probe contribution* to the objective function Q. It is a *probe objective*, so its definition is restricted to a point on a given geometrical entity. The *probe point*

used for the point evaluation is a point given by the user and has to be contained in the domain.

- $Q_{\text{int},k}$ is an *integral contribution* to the objective function Q. It is an *integral objective*, so its definition is restricted to a specific set of geometrical entities of the same dimension. For integral contributions on points, the integration reduces to a summation.

Several global, probe, and integral contributions can be defined. In such cases, the total global, probe, and integral contribution is given as the sum of the aforementioned global, probe, and integral contributions that are actively selected in the solver settings for the optimization.

Choosing a Sensitivity Method

To evaluate sensitivities as part of a multiphysics problem solution, an auxiliary linear problem must be solved, in addition to the original equation, using one of these methods:

- Select one of the Forward Sensitivity methods to evaluate the derivatives of all solution variables and an optional objective function.
- Select the Adjoint Sensitivity method to look only at derivatives of a scalar objective function.

FORWARD SENSITIVITY

Use the forward (or forward numeric) sensitivity method to solve for the derivatives of all dependent variables, plus an optional scalar objective function, with respect to a small number of control variables. The forward method requires one extra linear system solution for each control variable.

The linear system that must be solved is the same as the last linearizion needed for solving the forward model. Thus, when using a direct solver (for example, PARDISO) the extra work amounts only to one back-substitution per control variable DOF. The forward numeric method uses numerical perturbation rather than analytical methods to calculate forward sensitivities, and can be used when the analytical method fails for some reason or as a tool to verify that the analytical method is correct. In addition, the forward numeric method requires two additional residual evaluations. The iterative linear and segregated solvers can reuse preconditioners and other data but must otherwise perform a complete solution each time. Further, the forward numeric method only differentiates the PDE problem numerically (giving a numeric method for the forward sensitivity). The objective sensitivity is still differentiated analytically

(both with respect to the controls and with respect to the PDE variables). The functional sensitivity is therefore computed with a hybrid method.

ADJOINT SENSITIVITY

The adjoint method solves for the derivatives of a single scalar objective function with respect to any number of control variables, requiring only one single additional linear system solution. In addition to the objective function gradient, the discrete adjoint solution is computed. This quantity represents the sensitivity of the objective function with respect to an additional generalized force applied as a nodal force to the corresponding solution component.

The auxiliary linear system is in this case the transpose of the last linearizion needed for solving the forward model. The MUMPS and PARDISO linear solvers can solve the transposed problem at the cost of a back-substitution, while the SPOOLES linear solver needs to do a new factorization if the problem is not symmetric or Hermitian. The iterative solvers can reuse most preconditioning information as can the segregated solver, which, however, loops over the segregated steps in reversed order.

📄	Sensitivity analysis can be used together with all stationary and parametric standard solvers and with the BDF solver for transient studies. The available solvers are described in the section Studies and Solvers in the *COMSOL Multiphysics Reference Manual*.

Postprocessing Sensitivities

When a multiphysics problem is solved using sensitivity analysis, the generated solution contains stored sensitivity data. You can access this data in postprocessing using the `fsens` and `sens` operators:

- `fsens(<control_variable>)` evaluates the sensitivity (derivative) of the objective function with respect to the specified control variable. This result is available for all sensitivity methods. The result of `fsens` can be evaluated on the geometric entities where the control variable is defined. For a global control variable, `fsens` is available everywhere. In the same way, `fsensimag(<control_variable>)` evaluates the sensitivity (derivative) of the objective function with respect to the imaginary part of the specified control variable.

- `sens(<dependent_variable>,<control_variable>)` or `sens(<dependent_variable>,<control_DOF>)` evaluates the sensitivity (derivative) of the specified dependent variable with respect to the specified control

variable degree of freedom. This is only possible when forward sensitivity has been used, which computes and stores derivatives of the entire solution vector with respect to each control variable degree of freedom.

Global control variables can be identified by name. Otherwise, control variable degrees of freedom are identified by their index (starting from 1) among all control variables in the solution vector. The result of `sens` has the same geometric scope as the dependent variable argument; it can be plotted or evaluated wherever the dependent variable itself is available.

Issues to Consider Regarding the Control Variables

THE EFFECT OF DISCRETIZATION

The sensitivity analysis is always performed on the discretized system of equations. As already mentioned, the control variables can be a scalar, vector, or an element in some infinite-dimensional function space. In the latter case, it is represented on the finite element mesh, just like the solution variables, or global scalar quantities. When using a control variable field represented on the finite element mesh, the sensitivities are therefore associated with individual control variable degrees of freedom rather than with the field value at each point. This makes it difficult to interpret the result. For example, if a domain control variable is set up using a first-order Lagrange shape function representation to control the material density in a model, the solution contains the sensitivity of the objective function with respect to the discrete density value at each node point in the mesh. Because each node influences the density in a small surrounding region, the size of which varies from node to node, the individual sensitivities are not directly comparable to each other.

Displaying such domain control variables results in a plot that is not smooth due to the varying element size. It must therefore not be used to draw any conclusions about the physics and the effect of changing the physical field represented by the control variable. Some insight can, however, be gained by looking at the sensitivities divided by the mesh volume scale factor `dvol`. This makes the sensitivities in the plot comparable between different parts of the surface but still not mathematically well defined. In

particular, using discontinuous constant shape functions together with the division by `dvol` results in a plot that is proportional to the true pointwise sensitivity.

> 📋 If the plan is to use the sensitivities in an automatic optimization procedure, as is done through the Optimization interface available with the Optimization Module, the discrete nature of the sensitivities causes no additional complication. The optimization solver searches for optimum values of the discrete control variables using the discrete gradient provided by the sensitivity analysis.

GEOMETRICAL SENSITIVITY

You can use the control variables directly to parameterize any aspect of the physics that is controlled by an expression. This applies to material properties, boundary conditions, loads, and sources. However, the shape, size, and position of parts of the geometry cannot be changed as easily at solution time and require special attention.

Control variables cannot be used directly in the geometry description. Instead, the model must be set up using a Deformed Geometry or Moving Mesh interface to control the shape of the geometry. Then use control variables to control the mesh movement, effectively parameterizing the geometry.

> 📋 See Deformed Geometry and Moving Mesh in the *COMSOL Multiphysics Reference Manual* for details about these interfaces and ALE in general.

Issues to Consider Regarding the Objective Function

THE PRINCIPLE OF VIRTUAL WORK

Potential energy has a special status among scalar objective functions because its derivatives with respect to scalar control variables can in many cases be interpreted as (true or generalized) forces.

COMPLEX-VALUED OBJECTIVE FUNCTIONS

Sensitivity analysis can be directly applied only when the objective function is a real differentiable or complex analytic function of the control variables. This is usually not a severe constraint, even for frequency-domain models where the PDE solution variables are complex valued. One reason is that physical quantities of interest to the

analyst are always real valued, and if complex-valued control variables are required, it is possible to treat the real and imaginary parts separately.

Some PDE problem or the objective functions are, however, nonanalytic. This is the case, for example, when the equations or the objective function contain `real()`, `imag()`, or `abs()`. One solution in such cases is to enable **Split complex variables in real and imaginary parts** in the **Compile Equations** node corresponding to the study step for which sensitivity is computed. This converts the discretized PDE system from a complex-valued system to a real-valued system of double size, with separate degrees of freedom for the real and imaginary part. For this split system, also the nonanalytic functions are differentiable almost everywhere such that sensitivities can be computed.

One special from of nonanalytic objective function can be treated more efficiently than splitting the variables: many common quantities of interest are harmonic time averages, which can be written in the form $Q = \mathrm{real}(a \cdot \mathrm{conj}(b))$, where a and b are complex-valued linear functions of the solution variables and therefore implicit functions of the control variables. The problem with this expression is that, while Q is indeed a real-valued differentiable function of the control variables, it is not an analytical function of a and b. This complicates matters slightly because the sensitivity solver relies on symbolic partial differentiation and the chain rule.

While the partial derivatives of Q with respect to a and b are, strictly speaking, undefined, it can be proven that if they are chosen such that

$$Q(a + \delta a, b + \delta b) \approx Q(a, b) + \mathrm{real}\left(\frac{\partial Q}{\partial a}\delta a + \frac{\partial Q}{\partial b}\delta b\right) \qquad (2\text{-}10)$$

for any small complex increments δa and δb, the final sensitivities are evaluated correctly. The special function `realdot(a,b)` is identical to `real(a*conj(b))` when evaluated but implements partial derivatives according to Equation 2-10. For that reason, use it in the definition of any time-average quantity set as objective function in a sensitivity analysis.

Issues to Consider Regarding Constraints

The theory behind sensitivity analysis as presented above (under Theory for Stationary Sensitivity Analysis) assumes that constraints on the multiphysics problem are handled in the same way as with any other equations. This is indeed the case for *weak constraints*, which are implemented as a part of the main system of equations. Standard pointwise constraints are instead eliminated from the discretized equations at an early stage in the solution process. This elimination is not visible to the sensitivity

solver, which therefore may miss some symbolic derivative terms necessary for computing a correct sensitivity.

In particular, if the mixed second derivative of a standard constraint with respect to both PDE solution and control variables is nonzero, sensitivity will not be correctly computed. For example, for a solution variable u and a control variable p, a constraint:

- $u = p$ will give correct sensitivity.
- $u^2 = p^2$ will give correct sensitivity.
- $u^2 = up$ will give incorrect sensitivity.

If your multiphysics model contains constraints of the problematic type, you can still compute a correct sensitivity, provided that you enable *weak constraints* in the **Constraint Settings** section of the corresponding boundary condition node.

For technical details about the solver implementation, see The Sensitivity Analysis Algorithm in the *COMSOL Multiphysics Reference Manual*.

For more about the standard versus the weak constraints, see Boundary Conditions in the *COMSOL Multiphysics Reference Manual*.

Theory for Topology Optimization

About the Density Model

Topology optimization with the density method is one of the oldest and most simple techniques. A domain control variable is discretized on nodes or elements, and a fictitious material is introduced to account for the material boundary in an implicit way. An interpolation is then constructed such that the physical governing equation is solved wherever the control variable is equal to one, while an equation associated with the fictitious material is solved where the control variable is equal to zero. The interpolation is specific to the physics, and it is constructed such that intermediate value of the control variable are suboptimal; see Ref. 1. Furthermore, the problem might be ill posed in the absence of a constraint on the design freedom. This constraint is often imposed implicitly by means of a filter that introduces a minimum length scale via its *filter radius.*

The Helmholtz Filter

One can impose a minimum length scale, R_{min}, on a domain control variable, θ_c, using a Helmholtz filter (see Ref. 2):

$$\theta_f = R_{min}^2 \nabla^2 \theta_f + \theta_c$$

Here θ_f is the filtered material volume factor. The mesh element size is taken as the default filter radius, but the filter radius should not be smaller than mesh element size and you have to pick a fixed value to get mesh-independent designs. The filtered variable is continuous and discretized with linear polynomials. The filter can be turned

off, in which case $\theta_f = \theta_c$ holds. Figure 2-1 illustrates the filtering process, and it is taken from the Application Library model *Topology Optimization of an MBB Beam*.
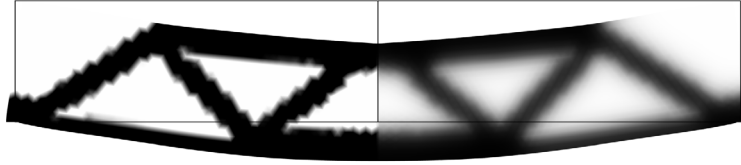

Figure 2-1: A symmetric topology optimization problem is shown with the control variable to the left and the filtered variable to the right.

Tanh Projection

The filtered design variable can have large areas with intermediate values. This tends to make the optimization problem easier to solve, but the design might rely on areas with unphysical properties due to intermediate values. These areas can then be reduced using a projection operation based on the hyperbolic tangent function (see Ref. 3):

$$\theta = \frac{(\tanh(\beta(\theta_f - \theta_\beta)) + \tanh(\beta\theta_\beta))}{(\tanh(\beta(1 - \theta_\beta)) + \tanh(\beta\theta_\beta))}$$

Here, θ is the output material volume factor while θ_β and β are the projection point and slope, respectively. θ is plotted in Figure 2-2 for $\theta_\beta = 0.5$ and $\beta = 8$. Projection can slow down the optimization progress, so it is turned off by default so that $\theta = \theta_f$.

Volume constraints are common in topology optimization. They are simple to implement using the average material volume factor, θ_{avg},

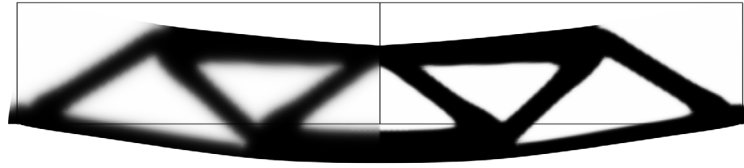$$\theta_{avg} = \int_\Omega \theta d\Omega / \int_\Omega d\Omega$$

Figure 2-2: A symmetric topology optimization problem is shown with the filtered variable to the left and the projected variable to the right.

Penalization

Intermediate design variables do not have a clear physical interpretation, so the optimization problem has to be modified, if they appear in the optimal design. In the context of volume constraints/objectives, this is typically achieved using penalization. This means that while θ is used for the volume computation, the penalized material volume, θ_p, is used for the material interpolation.

$$\text{SIMP} \qquad \theta_p = \theta_{min} + (1 - \theta_{min})\theta^p$$

$$\text{RAMP} \qquad \theta_p = \theta_{min} + \frac{\theta(1 - \theta_{min})}{1 + q(1 - \theta)}$$

$$\text{Darcy} \qquad \theta_p = \frac{q(1 - \theta)}{q + \theta}$$

The SIMP (Solid Isotropic Material with Penalization) and RAMP (Rational Approximation of Material Properties) are used for solid mechanics, while the Darcy interpolation is used within fluid mechanics.

Summary

The density topology adds the following variables listed in Table 2-1.

TABLE 2-1: VARIABLES ASSOCIATED WITH THE DENSITY TOPOLOGY FEATURE.

DESCRIPTION	NAME	INPUT TO
Control material volume factor	dtopo#.theta_c	Filter
Filtered material volume factor	dtopo#.theta_f	Projection
Output material volume factor	dtopo#.theta	Penalization

TABLE 2-1: VARIABLES ASSOCIATED WITH THE DENSITY TOPOLOGY FEATURE.

DESCRIPTION	NAME	INPUT TO
Penalized material volume factor	`dtopo#.theta_p`	Material properties
Average material volume factor	`dtopo#.theta_avg`	Volume constraint

References

1. M.P. Bendsøe and O. Sigmund, *Topology Optimization Theory, Methods, and Applications*, Springer, 2004.

2. B.S. Lazarov and O. Sigmund, "Filters in topology optimization based on Helmholtz-type differential equations", *International Journal for Numerical Methods in Engineering*, vol. 86, pp. 765–781, 2011.

3. F. Wang, B.S. Lazarov, and O. Sigmund, "On projection methods, convergence and robust formulations in topology optimization", *Structural and Multidisciplinary Optimization*, vol. 43, pp. 767–784, 2011.

3

The Optimization Interface

The Optimization interface, found under the **Mathematics>Optimization and Sensitivity** branch ( ) when adding an interface, is designed to facilitate setting up and solving advanced optimization problems. Problems which do not require least-squares contributions to the objective function, control variable fields, or pointwise constraints are preferably set up directly using only the Optimization study step node.

The optimization interface contains tools which let you set objective function, constraints, and bounds and introduce new control variable fields as well as global control variables.

In this section:

- Adding an Optimization Interface
- The Optimization Interface
- Topology Optimization

Adding an Optimization Interface

Add an **Optimization** interface when creating a new model or at any time during modeling. For a new model, physics interfaces are selected in the **Model Wizard** (after specifying the space dimension) or from the **Add Physics** window. In any active session, you can also right-click a **Component** node in the Model Builder to open the **Add Physics** window.

> ⊘ To add an **Optimization** interface to a Component using the **Model Wizard**, see Creating a New Model in the *COMSOL Multiphysics Reference Manual*.

To add an **Optimization** interface to a Component using the **Add Physics** window:

1 Under **Mathematics>Optimization and Sensitivity**, select **Optimization** (⦿).
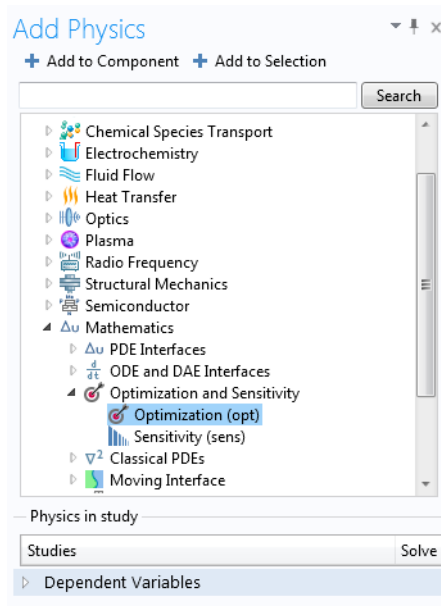
2 Click **Add to Component**. **Optimization** is added under the chosen Component in the Model Builder.

The Optimization Interface

The **Optimization (opt)** interface (🎯) contains tools for setting up advanced optimization problems. The main purpose of the interface is its ability to set up objective functions, constraint contributions and control variables which are defined locally only on certain geometric entities, as well as least-squares contribution with a time or parameter dependence.

Define objective functions and constraints in terms of control and solution variables (the latter are given as the solution to the differential equations defined by the multiphysics model) and restrict these to specific geometric entities, or make them globally available. The Optimization interface itself does not have any selection, and is not associated with any particular space dimension. Instead you find the same set of feature nodes for domains, boundaries, edges and points.

🔍
- Optimization Theory
- Common Physics Interface and Feature Settings and Nodes in the *COMSOL Multiphysics Reference Manual*

SETTINGS

The **Label** is the default physics interface name.

The **Name** is used primarily as a scope prefix for variables defined by the physics interface. Refer to such physics interface variables in expressions using the pattern `<name>.<variable_name>`. In order to distinguish between variables belonging to different physics interfaces, the `name` string must be unique. Only letters, numbers, and underscores (_) are permitted in the **Name** field. The first character must be a letter.

The default **Name** (for the first physics interface in the model) is `opt`.

The following nodes are available from the **Optimization** ribbon toolbar (Windows users), **Optimization** context menu (Mac or Linux users), or right-click to access the context menu (all users).

> For step-by-step instructions and general documentation descriptions, this is the **Optimization** toolbar. Subnodes are available by clicking the parent node and selecting it from the **Attributes** menu.

The following feature nodes are available for some dimensions. Some nodes are selected directly on the toolbar and others from submenus:

- Least-Squares Objective
- Integral Objective (Point Sum Objective)
- Probe Objective
- Integral Inequality Constraint (Point Sum Inequality Constraint)
- Pointwise Inequality Constraint
- Control Variable Field (which includes the settings for the associated bound constraints)
- Global Objective
- Global Least-Squares Objective
- Global Inequality Constraint
- Global Control Variables

The following subnodes are available by right-clicking the **Least Squares Objective** node:

- Value Column
- Time Column
- Parameter Column
- Coordinate Column
- Ignored Column

Least-Squares Objective

Uses a **Least-Squares Objective** feature to create an objective function representing the sum of squared differences between measurements stored in an experimental data file and a corresponding expression evaluated in the COMSOL Multiphysics model. The model expression is evaluated using interpolation on the feature's selection, at measurement locations specified in the data file.

	The number of coordinate columns in the data file must be the same as the dimension of the geometry, even when the selection of the **Least-Squares Objective** feature is on a lower dimension. In that case, model expressions are evaluated at the nearest points on the given selection.

	• Use a domain-level **Least-Squares Objective** node unless the model expressions corresponding to the measured data exists only on boundaries, edges, or points.
	• There is no need to add points to the geometry at the measurement locations specified in the file.
	• If your experimental data does not contain one or more columns with measurement locations, use a Global Least-Squares Objective feature instead.

To create a least-squares objective, first import an **Experimental Data** file containing comma-separated or semicolon-separated columns of measurement data from a single experiment. Each **Least-Squares Objective** feature corresponds to an experiment where the measurements have been obtained using given values for a set of **Experimental Parameters** (for example, the temperature during the experiment). The squared sum of the difference between the measurement values and the corresponding expressions evaluated in the model — when solved for the given parameter values — is added as a contribution to the total least-squares objective function.

Right-click the node to add column subnodes — Value Column, Time Column, Parameter Column, Coordinate Column, and Ignored Column — assigning meaning to the individual columns as values, times, parameter values, coordinate data, or values

to ignore, respectively. One column subnode must be added for each column in the data file and in the same order as the columns appear in the file.

> 💡 Move column nodes up and down using the context menu or a keyboard combination of the Ctrl key and an arrow key.

EXPERIMENTAL DATA

Enter a **Filename** or click the **Browse** button to specify a measurement data file containing comma-separated or semicolon-separated columns of measurements. The files are typically CSV files (*.csv), data files (*.dat), or plain text files (*.txt).

EXPERIMENTAL PARAMETERS

Click the **Add** button (➕) below the table to add an experimental parameter. Experimental parameters are useful for including additional parameters that represent model conditions for the experimental data and that are valid for the current experimental data file. In the **Name** column, choose a parameter name from the global parameters defined in the model. Enter a global-scope expression or value in the **Expression** column to assign a value to the parameter in this experiment. Use the **Load from File** (📂) and **Save to File** (💾) buttons to load and save experimental parameter names and expressions from and to a file. Use the **Delete** button (🗑) to remove the selected parameter from the table.

> 💡 If you have LiveLink™ *for* Excel®, you can also click the **Load from Excel File** (🔲) and **Save to Excel File** (🔲) buttons.
>
> See the *LiveLink™ for Excel® User's Guide* for more information. Or go to http://www.comsol.com/livelink-for-excel/ to learn more about the product.

Value Column

To add a **Value Column** subnode, right-click the **Least-Squares Objective** node. Use a **Value Column** to identify a column in the experimental data file as containing measurement values. Enter a corresponding **Expression**, which must be available for evaluation on the geometric entities selected in the node. Enter a corresponding **Column contribution weight**, which must be strictly positive and be available for evaluation in the global scope in the current model. Optionally, a **Variable name** can be specified to enable access to the data from the file for postprocessing. A **Unit** can also

be specified if desired, using the syntax for a valid unit definition in COMSOL Multiphysics. The difference between the **Expression** and the value from the file is squared and multiplied with the **Column contribution weight** and a factor 0.5 to give the contribution to the total objective for each measured value.

Time Column

To add a **Time Column** subnode, right-click the **Least-Squares Objective** node. Use a **Time Column** to identify a column in the experimental data file as containing the times at which measurements in the value columns were made. When computing the total least-squares objective value, the value column expressions are evaluated at these times in a forward transient solution.

Parameter Column

To add a **Parameter Column** subnode, right-click the **Least-Squares Objective** node. Use a **Parameter Column** to identify a column in the experimental data file as containing the parameter values for which measurements in the value columns were made. When computing the total least-squares objective value, the value column expressions are evaluated for these parameter values. The **Parameter name** has to correspond to one of the global parameters defined in the model.

Coordinate Column

To add a **Coordinate Column** subnode, right-click the **Least-Squares Objective** node. Use a **Coordinate Column** to identify a column in the experimental data file as containing the global coordinates at which measurements in the value columns were made. Select applicable **Coordinate** and **Frame** from the drop-down menus. The number of coordinates must correspond to the number of dimensions in the model.

Ignored Column

To add an **Ignored Column** subnode, right-click the **Least-Squares Objective** node. Use an **Ignored Column** to identify a column in the experimental data file that should not be used.

Integral Objective (Point Sum Objective)

An **Integral Objective** (or **Point Sum Objective** on points) is defined as the integral of a closed-form expression of control and solution variables (the latter are given as the solution to the differential equations defined by the multiphysics model) that are either global or available in the domain in question. Hence, its definition is restricted to a specific set of geometric entities of the same dimension. For integral objectives on points, the integration reduces to a summation.

OBJECTIVE

Enter an **Objective expression** that is integrated over the geometric entity level in the integral objective.

QUADRATURE SETTINGS

Specify the settings for the **Quadrature** used to numerically evaluate the integral in the integral objective: the integration order (default: 4) in the **Integration order** field and the frame to integrate on (default: the spatial frame), which is selected from the **Integrate on frame** list.

Probe Objective

A **Probe Objective** is defined as a point evaluation of a closed-form expression of control and solution variables (the latter are given as the solution to the differential equations defined by the multiphysics model) that are either global or available in the domain in question. The point used for the point evaluation has to be contained in the domain.

OBJECTIVE

Enter an **Objective expression** that is evaluated at the point in the domain.

PROBE COORDINATES

Specify the **Probe coordinates** for the point in the domain where the expression for the objective is evaluated. After specifying the probe coordinates, select an option from the **Evaluate in frame**: **Spatial** (the default), **Material**, or **Mesh**.

Integral Inequality Constraint (Point Sum Inequality Constraint)

Integral Inequality Constraints (**Point Sum Inequality Constraints** on points) specify bounds on the value of the integral of an expression $P_{int}(\xi,u)$ taken over a selected set of geometric entities of the same dimension, Ω:

$$\text{lb}_{\text{int}} \leq \int_{\Omega} P_{\text{int}}(\xi, u) \leq \text{ub}_{\text{int}}$$

The expression is a closed-form expression of control and solution variables (the solution variables are given as the solution to the differential equations defined by the multiphysics model).

For integral inequality constraints on points, the integration reduces to a summation over the selected points:

$$\text{lb}_{\text{int}} \leq \sum P_{\text{int}}(\xi, u) \leq \text{ub}_{\text{int}}$$

CONSTRAINT

Enter a **Constraint expression** that is integrated over the domain in the integral inequality constraint.

QUADRATURE SETTINGS

Specify the settings for the **Quadrature** used to numerically evaluate the integral in the integral objective: the integration order (default: 4) in the **Integration order** field and the frame to integrate on (default: the spatial frame), which is selected from the **Integrate on frame** list.

BOUNDS

By default, the **Lower bound** and **Upper bound** check boxes are selected to activate the required bounds. To specify equality constraints, simply make sure the upper and lower bounds have the same value.

Pointwise Inequality Constraint

A **Pointwise Inequality Constraint** is given as a restriction to the values of a closed-form expression at *all* points in a set of geometric entities of the same dimension. Due to computational issues, the expression has to be a closed-form expression of *only* control variables. Furthermore, only those control variables that are either global or available in the domain in question are usable.

CONSTRAINT

Enter a **Constraint expression** for the pointwise inequality constraint.

This section contains settings for the element used to discretize the control variable. Select a **Shape function type** — **Lagrange** (the default) or **Discontinuous Lagrange**. Also select an **Element order** — **Linear**, **Quadratic** (the default), **Cubic**, **Quartic**, **Quintic**, **Sextic**, or **Septic**.

The **Constraint method** setting controls where the constraints are evaluated:

- Choose **Elemental** to make the software assemble the constraint on each node in each element; that is, there are usually several constraints at the same global coordinates because elements in the computational mesh overlap at nodes.

- Choose **Nodal** (the default) to make the software assemble a single constraint for each global node point. The nodal constraint method provides an averaging of the constraints from adjacent elements.

The default is **Nodal** in order to minimize the number of constraints that must be handled by the optimization solvers.

> The **Constraint method** setting has no effect for **Discontinuous Lagrange** shape functions whose nodes all lie strictly inside the mesh elements.

BOUNDS

By default, the **Lower bound** and **Upper bound** check boxes are selected to activate the required bounds. To specify equality constraints, make sure that the upper and lower bounds have the same value.

Control Variable Field

Add a **Control Variable Field** node to define a control variable which varies as function of position within selected geometric entities (domains, boundaries, edges, or points). The control variable field is discretized using shape functions in the same way as other dependent variables in a multiphysics model. The discrete control variable degrees of freedom, on which the optimization solvers operate represent values at element nodes. Right-click the node to add a **Control Variable Bounds** subnode.

CONTROL VARIABLE

Enter a **Control variable name** and **Initial value**.

Enter a **Scale** indicating a typical magnitude of the control variable. The relative solver tolerances refer to variables rescaled with respect to this scale, and it may also influences the search pattern of some optimization solvers.

DISCRETIZATION

This section contains settings for the shape functions used to discretize control variables. Select a **Shape function type**: **Lagrange** (the default) or **Discontinuous Lagrange**. Also select an **Element order**: **Linear**, **Quadratic** (the default), **Cubic**, **Quartic**, **Quintic**, **Sextic**, or **Septic**.

The default choice of **Value type when using splitting of complex variables** is **Real**. This means that if the solver is set up to split complex variables in real and imaginary parts, no imaginary part is allocated for the control variable field, which is therefore guaranteed to be real. Choose **Complex** to allocate both real and imaginary parts.

Control Variable Bounds

The **Control Variable Bounds** node specifies simple bounds for its parent **Control Variable Field** node. You can only add one single **Control Variable Bounds** node for each **Control Variable Field**.

BOUNDS

By default, both lower and upper bounds are active and set to 0, which constraints the control variable field to be identically zero everywhere. Enter new **Lower bound** and **Upper bound** values to specify an allowed range for the control variable, or deactivate one of the bounds to specify a one-sided bound.

> Bound expressions must be parameter expressions, meaning that they must only contain numbers, model parameters and physical constants. Such expressions can be evaluated to a number independently of the geometry and the multiphysics model solution.

Global Objective

Specify the **Global Objective** contribution to the function. To add this feature, either right-click the **Optimization** interface node and select it from the context menu, or on the **Physics** toolbar, click **Global Objective** (⊚). In some cases, select it from the **Global** submenu.

Enter an **Objective expression** that defines the contribution to the objective function. It can be an expression of those components of the control and solution variables (the solution variables are given as the solution to the differential equations defined by the multiphysics model) that are globally available.

Global Least-Squares Objective

The **Global Least-Squares Objective** is similar to the **Least-Squares Objective** (see Least-Squares Objective) but compares measured data to a globally available model expression. Therefore it does not require any selection, and does not allow any **Coordinate Column** subnodes.

In addition, the **Experimental Data** section contains a **Data source** choice:

- Select **File** (the default) to take the experimental data directly from a file. Click **Browse** to locate and select the file. Add column nodes below the **Least-Squares Objective** to specify the contents of each column in the file. See Least-Squares Objective, Value Column, Time Column, Parameter Column, and Ignored Column for details.

- Select **Result table** to use experimental data from a **Table** node under **Results**. The data may have been generated by another feature under **Results**, or imported into the table feature. In either case, the experimental data will be stored in the COMSOL model file.

- Select **Local table** to enter experimental data directly into a local table in this **Settings** window. Click the **Add** button (+) below the table to add another column. Rows are added automatically as you fill in the first column. To remove a column of data, select some cell in that column and click the **Delete Column** button (![icon]). Similarly, click the **Delete** button (![icon]) to delete the current row. You can also save the definitions of the experimental data to a text file by clicking the **Save to file** button (![icon]). To load a text file with experimental data, use the **Load from file** button (![icon]). The file dialog box allows a number of different file types, both for import and export.

When using experimental data in a **Local table** or from a **Results table**, you must choose from the **Parameter type** list whether each data row in the table corresponds to a **Time** or a **Parameter** value, and in which **Time column** or **Parameter column** the corresponding value is stored. Also specify the **Time unit,** or the **Parameter name** and **Parameter unit**, as appropriate. Finally, fill in the **Model expression** corresponding to the experimental values in each **Data column**, as well as the column **Unit** and **Weight**.

When the data source is **Results table** or **Local table**, you can also select which items in the **Data column** by clearing or selecting the corresponding check box in the **Use** column. You can then, for example, use the same results table to contain both experimental input and output data when performing a time-dependent optimization using a time-dependent load.

To add this feature, either right-click the **Optimization** interface node and select it from the context menu, or on the **Physics** toolbar, click **Global Least-Squares Objective** ($\boxed{e_i^?}$). In some cases, select it from the **Global** submenu.

▥	See *Mooney-Rivlin Curve Fit* for an example of fitting unknown function parameters to measured data using a global least-squares objective: Application Library path **Optimization_Module/ Parameter_Estimation/curve_fit_mooney_rivlin**.

Global Inequality Constraint

Specify a **Global Inequality Constraint** that may involve both control variables and solution variables, as long as the expression is available for evaluation on the global level. To add this feature, either right-click the **Optimization** interface node and select it from the context menu, or on the **Physics** toolbar, click **Global Inequality Constraint** ($\boxed{\leq}$). In some cases, select it from the **Global** submenu.

CONSTRAINT
Enter a globally defined **Constraint expression** whose value is to be constrained.

BOUNDS
By default, the **Lower bound** and **Upper bound** check boxes are selected to activate the required bounds. To specify equality constraints, simply make sure the upper and lower bounds have the same value.

Global Control Variables

Specify those components of the **Global Control Variables** that are globally available. To add this feature, either right-click the **Optimization** interface node and select it from the context menu, or on the **Physics** toolbar, click **Global Control Variables** ($\boxed{Q_i}$). In some cases, select it from the **Global** submenu.

CONTROL VARIABLES

In the table, enter **Variable** names, **Initial values**, and **Lower** and **Upper Bounds** of global control variables. To specify equality constraints, simply make sure the upper and lower bounds have the same value.

Move control variable rows up and down using the **Move up** (⬆) and **Move down** (⬇) buttons. To remove a control variable, select some part of that variable's row in the table and click the **Delete** button (≡). You can also save the definitions of the global control variables to a text file by clicking the **Save to file** button (💾) and using the **Save to File** dialog box that appears. To load a text file with global control variables, use the **Load from file** button (📁) and using the **Load from File** dialog box that appears. Data must be separated by spaces or tabs.

> 💡 If you have LiveLink™ *for* Excel®, you can also click the **Load from Excel File** (📑) and **Save to Excel File** (📑) buttons.
>
> See the *LiveLink™ for Excel® User's Guide* for more information. Or go to http://www.comsol.com/livelink-for-excel/ to learn more about the product.

DISCRETIZATION

To show the **Discretization** section, select **Advanced Physics Options** on the **Show** menu in the **Model Builder** window' toolbar. The default choice of **Value type when using splitting of complex variables** is **Real**. This means that if the solver is set up to split complex variables in real and imaginary parts, no imaginary part is allocated for the control variable field, which is therefore guaranteed to be real. Choose **Complex** to allocate both real and imaginary parts.

CONTROL VARIABLE SCALING

Enter a **Scale** indicating a typical magnitude of the control variables. The relative solver tolerances refer to variables rescaled with respect to this scale, and it may also influences the search pattern of some optimization solvers.

Topology Optimization

Introduction

Topology optimization problems can be solved by relaxing the discrete optimization problem to a continuous optimization problem using a Density Model feature (see below). This feature adds a material volume factor as a control variable in a custom selection. A minimum length scale can be imposed together with common options for projection and interpolation. It is up to the user to use the penalized material volume factor to interpolate material properties; see the *Topology Optimization of an MBB Beam* for an example of topology optimization.

Density Model

The **Density Model** feature ( ) can be added from the model tree under **Component>Definitions** using the **Topology Optimization** context menu. It includes settings for

- Filtering
- Projection
- Interpolation
- Discretization

VARIABLES

The **Density Model** feature adds

- A domain control variable field (`dtopo#.theta_c`) (*control material volume factor*)
- A domain dependent variable (`dtopo#.theta_f`) (*filtered material volume factor*)
- A domain variable (`dtopo#.theta`) (*projected material volume factor*)
- A domain variable (`dtopo#.theta_p`) (*penalized material volume factor*)
- A global variable (`dtopo#.theta_avg`) (*average material volume factor*)

The control variable field is bounded to the interval [0;1]. It is used as input to a Helmholtz filter, which introduces a minimum length scale. The projected material volume factor is fixed to 1 outside the selection for the Density Model feature.

> ℚ For a more extensive introduction to the mathematics implemented by this feature, see Theory for Topology Optimization.

FILTERING

From the **Filter type** list, choose between no filtering (**None**) and a Helmholtz filter (**Helmholtz**; the default), which can impose a minimum length scale by means of a *filter radius* R_{min} (SI unit: m). The local mesh element size h is the default value because this radius should not be smaller than the mesh element size. However, a fixed length scale must be used to get mesh-independent results.

PROJECTION

From the **Projection type** list, specify no projection (**None**; the default) or projection based on the hyperbolic tangent function (**Hyperbolic tangent projection**). When using projection, you can choose the *projection point* and the *projection slope*. Projection with a large slope produces designs almost free of intermediate values, but the optimization problem will be difficult to solve if the slope is too large.

INTERPOLATION

The feature supports, using the **Interpolation type** list, **RAMP** and **SIMP** interpolation for solid mechanics, while **Darcy** interpolation can be used for fluid mechanics. You can also choose a **Linear** interpolation. For solid mechanics, it is common to interpolate the Young's modulus, and the relative void stiffness can be bounded using the *minimum penalized volume fraction*, θ_{min}. The **SIMP exponent** p_{SIMP} and the **RAMP parameter** q_{RAMP} properties determine the stiffness for intermediate design variables. Values in the interval [2;4] produces well defined topologies in the context of volume constrained compliance minimization. For fluid mechanics it is common to introduce a volume damping force, which should be large in the solid regions. The maximum value, however, depends on the fluid viscosity as well as the mesh size, so it is up to the user to specify this. The **Darcy penalization** parameter controls the damping for intermediate design variables and a value of 1 works well in the context of volume

constrained dissipation maximization, but sometimes a continuation starting from a lower value is required to find the global minimum.

> Note that you can use the `dtopo#.theta_avg` variable to impose a volume constraint and the `dtopo#.theta` variable to write a custom material interpolation.

DISCRETIZATION

The **Linear** discretization gives a continuous representation by associating the design variables with the mesh nodes, while an elementwise **Constant** discretization gives a discontinuous representation, which generally gives rise to more variables than the **Linear** discretization.

The **Initial value** θ_0 should be set such that the starting design does not violate any optimization constraints.

4

The Sensitivity Interface

The Sensitivity interface, found under the **Mathematics>Optimization and Sensitivity** branch ( ) when adding an interface, is designed to facilitate setting up and solving sensitivity problems.

To find the sensitivity of a model, add the sensitivity interface along with the physics interfaces in the model. The optimization interface lets you set objective function and to introduce the sensitivity parameters.

In this section:

• The Sensitivity Interface

The Sensitivity Interface

The **Sensitivity (sens)** interface ( ), found under the **Mathematics>Optimization and Sensitivity** ( ) branch when adding a physics interface, provides tools for adding advanced sensitivity evaluation to a stationary or time-dependent model. Basic problems defined only in terms of global scalar objective functions and model parameters can be set up directly in a Sensitivity study step and therefore do not require the use of a Sensitivity interface.

> For a more extensive introduction to the mathematics implemented by this physics interface, see the Theory for the Sensitivity Interface.

The objective functions are defined in terms of control and solution variables (the latter are given as the solution to the differential equations defined by the multiphysics model), which can be fields dependent on position in space or scalar quantities defined globally. This flexibility is reflected in the physics interface by grouping these settings according to the dimension of the domain to which they apply. In such a group of settings, the following settings can be specified, to which each corresponds a separate feature and its Settings window:

- Integral Objective
- Probe Objective
- Control Variable Field

> Note that adding a Sensitivity study step to a study makes it possible to perform a sensitivity analysis directly at the study level. See Sensitivity in the *COMSOL Multiphysics Reference Manual*.

SENSITIVITY TOOLBAR

The following nodes are available from the **Sensitivity** ribbon toolbar (Windows users), **Sensitivity** context menu (Mac or Linux users), or by right-clicking to access the context menu (all users).

> For step-by-step instructions and general documentation descriptions, this is the **Sensitivity** toolbar.

TABLE 4-1: THE SENSITIVITY TOOLBAR

BUTTON OR MENU	NAME
Physics ⚛	
⚛	Add Physics
Global	
◎	Global Objective
ⓠ	Global Control Variables

The main **Settings** window for the **Sensitivity** node contains the following section:

SETTINGS

The **Label** is the default physics interface name.

The **Name** is used primarily as a scope prefix for variables defined by the physics interface. Refer to such physics interface variables in expressions using the pattern <name>.<variable_name>. In order to distinguish between variables belonging to different physics interfaces, the name string must be unique. Only letters, numbers, and underscores (_) are permitted in the **Name** field. The first character must be a letter.

The default **Name** (for the first physics interface in the model) is sens.

Q
- Common Physics Interface and Feature Settings and Nodes in the *COMSOL Multiphysics Reference Manual*
- Global Objective
- Global Control Variables

▦
Sensitivity Analysis of a Communication Mast Detail: Application Library path **COMSOL_Multiphysics/Structural_Mechanics/ mast_diagonal_mounting_sensitivity**

Integral Objective

An **Integral Objective** is defined as the integral of a closed form expression of control and solution variables (the latter are given as the solution to the differential equations defined by the multiphysics model) that are either global or available in the domain in

question. Hence, its definition is restricted to a set of geometric entities of the same dimension. For integral objectives on points, the integration reduces to a summation.

OBJECTIVE

Enter an **Objective expression** that is integrated over the geometric entity level in the integral objective.

QUADRATURE SETTINGS

Specify the settings for the **Quadrature** used to numerically evaluate the integral in the integral objective: the integration order (default: 4) in the **Integration order** field and the frame to integrate on (default: the spatial frame), which is selected from the **Integrate on frame** list.

Probe Objective

A **Probe Objective** is defined as a point evaluation of a closed form expression of control and solution variables (the latter are given as the solution to the differential equations defined by the multiphysics model) that are either global or available in the domain in question. The point used for the point evaluation has to be contained in the domain.

OBJECTIVE

Enter an **Objective expression** that is evaluated at the point in the domain.

PROBE COORDINATES

Specify the **Probe coordinates** for the point in the domain where the expression for the objective is evaluated. After specifying the probe coordinates, select an option from the **Evaluate in frame** — **Spatial** (the default), **Material**, or **Mesh**.

Control Variable Field

Specify the **Control Variable Field** specific to the geometric entity level (domain, edge, boundary, or point) in question.

CONTROL VARIABLE

Enter a **Control variable name** and **Initial value**.

DISCRETIZATION

This section contains settings for the element used to discretize the control variable. Select a **Shape function type**: **Lagrange** (the default) or **Discontinuous Lagrange**. Also

select an **Element order**: **Linear**, **Quadratic** (the default), **Cubic**, **Quartic**, or **Quintic**. The value type (complex or real) for all the variables defined by this **Global Equations** node is selected in the **Value type when using splitting of complex variables** selection. The default value type is **Complex**.

🔍	[Common Physics Interface and Feature Settings and Nodes](#) in the *COMSOL Multiphysics Reference Manual*

Global Objective

Specify the **Global Objective** contribution to the function by entering an objective expression. To add this feature, either right-click the **Sensitivity** interface node and select it from the context menu, or on the **Physics** toolbar, click **Global Objective** (🎯).

OBJECTIVE

Enter an **Objective expression** that defines the contribution to the objective function. It can be an expression of those components of the control and solution variable (the solution variable is given as the solution to the differential equations defined by the multiphysics model) that are globally available.

Global Control Variables

Use the **Global Control Variables** node to specify any globally available control variables. To add this feature, either right-click the **Sensitivity** interface node and select it from the context menu, or on the **Physics** toolbar, click **Global Control Variables** (Qᵢ).

CONTROL VARIABLES

In the table, enter **Variable** names and **Initial values** of the control variables that are globally available. To add a control variable to the table, click the **Add** button (➕). To remove a control variable and its values from the table, click the **Delete** button (🗑).

5

The Optimization Solvers

The Optimization study step is the hub of all optimization tasks. There you can specify which objective functions, control variables, and constraints that are included in the optimization problem, as well as select an optimization solver and set its most important parameters. The Parameter Estimation study step provides a simplified interface for standard parameter estimation tasks.

This chapter describes the Optimization study step and Parameter Estimation study step settings, as well as the theory and detailed settings applying to the individual solvers:

- The Optimization Study
- The Parameter Estimation Study
- About the Optimization Solvers
- The Optimization Solver
- Advanced Solver Properties

The Optimization Study

The **Optimization** study ( ) node collects all settings necessary for solving optimization problems. It serves the dual purpose of defining the optimization problem to be solved and choosing an optimization solver, as well as controlling important solver properties and solver output.

The complete optimization problem can be set up directly in the **Optimization** study node when the objective function to be minimized or maximized is a global scalar expression and the only control variables to be varied are already defined as model parameters. If the model requires control variables or objective functions which depend on position in the geometry, general least-squares objective contributions or pointwise constraints, these must be set up separately using an Optimization interface. Such contributions are displayed in the **Optimization** study node settings where they can be individually disabled or enabled.

When you add an **Optimization** study node to a study, it is always inserted as the first node, at the top of the study sequence. Remaining nodes in the sequence define the multiphysics problem on which the optimization process will act. If this sequence contains **Study Reference** nodes and you are using a derivative-free solver, you can choose for each objective function contribution and constraint whether to compute its value for the main sequence or for the sequence pointed to by one of the study references.

	A derivative-free optimization solver can be combined with a parametric sweep, but in general it is only possible to use one Sensitivity, Optimization, Parameter Estimation, or Parametric Sweep study step in each study. These study nodes tend to control the same solver settings and are therefore incompatible with each other. To perform parametric or nested optimization, you can call a study containing an Optimization node from inside another study, via a Study Reference node.

	This section describes the Optimization study node available with the Optimization Module. See also Studies and Solvers in the *COMSOL Multiphysics Reference Manual* for more information about solvers in general. In this guide, see About the Optimization Solvers for details on the capability and settings of the individual optimization solvers.

Expressions for user-defined objective and constraint functions are evaluated in the global namespace, while the expressions for physics-defined objective and constraint expressions are evaluated in the component namespace. If there are a global parameter or function and a component variable or function with the same name (for example, `par` or func: `root.par` and `compi.par`, `root.func`, and `compi.func`) and a user-defined objective function or constraint with expression `f(par)` or `func`, it is evaluated as `f(root.par)` and `root.func`, while a physics-defined objective or constraint with expression `f(par)` or `func` is evaluated as `f(compi.par)` or `compi.func`. Note that the evaluation of the expressions of the user-defined objective and constraint functions in the Optimization study differs from the evaluation in Results (where the expressions are evaluated in the component namespace).

The **Settings** window has the following sections:

OPTIMIZATION SOLVER

Select an optimization solver and specify its most important parameters.

Method

Choose an optimization solver method. The list of available solvers contains both gradient-based and derivative-free methods.

- Derivative-free (gradient-free) optimization options: **Coordinate search**, **Monte Carlo**, **Nelder-Mead** (the default), **BOBYQA**, and **COBYLA**.

- Gradient-based optimization options: **SNOPT**, **MMA**, and **Levenberg-Marquardt**.

The different solvers are more or less suitable for different types of optimization problems. There are also differences in which problem features they can handle. Objective contributions, control variables and constraints which are not compatible with the selected solver are marked by a warning sign (⚠) in the first column of the corresponding table.

Optimality Tolerance

Specify the relative **Optimality tolerance**. The value is applied relative to each control variable after scaling with its corresponding specified scale. The default value of this setting varies depending on the selected optimization **Method**.

Study Step

The gradient-based solvers can optimize only over the output from a single study step, and not for all study step types. Choose any of the available study steps from the **Study step** list. Click the **Go to Source** button ⧉) to move to the **Settings** window for the selected study node.

Maximum Number of Model Evaluations

Specify the **Maximum number of model evaluations**. The default is 1000. This number limits the number of times the objective function is evaluated, which is related to the number of times the multiphysics system is simulated for different values of the optimization control parameter.

📋	The number of objective evaluations is not equal to the number of iterations taken by the optimizer because each iteration can invoke more than a single objective function evaluation.

Distribute Parametric Sweep

If you are using a solver that supports batch evaluation of objective values, you can choose to distribute computations over the nodes in a cluster. To enable this functionality, first click the **Show** button (👁) and select **Advanced Study Options**, then select the **Distribute parametric sweep** check box.

Least-Squares Time/Parameter Method

If least-squares objectives are defined, you can specify the **Least-squares time/parameter method**. The default is **Manual**. In that case, all least-squares defined time or parameter values are merged with the time or parameter values defined in general parameter value lists. The other option is **From least-squares objective**. In that case, only least-squares defined time or parameter values are used and all other time or parameter values are disregarded.

OBJECTIVE FUNCTION

You specify the objective function for the optimization problem in the table's **Expression** column. Enter any globally available expression that evaluates to a real number. Optionally, you can add a description in the **Description** column. Click the **Add**

Expression (➕) and **Replace Expression** (▶) buttons to search through a list of predefined expressions.

> 💡 For an optimization objective that is expressed in terms of the solution *u* of a PDE, Integration (described in *COMSOL Multiphysics Reference Manual*) is one example of how you can define a scalar objective as required by the optimization solver. The evaluation of the objective function is similar to Global Variable Probe (described in *COMSOL Multiphysics Reference Manual*), so any variable that can be represented by a global variable probe is suitable as an objective.

The **Evaluate for** column specifies the study step for which each objective contribution will be evaluated. Available options are the last study step in the sequence plus all Study Reference nodes. When a gradient-based solver is used, the column is not active: all contributions are then evaluated for the same study step.

If there is an **Optimization** or **Sensitivity** interface in the model, containing objective function nodes, these show up in a separate table under **Objective Function**. Use the **Active** column to deactivate individual contributions as needed. Contributions not supported by the currently selected solver are marked by a warning sign (⚠) in the first column, which disappears if you manually deactivate the objective.

Type
Select whether to perform a **Minimization** or a **Maximization** of the objective function. The default is to minimize the objective function.

Multiple objectives
If you have defined more than one objective function, choose how to evaluate the overall objective: Choose from **Sum of objectives** (the default), **Minimum of objectives**, or **Maximum of objectives**. Note that not all options are available together with all solvers.

Solution
Here you select which solution or solutions to use for evaluating the objective function when several solutions are present, like for Time Dependent or Eigenvalue studies. The possible choices are **Auto** (the default), **Use first**, **Use last**, **Sum of objectives**, **Minimum of objectives**, and **Maximum of objectives**. Note that the last three options first evaluate multiple objectives according to the **Multiple objectives** setting for each solution individually, and then compute the sum, minimum or maximum of the individual results.

All options are available only with the derivative-free solvers. When using **SNOPT** or **Levenberg-Marquardt**, **Auto** is the only choice. **MMA** supports all options except when optimizing over a **Time Dependent** study step.

With **Auto** selected, the solver chooses the evaluation method based on the innermost study. For the studies of Eigenvalue, Eigenfrequency, or Linear Buckling type (all described in *COMSOL Multiphysics Reference Manual*), the first solution corresponding to the smallest eigenvalue is chosen. For studies with parametric solutions (for example, studies in the Frequency Domain), the contributions from all solutions are summed (equivalent to the **Sum of objectives** option). For all other study types, the optimization solver selects the last solution, like the solution at the final time for a Time Dependent problem.

Outer solution
When there is a parametric sweep below the optimizations study step, the objective is evaluated using the **Outer solutions** from the parametric sweep. The options are between the **Minimum of objectives**, **Maximum of objectives** or **Sum of objectives** (the default).

CONTROL VARIABLES AND PARAMETERS

The first table under **Control Variables and Parameters** is used to select model parameters for use as control variables. Click the **Add** (➕) button to add one of the parameters defined in the **Settings** window for **Parameters** under **Global Definitions** to the set of control variables.

From a list in the **Parameter name** column, select the parameter to redefine as a control variable. Specify an **Initial value** for the control variables you add. The initial value is used as initial guess in the optimization solver and the objective function is explored around this point.

The **Scale** column is important. Each control variable is rescaled with its specified scale. This means, in practice, that the solvers only get to see the control variables divided by their corresponding scale, and it is on these rescaled variables that all tolerances are applied — both user-defined and internal tolerances intended to ensure the stability of the optimization methods. The default value is 1, which makes the solver work with the original, unscaled, variables.

Use the **Lower bound** and **Upper bound** columns to add lower and upper bounds to the control variables. The Optimization solver only evaluates the objective function within

these bounds. If you do not want to set bounds on a variable, leave the cell in the table empty.

	The Optimization solver determines whether bounds are allowed or not. The Monte Carlo solver can only be run when both a lower and an upper bound are given; the MMA solver also needs bounds but can estimate them automatically — but at a cost.

Move control parameter rows up and down using the **Move up** (↑) and **Move down** (↓) buttons. To remove a control parameter, select some part of that variable's row in the table and click the **Delete** button (▤). You can also save the definitions of the control parameters to a text file by clicking the **Save to file** button (🖫) and using the **Save to File** dialog box that appears. To load a text file with control parameters, use the **Load from file** button (📂) and using the **Load from File** dialog box that appears. Data must be separated by spaces or tabs.

	If you have the LiveLink™ *for* Excel®, you can also save and load control parameters to and from Microsoft Excel Workbook (*.xlsx) files.

If there is an **Optimization** or **Sensitivity** interface in the model, containing control variable nodes, these show up in a separate table under **Control Variables and Parameters**. Use the **Solve for** column to deactivate individual control variables as needed. Variables not supported by the currently selected solver are marked by a warning sign (⚠) in the first column, which disappears if you manually choose not to include it in the solution.

CONSTRAINTS

The first table under **Constraints** lets you specify additional constraints to be imposed on the optimum solution. The **Expression** column accepts any globally available expression which evaluates to a real number. Constraints can be functions of the control variables both directly and indirectly via PDE solution variables. The **Lower bound** and **Upper bound** columns can only contain *parameter expressions*; they must not depend on control variables, PDE solution variables, or any other user-defined variables, but can include model parameters, physical constants and units. One bound column can be left blank to indicate that no upper or lower bound is required.

The **Evaluate for** column specifies the study step for which each constraint will be evaluated. Available options are the last study step in the sequence plus all Study

Reference nodes. When a gradient-based solver is used, the column is not active: all contributions are then evaluated for the same study step.

If there is an **Optimization** interface in the model, containing inequality constraint nodes, these also show up in a separate table under **Constraints**. Use the **Active** column to deactivate individual constraints as needed. Constraints not supported by the currently selected solver are marked by a warning sign (⚠) in the first column, which disappears if you manually choose to deactivate the constraint.

For some optimization solvers, you can select a **Constraint handling method**. The possible options are **Penalty** and **Augmented Lagrangian**. The former is the default when available and requires no further settings. Choosing the **Augmented Lagrangian** method activates additional options to control its behavior and accuracy. See Constraint Handling for Derivative-Free Methods for further details.

OUTPUT WHILE SOLVING

Plot
Select **Plot** and choose a **Plot group** to update after each major iteration of the optimization algorithm.

Probes
Select which **Probes** to evaluate and plot in each iteration.

Keep objective values in table
Select the **Keep objective values in table** check box to retain the table containing control variable and objective function values after the solver completes. Choose an existing **Output table**, or select **New** to create a new table. After computing the study, the **Output table** setting will be changed to the table actually being used.

The derivative-free solvers add a new line to this table for each evaluation of the objective function, while the gradient-based solvers add a new line for every accepted step. When using derivative-free methods, the control variable values are also displayed in the table. Objective values displayed on the last line of the table are the converged result of the optimization when the **Nelder-Mead**, **BOBYQA**, **Coordinate search**, **MMA**, **Levenberg-Marquardt**, or **SNOPT** solver is used.

Select the **Show individual objective values** check box to include one table column for each contribution to the objective. Otherwise, only the total objective is displayed.

Select the **Table graph** check box to plot the objective function values displayed in the objective table. Choose an existing **Plot window**, including the standard **Graphics**

window, or select **New window**. After computing the study, the **Plot window** setting will be changed to the window actually being used.

Keep constraint values in table
Only available for derivate-free methods. Select the **Keep constraint values in table** check box to retain the table containing global constraint values after the solver completes. Choose an existing **Constraint table**, or select **New** to create a new table. After computing the study, the **Constraint table** setting will be changed to the table actually being used.

The derivative-free solvers add a new line to this table for each evaluation of the objective function. When using the **Nelder-Mead**, **BOBYQA**, or **Coordinate search** solver, the control variable and constraint values displayed on the last line correspond to the converged result of the optimization.

Optimization log
Choose **Minimal**, **Normal** (the default) or **Detailed** to control the amount of information output from the optimization solver and the inner solvers it calls.

Memory settings for jobs
These settings are only available for the derivative-free optimization methods: **Coordinate search**, **Monte Carlo**, **Nelder-Mead**, **BOBYQA**, and **COBYLA**.

You can use the **Keep solutions in memory** list to control how to store the solutions from the individual optimization solutions. Select **All** to store all the optimizations solutions in memory, or select **Only last** or **Automatic** (the default) to store only the last solution from the optimization job. If the there are **Parametric Sweep** study steps in an **Optimization** study, the default option (**Automatic**) will save the last solution from the optimization job and all solutions from the parametric jobs.

If you want the optimization job to stop at once if an error occurs, select the **Stop if error** check box.

The Parameter Estimation Study

The **Parameter Estimation** ( ) study node provides a simplified interface for performing least-squares parameter estimation. It can be used when the reference data is a function of time or a single parameter, and the multiphysics model result expected to match the data is a single global expression evaluated for a selected study step in the same study.

When the reference data consists of measured values, you typically first import it as an interpolation function which you can easily plot and visually compare to the result of your multiphysics model. The Parameter Estimation study node can refer directly to the interpolation function and is independent of whether the interpolation function was specified directly in the user interface or imported from a file, and is also independent of the file format used.

Alternately, the reference data can be given as a user-defined expression which is evaluated at the time steps or parameter values specified in the corresponding study step. This is useful, for example, when estimating coefficients of a polynomial expected to match the model output, and in general when estimating parameters in a mathematical model intended to replicate the output of the full multiphysics model.

The parameter estimation problem is implemented as an optimization problem minimizing a sum of squared differences between model and reference data. The sum is computed over time or parameter steps as specified either in the argument column of an interpolation function used as reference data, or in the study step selected for evaluation of the model data.

MODEL DATA

Choose one of the allowed study steps from the **Study step** list. Click the **Go to Source** button) to move to the **Settings** window for the selected study node. Enter a global **Model expression** which is evaluated at each time or parameter step, where it is compared to the corresponding reference data. You can also click the **Add Expression** ( ) and **Replace Expression** ( ) buttons to search through a list of predefined expressions.

REFERENCE DATA

Select a **Reference data source**: **Interpolation function** or **User defined**. If you choose an interpolation function as data source, its argument column will decide at which time or parameter values the difference between model and reference data is evaluated.

When the data source is set to **User defined**, enter a **Reference expression** or click the **Replace Expression** (![icon]) buttons to search through a list of predefined expressions. You can also press Ctrl+Space to add a predefined expression to the text field. The difference between the **Model expression** and the **Reference expression** is then evaluated at output time steps or parameter values as specified in the selected **Study step**.

PARAMETERS

Click the **Add** (![icon]) button to add one of the parameters defined in the **Settings** window for **Parameters** under **Global Definitions** to the set of parameters to be estimated. Use the **Move Up** (![icon]), **Move Down** (![icon]), and **Delete** (![icon]) buttons under the table to organize the data.

From a list in the **Parameter name** column, select one of the available global parameters. Specify an **Initial value** which is used as initial guess when estimating the parameter.

The **Scale** column is important. Each control variable is rescaled with its specified scale. This means, in practice, that the solvers only get to see the parameters divided by their corresponding scale, and it is on these rescaled variables that all tolerances are applied — both user-defined and internal tolerances intended to ensure the stability of the optimization methods. The default value is 1, which makes the solver work with the original, unscaled, parameters.

Use the **Lower bound** and **Upper bound** columns to set lower and upper bounds on the parameters. The Optimization solver only evaluates the objective function within these bounds. When doing parameter estimation, bounds are typically not needed and the cells can therefore be left empty.

> If you choose to add bounds to help the solver, make sure to check the solution afterward. If any of the estimated parameters has reached its bound value, then the bound should be relaxed or the parameter should be eliminated from the problem. In both cases, it is necessary to solve the modified problem one more time.

An alternative to specifying parameter names and values directly in the table is to specify them in a text file. Use the **Load from File** button (![icon]) to browse to such a text file. The program appends the read names and values to the current table. The format of the text file must be such that the parameter names appear in the first column and the values for each parameter appear row-wise with a space separating the name and values, and a space separating the values. Click the **Save to File** button (![icon]) to save the

contents of the table to a text file (or to a Microsoft Excel Workbook spreadsheet if the license includes LiveLink™ *for* Excel®).

PARAMETER ESTIMATION METHOD

Select an optimization **Method** — **BOBYQA** (the default), **Levenberg-Marquardt**, or **SNOPT** — to solve the parameter estimation problem. Choose **BOBYQA** when the parameters to be estimated control the geometry, mesh or any other aspect of the model which is not represented as a term in the model equations. Otherwise try **Levenberg-Marquardt**, which is generally faster than SNOPT but does not support bounds on the parameters.

Specify the relative **Optimality tolerance**. The value is applied on rescaled variables, using the scale specified for each parameter. The default value of this setting varies depending on the selected optimization **Method**. The **Maximum number of objective evaluations** limits the number of times the objective function is evaluated, which is related to the number of times the multiphysics system is simulated for different attempted values of the parameters.

> 📝 The **Least-squares time/parameter method** list only appears in a model where there is a least-squares objective.

Set the **Least-squares time/parameter method** to choose how the parameter estimation defined times or parameters should be used. If you choose **Manual** (the default), all parameter estimation defined time or parameter values are merged with the time or parameter values defined in general parameter value lists. If **From least-squares objective** is chosen, only time or parameter values defined by the parameter estimation are used and all other time or parameter values are disregarded.

OUTPUT WHILE SOLVING

The settings in the Output While Solving section are identical to the corresponding settings in the Optimization study step. See Output While Solving under The Optimization Study.

About the Optimization Solvers

The Optimization Module provides a selection of optimization solver algorithms which can be divides into two main groups: on one hand *gradient-based* solvers and on the other hand *derivative-free* solvers. The two groups are suitable for different types of problems and have different performance characteristics.

In this section:

About Derivative-Free Solvers

The defining characteristic of the derivative free solvers is that they do not need to compute derivatives of the objective function with respect to the control variables. They do not even require the objective function to be differentiable in principle. This makes them suitable for problems where the objective function is non-smooth or contains noise.

One typical example of a noisy objective function is when the control variables define geometry dimensions. The geometry changes induced by modifying the control variables then lead to different finite element meshes, superimposing different discretization errors on the objective function when evaluated for different control variable values.

Since the derivative-free solvers do not trust the pointwise behavior of the objective function to be a good indicator of where to search for the next, improved, update to the control variables, they must rely on sampling the objective function at different positions in the control variable space. This is more expensive than following a single path toward the optimum but also more robust. Some of the performance penalty is offset by the fact that evaluations that do not depend on one another can be done in parallel — for example, in a cluster environment.

Finally, derivative-free solvers can be further subdivided into *local*, "hill-climbing," methods and *global*, evolutionary or statistical, methods. The former type starts from an initial guess and strives to improve the objective function in a stepwise manner. Imagine a group of people trying to climb a hill together in dense fog; as long as they stay together and move upward, they find a top but not necessarily the highest one. Global methods, in contrast, try to produce a map of the entire design space, refining it iteratively in areas that appear to be good candidates for containing the global optimum.

The Optimization Module provides five different derivative-free algorithms:

- The **Coordinate search** solver aims at improving the objective function along the coordinate directions of the control variable space. See The Coordinate Search Solver.
- The **Monte Carlo** solver samples points randomly with uniform distribution inside a box specified by the user. See The Monte Carlo Solver.
- The **Nelder-Mead** solver walks toward improved objective function values by iteratively replacing the worst corner of a simplex in the control variable space. See The Nelder-Mead Solver.
- The **BOBYQA** solver walks toward improved objective function values by using an iteratively constructed quadratic approximation of the objective. See The BOBYQA Solver.
- The **COBYLA** solver solves a sequence of linear approximations constructed from objective and constraint values sampled at the corners of a simplex in control variable space. See The COBYLA Solver.

These methods are each described in more details below.

About Gradient-Based Solvers

The defining characteristic of a gradient-based solver is that follows a path in the control variable space where each new iterate is based on local derivative information

evaluated at previously visited points. The methods implemented in the Optimization Module require the complete vector of first-order derivatives of the objective function with respect to the discrete vector of control variable degrees of freedom, which is referred to as the discrete *gradient* of the objective function in the control variable space.

The gradient can be computed in different ways. In general, the **Adjoint** method is the most efficient (and also the default), followed by the **Forward** method. The pure **Numeric** method is the most expensive as it is based on repeated solution of the multiphysics problem, while the **Forward numeric** method requires only repeated assembly of the problem residual.

The Optimization module provides three different gradient-based algorithms:

• The **SNOPT** solver is a general purpose solver suitable for dealing with large-scale problems with many or difficult constraints. See The SNOPT Solver.

• The **MMA** solver can handle problems of any form and is especially suitable for problems with a large number of control variables, such as topology optimization. See The MMA Solver.

• The **Levenberg-Marquardt** solver is specifically designed for solving least-squares problems. See The Levenberg-Marquardt Solver.

These methods are each described in more details below.

The Coordinate Search Solver

The Coordinate search solver aims at improving the objective function along the coordinate directions of the control parameter space. The step lengths are decreased or increased according to the values of the objective function. The Coordinate search solver does not directly evaluate gradients of the objective function. Gradients are not available for all types of parameters or might not be mathematically well-defined in certain circumstances. One example is an objective function that contains noise.

However, when the solver has collected enough information around the current search point, an estimate of the gradient is constructed and a line search along this direction is attempted before a new evaluation along the coordinate directions. This accelerates the search procedure, in particular for points close to (local) minima. The algorithm is based on the description in Ch. 7 in Ref. 1.

The Monte Carlo Solver

The Monte Carlo solver samples points randomly with uniform distribution inside a box specified by the user. This solver is slow for finding accurate values of a minimizer of the objective function; however, it is useful for gathering statistical data of design variations by analyzing the range of values the objective function takes. As compared to the other optimization algorithms implemented in COMSOL Multiphysics, it does not get stuck in local minima. It always explores the whole search space specified by the parameter bounds.

The generation of random numbers in the **Monte Carlo** solver is controlled by the value of the **Random seed**. If the check box is cleared, the random number generator is initialized by a number based on the current system time. In this case, two runs produce in general different sets of parameters during operation. If a seed is given, the parameter selection is random during the operation of the solver but produces the same sequence of numbers from one run of the optimization solver to the next.

The Nelder-Mead Solver

The Nelder-Mead solver relies on a simplex of $N+1$ points, where N is the number of control variables. The solver does not use derivatives of the objective function. In a Nelder-Mead iteration, the solver uses reflections, expansions, and contractions to improve the worst point in the simplex.

The implementation in COMSOL Multiphysics includes a restart procedure for the case when the simplex shape degenerates (that is, the simplex collapses along a direction). Moreover, the solver respects lower and upper bounds in the control variable space by suitably restricting the length of reflections. The implementation follows the discussion of the Nelder-Mead method in Ch. 8 in Ref. 1, employing the parallelization strategy described in Ref. 2.

The BOBYQA Solver

The name BOBYQA is an acronym for *Bound Optimization by Quadratic Approximation*. The basic idea of the method is to iteratively approximate the objective function by a quadratic model which is valid in a region around the current iterate, the so-called *trust region*. The quadratic model is updated by minimizing the Frobenius norm of the difference in the Hessians of the two consecutive quadratic

approximations. The implementation in COMSOL is based on Ref. 3. There are, however, some modifications:

- The number of interpolation conditions is fixed to $2n+1$ where n is the number of control variables, since the updating of the quadratic approximation requires only $O(n^2)$ operations in that case.

- Since the COMSOL implementation works in scaled control variables, the initial trust region radius is fixed at 0.2 in relative terms.

- Subroutine RESCUE is not included due to the unavailability of test problems that would invoke that procedure. Instead, an error message is given.

The COBYLA Solver

The name COBYLA is an acronym for Constrained Optimization by Linear Approximation. It is an iterative method for derivative-free constrained optimization. The method maintains and updates linear approximations to both the objective function and to each constraint. The approximations are based on objective and constraint values computed at the vertices of a well-formed simplex. Each iteration solves a linear programming problem inside a *trust region* whose radius decreases as the method progresses toward a constrained optimum. Further details can be found in Ref. 4.

The SNOPT Solver

The SNOPT solver uses a gradient-based optimization technique to find optimal solutions to a very general class of optimization problems. It requires gradients of both the objective function and all constraints, which can either be computed externally (analytically or semi-numerically) or internally, using numeric differentiation.

The underlying algorithm is an implementation of sequential quadratic programming (SQP). This means that SNOPT solves a sequence of approximations to the original problem, where the objective function is assumed to be a quadratic polynomial and constraints are treated as linear. Steps in this sequence are referred to as *major* or *outer* iterations. Each approximate quadratic programming (QP) problem is also solved iteratively, requiring a number of *minor* or *inner* iterations. The QP solver returns a step direction to the outer SQP algorithm, which decides on a step length and updates the QP approximation before proceeding to the next major iteration.

The overall structure of SNOPT as well as the default QP solver (Cholesky) assume that optimal solutions are more likely to be found at corners of the feasible set, bound

by constraints, than in the interior of the feasible set. Therefore, the method performs best on problems with many active constraints relative to the number of control variable degrees of freedom, such that close to the optimum, most degrees of freedom are bound by constraints and only a few are free. Such free degrees of freedom are referred to as *superbasic variables*. When the number of such superbasic variables becomes large, the full Cholesky factorization-based QP-solver is unsuitable. Instead one of the iterative, *conjugate gradients* or *quasi-Newton*, methods should be selected instead. For details, see Ref. 5 and Ref. 6.

COMMAND-LINE OPTIONS

SNOPT can optionally output diagnostic information to a file. The file contents and format are described in Ref. 5. To turn this functionality on, set the following command-line options when starting COMSOL Multiphysics:

```
-cs.snoptprintdir <dir> -cs.snoptprintfile <filename>
```

where `<dir>` is the desired output directory and `<filename>` is the filename. You can also specify the same options in the applicable INI file as

```
-Dcs.snoptprintdir=<dir>
-Dcs.snoptprintfile=<filename>
```

See The COMSOL Commands in the *COMSOL Multiphysics Reference Manual* for further information about setting options when starting COMSOL Multiphysics.

The MMA Solver

The MMA implementation in the Optimization Module is the globally convergent version of the method of moving asymptotes, referred to as GCMMA in Ref. 7.

MMA Method of Moving Asymptotes, GCMMA, Globally Convergent MMA, and Globally Convergent Method of Moving Asymptotes authored by Krister Svanberg. Copyright © 2013 Krister Svanberg.

This is a three-level algorithm:

- Outer iteration k uses the current control variable estimate, x_k, to evaluate objective function, constraints and their gradients, which are used together with current asymptote estimates, l_k and u_k, to construct an approximating subproblem. This subproblem, which is guaranteed to be convex and feasible, is passed to the inner iterations.

- Each inner iteration m solves an approximating subproblem for its unique optimum x_{km} and then evaluates the true objective function and constraints at this point. If the approximating subproblem is found to be conservative compared to the true function values, the inner iteration is terminated and the point is accepted as the next outer estimate x_{k+1}. Otherwise, the approximating subproblem is modified to make it more conservative and then passed to the next inner iteration.

- The subproblem in each inner iteration is solved using a dual active set strategy. The approximating subproblems are nonlinear and inequality-constrained, but have a special structure which makes solving the primal problem for fixed dual variables very fast. From the solution to the primal problem, a gradient and full Hessian can be computed for the dual problem, which is solved using a modified Newton active set algorithm.

Note that function (objective and constraints) gradients are computed strictly once in each outer iteration, while function values must be computed once for each extra inner iteration required. The innermost level sees only an analytical approximating form of the subproblem where current function and gradient estimates appear in various coefficients.

The special structure of the generated approximating subproblems influences the global behavior of the algorithm. In contrast to the SNOPT and Levenberg-Marquardt solvers, which rely on approximating second-order information about the objective function, MMA is essentially a linear method. Its subproblems are linear approximations to the original problem but with barrier-like rational function contributions controlled by the moving asymptotes. No information about the problem is retained between outer iterations except the current position of the asymptotes.

In practice, this means that MMA does not show the quadratic convergence close to the optimum associated with Newton-like methods. In fact, there are very simple problems dominated by a quadratic term in the objective function for which MMA converges very slowly or not at all. In particular, in order for MMA to work efficiently, least-squares problems must be formulated using Least Squares Objective features in an Optimization interface. These features trigger a reformulation of the problem to a form that is more suitable for MMA. Moreover, if the defect is complex valued, only the real part is used by MMA internally.

Because of the linear approximation of the objective function, the first inner iteration in each outer MMA iteration effectively steps into a corner of the feasible set, where it is completely bound by constraints and simple bounds. If this point is found to be

nonconservative, as is the case if the objective function is convex with an optimum in the interior of the feasible set, the inner iteration generates a series of iterates gradually moving away from the constraints until a conservative point is found. This behavior favors points close to the constraints, in contrast to the line search used in SNOPT and the trust region in Levenberg-Marquardt which favor points close to the previous iterate. If the objective function has multiple local minima, the different methods can therefore be expected to find different local solutions.

For further details, see Ref. 7, which you can find under `<COMSOL_root>/doc/pdf/Optimization_Module/gcmma.pdf`, where `<COMSOL_root>` is the root folder of your COMSOL installation.

The Levenberg-Marquardt Solver

The Levenberg-Marquardt solver works exclusively with objective functions of least-squares type. Constraints are not supported. Because this method is designed specifically for solving problems of least-squares type, it typically converges faster than SNOPT for such problems. The objective function is

$$V(\eta) = \frac{1}{2} \sum_{m=1}^{M} \sum_{j=1}^{J_m} \sum_{k=1}^{K_{jm}} w_{jm} f_{jm}^2(x_{jmk}, u_m(x, p_{jm}, \eta), \eta, C_m) \qquad (5\text{-}1)$$

where M is the number of *series* (measurement series), J_m is the number of measurements, and K_{jm} is the number of points. The variable x is the space coordinates, η are the parameters for which the cost function should be minimized and $u_m(x, p, \eta)$ solves a given PDE or ODE. The variable p is time if the PDE or ODE is time-dependent but it can also represent any parameter when the forward problem is stationary. The functions w_{jm} are weight functions, and f_{jm} represent the difference between some model function g_{jm} and some measured data g_{jmk}; that is, f_{jm} can be written as

$$f_{jm}(x_{jmk}, u_m(x, p_{jm}, \eta), \eta, C_m) = g_{jm}(x_{jmk}, u_m(x, p_{jm}, \eta), \eta, C_m) - \hat{g}_{jmk} \qquad (5\text{-}2)$$

The Levenberg-Marquardt algorithm as implemented in the Optimization Module relies on two fundamental ideas: evaluation of an approximate Hessian and regularization of the Hessian approximation. The special structure of least-squares objective functions allows cheap evaluation of an approximate Hessian (matrix of second derivatives), which can in principle be used directly in a Newton iteration. However, least-squares problems are also often ill-conditioned, making the full

Newton process unstable. Therefore, the Hessian is modified using a regularization parameter to guarantee its positive definiteness. This parameter is updated between iterations, based on the success or failure of the previous step. For further details, see Ref. 8

About Optimality Tolerances

The optimality tolerance is an important setting for all optimization solvers. It is intended to represent the relative accuracy in the final scaled control variable values, but because of the wide differences between different solver implementations, uniform behavior cannot be guaranteed.

In particular, the optimality tolerance can play tricks on you if your objective function or your optimization variables are badly scaled. Therefore, take care to specify correct scales for your control variables and make sure that objective functions and constraints are of order 1 — or at least not too far from — for reasonable values of the control variables.

Tweaking the **Optimality tolerance** parameter might be necessary if you are confronted with problems related to convergence. As an example, if the optimization solver reports a converged solution after just a few iterations, try to restart it with a tighter tolerance to make sure it has actually found the solution. If, on the contrary, it seems to iterate forever — despite the value of the objective function having converged (check the output on the **Log** page in the **Progress** window) — chances are that the tolerance value is too strict.

OPTIMALITY TOLERANCE FOR DERIVATIVE-FREE METHODS

For the derivative-free optimization methods, the optimization tolerance, with a default value of 0.01, is used to determine whether a stationary point has been reached. The **Coordinate search**, **BOBYQA**, **COBYLA**, and **Nelder-Mead** methods stop iterating as soon as no improvement over the current best estimate can be found with steps in the scaled control variables of relative size larger than or equal to the optimality tolerance. For the **Monte Carlo** solver, the iteration stops when a new sampling point improves the objective function but is within the optimality tolerance to the previous best point.

Compared to gradient-based optimization methods, which improve based upon the gradient of the objective function with respect to control variables, derivative-free

methods explore the region around the current point by function evaluations only and use that information for determining convergence.

The returned point is not necessarily located close to a stationary point to within the optimality tolerance. When problems are badly scaled or functions are nonsmooth (for example, because of noise in the objective evaluation), the algorithms might miss an opportunity for improvement that requires an absolute step length larger than the optimality tolerance times the specified scale. Also, the Monte Carlo solver does not explore all directions systematically but rather determines convergence based on one randomly sampled point only. In case of convergence problems, try to reduce the optimality tolerance, or choose a different initial condition.

OPTIMALITY TOLERANCE FOR SNOPT

For SNOPT, the optimality tolerance parameter (corresponding to the *major optimality tolerance* in Ref. 5 and further explained together with parameter Opttol), with a default of $1.0 \cdot 10^{-3}$, is used by the linear and quadratic solvers to determine, on the basis of the reduced-gradient size, whether optimality has been reached. More precisely, it regulates the accuracy to which the final iterate in SNOPT is required to fulfill the first-order conditions for optimality.

When SNOPT cannot achieve the requested tolerance level, the solver eventually returns a solution together with a warning message as follows:

- The warning message "requested accuracy could not be achieved" refers to the case when a feasible solution has been found, but the requested accuracy cannot be achieved. Hence, an abnormal termination has occurred, but the solver is within good reach of satisfying the **Optimality tolerance**. If this happens, check that the **Optimality tolerance** is not too small.

- The warning message "the current point cannot be improved upon" can occur in cases when the objective or constraint evaluation requires an iterative process which is terminated as soon as a given tolerance is achieved, or when the function evaluation contains some other source of noise. In such case the evaluation might be accurate to rather few significant digits, and gradients are probably unreliable.

   Theoretically the **Optimality tolerance** should not be set smaller than the square-root of the *function precision*. The latter is the expected stability of the numerical model rather than its accuracy as a model of physical reality. When using a direct linear solver on a linear model, the function precision is generally of the same order as the

inverse of the condition number. For a nonlinear or iterative solver, you can expect the precision to be of the same order as the solver tolerances, which is then also the numerical precision in the evaluation of the objective and constraints.

Furthermore, even when you set the **Optimality tolerance** based on the function precision, the same exit condition might occur. At present, the only remedy is to increase the accuracy of the function calculation, using all available means.

The final SNOPT iterate is not guaranteed to be a constrained local minimizer despite a successful run. For example, the constraint qualification might not hold at the final iterate. Similarly, the final iterate might satisfy the first-order but not the second-order conditions for optimality. Verifying second-order conditions requires second derivatives. See section 2.11 in Ref. 6 and p. 76 of the *SNOPT User's Guide* (Ref. 5) for further details.

OPTIMALITY TOLERANCE FOR MMA

The MMA solver terminates when the relative change in all scaled control variables is less than the specified optimality tolerance parameter, with a default of **1e-3**. The relative change is defined as the change in the variable since the last outer iteration divided by the range of the variable. The range of the variable is the upper bound minus the lower bound. For unbounded variables, the MMA solver internally estimates bounds based on the previous iteration points.

OPTIMALITY TOLERANCE FOR LEVENBERG-MARQUARDT

Let $\mathbf{tol}$ be the specified optimality tolerance. Define $\mathbf{tol}_d = \gamma_d \cdot \mathbf{tol}$, where γ_d is the defect reduction tolerance factor, and $\mathbf{tol}_x = \gamma_x \cdot \mathbf{tol}$, where γ_x is the *control variable tolerance factor*. Moreover, let the defect vector be defined by

$$(d_l)_{l=1}^{L} = \sqrt{\omega_l} f_l$$

where ω_l and f_l are defined in Equation 5-1 and Equation 5-2, and L is the total number of the measurement evaluations. Then, when the Levenberg-Marquardt solver is used, the following conditions are used to determine when optimality has been reached:

• Terminate when the defect has been reduced enough; that is,

$$\frac{\|d_j\|_2}{\|d_0\|_2} \le \mathrm{tol}_d$$

where d_0 is the initial defect vector, and d_j is the current defect vector.

- Terminate when the relative increment of the scaled control variable x is below the control variable tolerance; that is,

$$\|x_j - x_{j-1}\|_2 \le \mathrm{tol}_x$$

- Terminate when the cosine between the defect and the Jacobian columns is below the optimality tolerance; that is

$$\max\left(\frac{(J^T \cdot d_j)_i}{\|J(:, i)\|_2 \|d_j\|_2}\right) \le \mathrm{tol}$$

where d_j is the current defect vector and J is the Jacobian.

The default values of the optimality tolerance, defect reduction tolerance factor, and control variable tolerance factor are $1.0 \cdot 10^{-3}$, 1, and 1, respectively. The termination condition defined as the first condition above is not used by default and should be enabled in order to be included.

About Constraint Handling

An important difference between the available optimization solvers is the types of constraints they can handle, and how they do it. Constraints specified in the **Optimization** interface and **Optimization** study step can be divided into three categories:

- *Simple bounds* are upper and lower bounds prescribed directly on the individual control variables, for example in the Optimization study step.
- *Pointwise constraints* specify limits on an expression to be enforced at every node point in some region in space. In order to avoid excessively expensive gradient computations, such constraints are required to only depend on control variables directly and not indirectly via PDE solution variables. The constraint expression can, however, be a nonlinear expression in the control variables.
- *General constraints* specify limits on global scalar expressions, typically evaluated as integrals over some domain. This generates a single constraint, as compared to one for each mesh node for a pointwise constraint. Therefore, the solvers can afford to compute a complete gradient also when the constraint is a, possible nonlinear, function of the PDE solution.

Note that all constraints are treated as inequalities. An equality constraint can be implemented by specifying the same upper an lower bound for an expression. However, not all constraint handling methods are able to deal with the reduction in control variable space dimension which this implies. Therefore, when possible, it is better to perform a manual change of variables, eliminating a control variable dimension.

CONSTRAINT HANDLING FOR DERIVATIVE-FREE METHODS

The derivative-free methods **Coordinate search**, **Monte Carlo**, **Nelder-Mead**, and **COBYLA** can internally handle simple bounds and general constraints on global scalar expressions. In practice this means that in addition to simple bounds and constraints defined in the Optimization study step, **Integral Inequality Constraint** nodes and **Global Inequality Constraint** nodes in Optimization interfaces are accounted for.

The constraint handling algorithm used in **Coordinate search**, **Monte Carlo**, and **Nelder-Mead** is in principle based on filtering out candidate points in the control variable space which fall outside the feasible region, and to some extent adjust search directions accordingly. The procedure is not guaranteed to find a constrained local minimum fulfilling the KKT conditions.

COBYLA, in contrast, approximates objective function and constraints in a uniform way. Therefore, provided all functions are sufficiently smooth, it will in general find an approximate constrained local minimum. The returned solution may, however, lie slightly outside the feasible set. This can happen, in particular, if the constraints are nonlinear.

The Augmented Lagrangian Method

BOBYQA handles simple bounds internally, but general constraints only via an external iterative procedure based on repeated minimization of an augmented Lagrangian. This augmented Lagrangian method can also be used as an alternative to the internal penalty methods in the **Coordinate search** and **Nelder-Mead** solvers, but is not selected by default.

The basic principle behind the augmented Lagrangian method is to include the Lagrange multipliers of general constraints as control variables in an augmented problem. In the first iteration, the Lagrange multipliers are set to zero and a modified objective function including a quadratic penalty for constraint violation is minimized. This gives a solution which is in general outside the feasible set, that is, it violates the constraints. In each subsequent iteration, the Lagrange multipliers — as well as a number of penalty parameters — are updated based on the current constraint

violation, and a new subproblem is solved. The sequence of subproblems, which converges toward the feasible set from the outside, is terminated once a specified tolerance for constrain violation is reached.

The augmented Lagrangian method as such is very general and quite robust, but to be efficient, it requires balancing the effort spent on each subproblem against the improvement in each outer iteration. It also requires selection of an initial penalty factor. A higher penalty factor generally leads to faster convergence of the augmented Lagrangian algorithm, but subproblems become more ill-conditioned and there is a threshold beyond which the method may become unstable; conversely, a lower penalty factor makes the algorithm more robust but required more iterations.

In the **Settings** window for the **Optimization** study step, you can choose to use an **Automatic** or **Manual** definition of the initial **Penalty parameter** ρ. The automatic setting is default and computes an initial value based on the objective and constraint function values at the initial point. You can also limit the **Maximum number of augmented iterations** and select a strategy for updating δ **(tolerance for the subsolver)**. The options **Dynamic I** and **Dynamic II** both tighten the subsolver tolerance from iteration to iteration, the latter providing some additional control. There is also a **Manual** option. Finally, specify the **Constraint tolerance**, that is, the maximum allowable constraint violation in the final solution.

Since the augmented Lagrangian method computes Lagrange multipliers for each constraint explicitly, these are also available for postprocessing. Their values represent the sensitivity (derivative) of the objective function with respect to changes in a constraint bound. In the **Insert Expression** and **Add Expression** menus, available for most postprocessing features, you will find the Lagrange multipliers under **Model>Solver>Lagrange multipliers**.

CONSTRAINT HANDLING FOR GRADIENT-BASED METHODS

The **SNOPT** algorithm handles constraints of all types efficiently. Constraint handling in this SQP method is based on linearizing the constraint in the outer, major, iteration, and using an active-set QP solver in the inner loop to decide which constraints are active and bounding the solution at the current iterate. This process requires accurate evaluation of the gradient of the constraints, also known as the *constraint Jacobian*.

The **MMA** algorithm accepts constraints of the same general type as SNOPT, requiring an accurate constraint Jacobian, but handles them differently. In each outer, major, iteration, linear and linearized constraints are combined with a linearized objective function into a convex smooth approximation whose unique optimum is always feasible unless the feasible set is empty. The globally convergent version of MMA

implemented in the Optimization module is conservative in a way which ensures that each major iterate is feasible not only with respect to the linearized constraints, but with respect to the fully nonlinear constraints.

📄 In the MMA implementation, simple bounds are much less expensive memory-wise than global or pointwise constraints. In particular for control variable fields, simple control variable bounds (added by right-clicking on the **Control Variable Field** feature) are more efficient than enforcing the same bounds using a **Pointwise Inequality Constraint** feature.

The **Levenberg-Marquardt** solver does not support any type of constraints or bounds.

References for the Optimization Solvers

1. A.R. Conn, K. Scheinberg, and L.N. Vicente, *Introduction to Derivative-Free Optimization*, MPS-SIAM Series on Optimization, SIAM, 2009.

2. D. Lee, and M. Wiswall, "A Parallel Implementation of the Simplex Function Minimization Routine", *Computational Economics*, vol. 30, pp. 171–187, 2007.

3. Mike J.D. Powell: *The BOBYQA algorithm for bound constraint optimization without derivatives*, Report DAMTP 2009/NA06, University of Cambridge, UK, 2009.

4. Mike J.D. Powell: "A direct search optimization method that models the objective and constraint functions by linear interpolation", *Proc. Sixth Workshop on Optimization and Numerical Analysis*, vol. 275, pp. 51–67, Kluwer Academic Publishers, Dordrecht, NL, 1994.

5. P.E. Gill, W. Murray, and M.A. Saunders, *User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming*, Systems Optimization Laboratory (SOL), Stanford University, 2006.

6. P.E. Gill, W. Murray, and M.A. Saunders, "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization", *SIAM Review*, vol. 47, no. 1, pp. 99–131, 2005.

7. Krister Svanberg, *MMA and GCMMA − Fortran versions March 2013*, KTH, Royal Institute of Technology, Stockholm, 2013.

8. K. Madsen, H.B. Nielsen, and O. Tingleff, *Methods for Non-Linear Least Squares Problems*, 2nd ed., 2004.

9. R. Andreani, E.G. Birgin, J.M. Martinez, and M.L. Schuverdt, "On Augmented Lagrangian Methods with general lower-level constraints.", *SIAM Journal on Optimization*, 18, pp. 1286–1309, 2007.

> Ref. 7 is available in `<COMSOL_root>`/`doc`/`pdf`/ `Optimization_Module`/`gcmma.pdf`, where `<COMSOL_root>` is the root folder of your COMSOL installation.

The Optimization Solver

The **Optimization Solver** node ( ) contains settings for selecting a gradient-based optimization method and specifying related solver properties.

📄 This section describes **Solver** features available with the Optimization Module. See also Studies and Solvers in the *COMSOL Multiphysics Reference Manual* for more information about solvers in general.

🔍 For a more extensive introduction to the mathematics implemented by this interface, see the Optimization Theory.

For a more extensive treatment of the gradient-based solvers available in this node, see About Gradient-Based Solvers.

GENERAL

The **Optimality tolerance**, **Maximum number of model evaluations** and **Method** settings are fundamental and can be controlled from an **Optimization** study step.

Defined by Study Step

Choose to let an **Optimization** study step control the fundamental optimization method settings (the default). For **User Defined** specify the settings directly in this node.

Optimality Tolerance

Specify the **Optimality tolerance**, which has default value 1e-3. See About Gradient-Based Solvers. Note that this can be too strict, in particular if the forward multiphysics model is not solved accurately enough. See About Optimality Tolerances.

Maximum Number of Model Evaluations

Specify the **Maximum number of model evaluations**, which defaults to 1000. This number limits the number of times the objective function is evaluated, which in most cases is related to the number of times the multiphysics model is simulated for different values of the optimization control variable. Note, however, that it is not equal to the number of iterations taken by the optimizer because each iteration can invoke more than a single objective function evaluation. Furthermore, by setting this parameter to a smaller value and calling the optimization solver repeatedly, you can study the

convergence rate and stop when further iterations with the optimization solver no longer have any significant impact on the value of the objective function.

OPTIMIZATION SOLVER

This section contains settings related to the numerical methods that the solvers use.

Method

The three available choices are **SNOPT** (the default), **MMA** and **Levenberg-Marquardt**. The Levenberg-Marquardt method can only be used for problems of least squares type without constraints or bounds on the control variables, while SNOPT and MMA can solve any type of optimization problem. See About Gradient-Based Solvers.

Solution

This setting controls the behavior when the solver node under the Optimization solver node returns a solution containing more than one solution vector (for example, a frequency response). The SNOPT and Levenberg-Marquardt solvers only support the **Auto** setting, meaning in practice the sum over frequencies and parameters or the last time step. For MMA, the options are as for the derivative-free solvers: **Auto**, **Use first**, **Use last**, **Sum of objectives**, **Minimum of objectives**, and **Maximum of objectives**. The last two settings make the MMA algorithms handle maximin and minimax problems efficiently.

📄 | When optimizing over a **Time Dependent** study step using a gradient-based solver, the objective and its gradient are always evaluated only for the last time step. MMA still presents multiple options, but these are effectively ignored since there is only one objective value that can be used.

Objective Contributions

When SNOPT or MMA is used, the expression used as objective function can be controlled through this setting. The default is **All**, in which case the sum of all objective contributions not deactivated in an **Optimization** study step are used as objective function.

By selecting **Manual**, you can enter an expression that is used as the objective function in the **Objective expression** field. The expression `all_obj_contrib` represents the sum of all objective contributions not deactivated in a controlling **Optimization** study step. Hence, this expressions leads to the same optimization problem as selecting **All**. Note, however, that MMA treats least-squares objective contributions in a more efficient way when **All** is selected.

When you use Levenberg-Marquardt, the objective function is always the sum of all active least-squares objective contributions present in the model.

Gradient Method

SNOPT, MMA and Levenberg-Marquardt are gradient-based methods. The gradient can be computed according to the choices **Automatic, analytic** (default), **Forward**, **Adjoint, Forward Numeric** and **Numeric**. The latter is not supported by MMA. When **Automatic, analytic** is chosen, either the *adjoint method* or the *forward method* is used to compute the gradient analytically. The adjoint method is used when the number of optimization degrees of freedom is larger than the number of objective functions plus the number of global and integral constraints plus two, otherwise the forward method is used.

It is also possible to explicitly choose to use either the adjoint or forward method using the corresponding alternatives from the menu. With the option **Forward Numeric** a semi-analytic approach is available where the gradient of the PDE residual with respect to control variables is computed by numerical perturbation and then substituted into the forward analytic method. When **Numeric** is chosen, finite differences are used to compute the gradient numerically.

> 📝 When the number of control variables is large, calculating the gradient numerically or with forward sensitivity can be time consuming.

Gradient Method Parameters for Time-Dependent Problems

For time-dependent problems, all analytic gradient methods have options to adjust the default integration tolerances for the sensitivity solver.

For the **Forward** and **Forward Numeric** gradient methods a **Forward sensitivity rtol factor** can be specified. This factor multiplied by the forward problem relative tolerance to calculate the relative tolerance for the sensitivity solver. You can also specify a **Forward sensitivity scaled atol**, which is a global absolute tolerance that is scaled with the initial conditions. The absolute tolerance for the sensitivity solution is updated if scaled absolute tolerances are updated for the forward problem.

When using the **Adjoint** gradient method, an **Adjoint rtol factor** and **Adjoint scaled atol** factors can be given, which control the accuracy of the adjoint solution, similarly to the corresponding Forward sensitivity factors. In addition an **Adjoint quadrature rtol factor** and an **Adjoint quadrature atol** can be given. These settings control the relative and absolute accuracy of time integrals (or quadratures) used to calculate objective function gradients. Note that the absolute tolerance is unscaled.

The **Adjoint** gradient method uses checkpointing to reduce the amount of data which needs to be stored from the forward to the backward (adjoint) solution stage. Optionally, set the number of **Adjoint checkpointing steps** to control the number of checkpoints stored.

Numeric Gradient Method Parameters

When the **Numeric** gradient method is selected, you can further specify a **Difference interval** (default $1.5E-8$). This is the relative magnitude of the numerical perturbations to use for first-order gradient approximation in **SNOPT** and for all numeric differentiation in the **Levenberg-Marquardt** solver. The former automatically chooses between first- and second-order gradient approximation, using the specified relative **Central difference interval** (default $6.0E-6$) for central differencing.

For the **Levenberg-Marquardt** method you can choose the **Gradient approximation order** explicitly. Selecting **First** gives a less accurate gradient, while selecting **Second** gives a better approximation of the gradient. However, **Second** requires twice as many evaluations of the objective function for each gradient compared to **First**. In many applications, the increased accuracy obtained by choosing **Second** is not worth this extra cost.

Store Functional Sensitivity

The sensitivity of the objective function is by default stored in the solution object such that it can be postprocessed after the solver has completed. To save memory by discarding this information, change **Store functional sensitivity** to **Off**. Instead choosing **On for results while solving**, sensitivity information is also computed continuously during solution and made available for probing and plotting while solving. This is the most expensive option.

SNOPT-Specific Settings

When using **SNOPT**, you have the possibility to specify which solver to use for solving linear systems containing a reduced Hessian approximation, which is in principle a full matrix. Solving a system involving this matrix is necessary in order to take a single step in the active-set algorithm used for solving the QP subproblems that are formed during each major SQP iteration. Select one of the following strategies from the **QP Solver** list:

• **Cholesky** — This option computes the full Cholesky factor of the reduced Hessian at the start of each major iteration. As the QP iterations (minor iterates) proceed, the dimension of the Cholesky factor changes with the number of superbasic variables and the factor is updated accordingly. If the number of superbasic variables increases beyond a preset limit (1000), the reduced Hessian cannot be stored and the solver switches to conjugate gradient.

- **Conjugate gradient** — This method uses the conjugate-gradient method to solve all systems involving the reduced Hessian, which is only accessed implicitly in the form of a black-box linear operator on the superbasic variables. Since no data is stored between inner iterations, the method is most appropriate when the number of superbasics is large but each QP subproblem requires relatively few minor iterations. Selecting **Conjugate gradient** also triggers a limited-memory procedure which stores only a fixed number of BFGS update vectors together with a diagonal Hessian approximation between major iterations.

- **Quasi-Newton** — This method uses a quasi-Newton strategy to update an approximation of the Cholesky factor of the reduced Hessian as the iterations proceed. It has the same memory requirement as the Cholesky option, but does not recompute the complete Cholesky factor at the beginning of each major iteration. It can be an appropriate choice when the number of superbasics is large but the nonlinear problem is well-scaled and well-behaved such that relatively few major iterations are needed for the approximate Hessian to stabilize.

	The Quasi-Newton option for solving reduced Hessian systems must not be confused with the fact that the major SNOPT iterations always use a quasi-Newton BFGS strategy to approximate the full Hessian — also when using Cholesky factorization or conjugate gradients to solve the reduced systems.

	The constraint Jacobian matrix is always assumed to be sparse such that sparse LU factors of the basic part of this matrix can be stored and updated when the set of superbasic variables changes. These factors are used implicitly to define the null space of the active constraints and appear in the implicit representation of the reduced gradient and Hessian.

	• See The SNOPT Solver. • See page 80 of the *SNOPT User's Guide* (Ref. 5 under About the Optimization Solvers).

In the **Use step condition** field you can enter an expression that tells the optimization solver to reduce the step length in the current line search used by SNOPT to generate the next iterate.

The solver uses the condition to restrain the iterates from entering into areas in the control-variable space where the PDE problem is not well defined. A typical example is when a mesh element becomes inverted during geometry optimization using a Moving Mesh interface. A step limit condition that identifies this situation might be of the form `minqual1_ale-0.05`, where 0.05 is a threshold value for the mesh quality. This step limit condition has a direct analog in the stop condition for the time-dependent and parametric solvers.

> Only use the step limit condition as a last resort to keep the optimization solver in a feasible region. Instead, if possible, use pointwise constraints on the optimization variables to enforce the condition.

When the step limit condition is violated, the solver reduces the line-search step until an acceptable point is found. However, because no Jacobian is computed for the step limit condition, there is no mechanism to prevent the solver from immediately attempting another step in the same infeasible direction. As a result, the solver might get stuck at the same point without converging until it reaches the maximum number of model evaluations or you stop the iteration manually.

You can specify a linesearch tolerance as a value between 0 and 1 in the **Linesearch tolerance** field (default value: 0.9). This controls the accuracy with which a step length will be located along the direction of search in each iteration. At the start of each linesearch, a target directional derivative for the merit function is identified. This parameter determines the accuracy to which this target value is approximated:

- The default value of 0.9 requests just moderate accuracy in the linesearch.
- If the nonlinear functions are cheap to evaluate, a more accurate search may be appropriate; try 0.1, 0.01, or 0.001. The number of major iterations might decrease.
- If the nonlinear functions are expensive to evaluate, a less accurate search may be appropriate. If all gradients are known, try a tolerance of 0.99. (The number of major iterations might increase, but the total number of function evaluations may decrease enough to compensate.)
- If not all gradients are known, a moderately accurate search remains appropriate.

Each search will require only 1–5 function values (typically), but many function calls are then needed to estimate missing gradients for the next iteration.

From the **Linesearch strategy** list, choose **Derivative** (the default) or **Nonderivative**. At each major iteration a linesearch is used to improve the merit function. A derivative linesearch uses safeguarded cubic interpolation and requires both function and gradient values to compute estimates of the step. If some analytic derivatives are not provided, or a nonderivative linesearch is specified, SNOPT uses a linesearch based on safeguarded quadratic interpolation, which does not require gradient evaluations.

A nonderivative linesearch can be slightly less robust on difficult problems, and it is recommended that you use the default derivative linesearch if the functions and derivatives can be computed at approximately the same cost. If the gradients are very expensive relative to the functions, a nonderivative linesearch may give a significant decrease in computation time.

MMA-Specific Settings

By default, the MMA solver continues to iterate until the relative change in any control variable is less than the optimality tolerance. If the **Maximum outer iterations** option is enabled, the solver stops either on the tolerance criterion or when the number of iterations is more than the maximum specified.

The Optimization Module's globally convergent version of the MMA solver has an inner loop which ensures that each new outer iteration point is feasible and improves on the objective function value. By default, the **Maximum number of inner iterations per outer** iteration is 10. When the maximum number of inner iterations is reached, the solver continues with the next outer iteration.

The **Internal tolerance factor** is multiplied by the optimality tolerance to provide an internal tolerance number that is used in the MMA algorithm to determine if the approximations done in the inner loop are feasible and improve on the objective function value. The default is 0.1. Decrease the factor to get stricter tolerances and a more conservative solver behavior.

The MMA algorithm penalizes violations of the constraints by a number that is calculated as the specified **Constraint penalty factor** times 1e-4 divided by the optimality tolerance. Increasing this factor for a given optimality tolerance forces the solver to better respect constraints, while relatively decreasing the influence of the objective function.

Levenberg-Marquardt-Specific Settings

The Levenberg-Marquardt method controls the step length and direction through a positive scalar regularization parameter. A value close to zero means that the optimization solver takes a step close to a full Gauss-Newton step. A large value means

that it takes a small step close to the steepest-descent direction. See The Levenberg-Marquardt Solver.

The Levenberg-Marquardt method controls this penalty factor internally and tries to have as small penalty as possible in order to approach second-order Newton convergence. Therefore, a small value of the **Initial damping factor** means that the solver tries to be aggressive initially, while a large value means that the solver is more cautious.

RESULTS WHILE SOLVING

Select the **Plot** check box to plot the results while solving the model. Select a **Plot group** from the list and any applicable **Probes**.

ADVANCED

Use the **Compensate for nojac terms** list to specify whether to try to assemble the complete Jacobian if an incomplete Jacobian has been detected. Select:

- **Automatic** (the default) to try to assemble the complete Jacobian if an incomplete Jacobian has been detected. If the assembly of the complete Jacobian fails or in the case of nonconvergence, a warning is written and the incomplete Jacobian is used in the sensitivity analysis for stationary problems. For time-dependent problems, an error is returned.

- **On** to try to assemble the complete Jacobian if an incomplete Jacobian has been detected. If the assembly of the complete Jacobian fails or in the case of nonconvergence, an error is returned.

- **Off** to use the incomplete Jacobian in the sensitivity analysis.

See Theory for Stationary Sensitivity Analysis for details about the algorithm.

CONSTANTS

In this section you can define constants that can be used as temporary constants in the solver. You can use the constants in the model or to define values for internal solver parameters. Click the **Add** (➕) button to add a constant and then define its name in the **Constant name** column and its value (a numerical value or parameter expression) in the **Constant value** column. By default, any defined parameters are first added as the constant names, but you can change the names to define other constants. Click **Delete** (🗙) to remove the selected constant from the list.

LOG

The **Log** displays the information about the progress of the solver.

Advanced Solver Properties

This section provides detailed explanations of some of the properties that control the behavior of the SNOPT optimization solver in the Optimization Module.

When solving multiphysics optimization problems in the COMSOL Desktop using the Optimization interface, some of the properties listed in this section can be controlled while others always take their default values. Modifying the value of those properties requires that the value is changed using LiveLink™ *for* MATLAB® or by running a compiled COMSOL API history file.

	In the following sections, ε represents the machine precision (available as `eps` in MATLAB) and is approximately equal to $2.2 \cdot 10^{-16}$.

In this section:

- SNOPT Solver Properties
- MMA Solver Properties

	For a list of all available optimization solver properties see `Optimization` in the *COMSOL Multiphysics Programming Reference Manual*.

SNOPT Solver Properties

FEASTOL

Feasibility tolerance
Type: numeric
Default: $1.0 \cdot 10^{-6}$

The solver tries to ensure that all bound and linear constraints are eventually satisfied to within the feasibility tolerance t. (Feasibility with respect to nonlinear constraints is instead judged by the major feasibility tolerance, `majfeastol`.)

If the bounds and linear constraints cannot be satisfied to within t, the problem is declared infeasible. Let **sInf** be the corresponding sum of infeasibilities. If **sInf** is quite small, it might be appropriate to raise t by a factor of 10 or 100. Otherwise you should suspect some error in the data.

Nonlinear functions are evaluated only at points that satisfy the bound and linear constraints. If there are regions where a function is undefined, every attempt should be made to eliminate these regions from the problem. For example, if

$$f(x) = \sqrt{x_1} + \log x_2$$

it is essential to place lower bounds on both variables. If $t = 10^{-6}$, the bounds

$$x_1 \geq 10^{-5} \text{ and } x_2 \geq 10^{-4}$$

might be appropriate. (The log singularity is more serious. In general, keep x as far away from singularities as possible.)

In practice, the solver uses t as a feasibility tolerance for satisfying the bound and linear constraints in each QP subproblem. If the sum of indefeasibility cannot be reduced to zero, the QP subproblem is declared infeasible. The solver is then in the Elastic mode thereafter (with only the linearized nonlinear constraints defined to be elastic).

FUNCPREC

Function precision
Type: numeric
Default: $\varepsilon^{0.8} \approx 3.8{\cdot}10^{-11}$

The relative function precision is intended to be a measure of the relative accuracy with which the nonlinear functions can be computed. For example, if $f(x)$ is computed as 1000.56789 for some relevant x and if the first 6 significant digits are known to be correct, the appropriate value for the function precision would be 10^{-6}. (Ideally the functions should have a magnitude of order 1. If all functions are substantially less than 1 in magnitude, the function precision should be the absolute precision. For example, if $f(x) = 1.23456789{\cdot}10^{-4}$ at some point and if the first 6 significant digits are known to be correct, the appropriate precision would be 10^{-10}.)

The default value is appropriate for simple analytic functions.

In some cases the function values are the result of extensive computations, possibly involving an iterative procedure that can provide rather few digits of precision at reasonable cost. Specifying an appropriate function precision might lead to savings by allowing the line search procedure to terminate when the difference between function values along the search direction becomes as small as the absolute error in the values.

HESSUPD

Hessian updates
Type: integer
Default: 10

When the number of nonlinear variables is large (more than 75) or when the QP problem solver is set to conjugate-gradient, a limited-memory procedure stores a fixed number of BFGS update vectors and a diagonal Hessian approximation. In this case, if `hessupd` BFGS updates have already been carried out, all but the diagonal elements of the accumulated updates are discarded and the updating process starts again. Broadly speaking, the more updates stored, the better the quality of the approximate Hessian. However, the more vectors stored, the greater the cost of each QP iteration. The default value is likely to give a robust algorithm without significant expense, but faster convergence can sometimes be obtained with significantly fewer updates (for example, `hessupd = 5`).

MAJFEASTOL

Major feasibility tolerance
Type: numeric
Default: $1.0 \cdot 10^{-6}$

This parameter specifies how accurately the nonlinear constraints should be satisfied. The default value of $1.0 \cdot 10^{-6}$ is appropriate when the linear and nonlinear constraints contain data to roughly that accuracy.

Let **rowerr** be the maximum nonlinear constraint violation, normalized by the size of the solution. It is required to satisfy

$$\text{rowerr} \;=\; \max_i \; \text{viol}_i / (\|x\| + 1) \leq \text{majfeastol}$$

where viol_i is the violation of the ith nonlinear constraint. If some of the problem functions are known to be of low accuracy, a larger major feasibility tolerance might be appropriate.

OPTTOL

Optimality tolerance
Type: numeric
Default: $1.0 \cdot 10^{-3}$

This is the major optimality tolerance and specifies the final accuracy of the dual variables. On successful termination, the solver computes a solution (x, s, π) such that

$$\text{maxComp} = \max_{j} \ \text{Comp}_j / \|\pi\| \le \text{opttol}$$

where Comp_j is an estimate of the complementarity slackness for variable j. The values Comp_j are computed from the final QP solution using the reduced gradients $d_j = g_j - \pi^T a_j$, as above. Hence you have

$$\text{Comp}_j = \begin{cases} d_j \ \min \ \{x_j - l_j \,, 1\} & \text{if } d_j \ge 0 \\ -d_j \ \min \ \{u_j - x_j \,, 1\} & \text{if } d_j < 0 \end{cases}$$

🗒️ | See the SNOPT User's Guide for further details.

QPSOLVER

QP problem solver
Type: string 'cholesky', 'cg', or 'qn'
Default: 'cholesky'

Specifies the active-set algorithm used to solve the QP problem, or in the nonlinear case, the QP subproblem.

`'cholesky'` holds the full Cholesky factor R of the reduced Hessian $Z^T H Z$. As the QP iterations proceed, the dimension of R changes with the number of superbasic variables.

`'qn'` solves the QP subproblem using a quasi-Newton method. In this case, R is the factor of a quasi-Newton approximate reduced Hessian.

`'cg'` uses the conjugate-gradient method to solve all systems involving the reduced Hessian. No storage needs to be allocated for a Cholesky factor.

The Cholesky QP solver is the most robust but might require a significant amount of computation and memory if the number of superbasics is large.

The quasi-Newton QP solver does not require the computation of the exact R at the start of each QP and might be appropriate when the number of superbasics is large but each QP subproblem requires relatively few minor iterations.

The conjugate-gradient QP solver is appropriate for problems with large numbers of degrees of freedom (many superbasic variables). The Hessian memory option `'hessmem'` is defaulted to `'limited'` when this solver is used.

MMA Solver Properties

In addition to the settings available in the COMSOL Desktop, MMA when called through the API allows explicit tuning of the approximating subproblems solved in each inner iteration. In particular, update rules for the moving asymptotes can be modified. It is also possible to switch off the automatic transformation performed on least-squares, minimax and maximin problems, as well as disable the globally convergent extension of the MMA method.

For a list of all available options see `Optimization` in the *COMSOL Multiphysics Programming Reference Manual*.

6

Glossary

This Glossary of Terms contains modeling terms in an optimization and sensitivity context. For general mathematical and finite element terms, and geometry and CAD terms specific to the COMSOL Multiphysics software and documentation see the glossary in the *COMSOL Multiphysics Reference Manual*. For references to more information about a term, see the index.

Glossary of Terms

adjoint method The *adjoint method* for sensitivity analysis is based on exploiting the adjoint identity given an objective functional to derive and solve a set of adjoint equations. The adjoint solution is then used to compute the functional sensitivity with respect to sensitivity parameters.

BFGS (Broyden–Fletcher–Goldfarb–Shanno) A specific family of optimization algorithms where updates are made to approximate the inverse of the Hessian matrix.

bounds An inequality constraint setting lower and upper bounds directly on each control variable degree of freedom.

contributions to objective function The objective function is a scalar function of the control variables. In the optimization interface, the objective is formed by the summation of contributions from *global contributions*, *probe contributions*, and *integral contributions* to the objective functions.

control variable The *control variables* parameterize the optimization or sensitivity problem. The objective function and constraint are expressed in the terms of the control variables. In the mathematical and engineering literature, the control variables are sometimes also referred to as *optimization variables, design variables,* or *decision variable*.

design constraint A constraint which can be evaluated before any multiphysics simulation has been performed. A design constrain can be expressed explicitly in the control variables, without involving the multiphysics problem solution.

design problem An optimization problem where the objective function quantifies the performance in a multiphysics model. For such problems, the control variable is sometimes referred to as the *design variables*. Problems of this kind arise in, for example, structural optimization, antenna design, and process optimization.

feasible set The control variables can be constrained to a feasible set. The feasible set is typically expressed by a set of constraints acting on the control variables. The feasible set can also be implicitly limited by the existence of a solution to a multiphysics problem.

forward method The *forward method* for sensitivity analysis is based on solving the equations obtained by applying the chain rule of differentiation, with respect to sensitivity parameters, to the original DAEs.

global inequality constraint A constraint that sets upper and lower bounds on a general global expression, possibly involving both the control variables and the PDE solution.

integral inequality constraint A constraint that sets upper and lower bounds on an integral of an expression, possibly involving the PDE solution and control variables, over a set of geometric entities of the same dimension

objective function A single-valued function of the PDE solution and control variables representing the performance of a multiphysics model or how well a parameterized model matches measured data. Alternative terminology used for the objective function is *cost function*, *goal function*, or *quantity of interest*.

optimization problem The *optimization problem* is to find values of the control variables, belonging to a given feasible set, such that the objective function attains its minimum (or maximum) value.

parameter estimation problem An inverse problem where the objective function defines how well a parameterized model matches measured data. Replacing the parameters with control variables leads to an optimization problem, which can arise in, for example, geophysical imaging, nondestructive testing, and biomedical imaging.

PDE-constrained optimization problem An optimization problem where the feasible set is limited by the condition that a given multiphysics model, represented as a PDE, has a unique solution.

PDE solution The solution to a multiphysics problem in response to specific values of the control variables.

performance constraint A constraint involving the multiphysics simulation result. Performance constraints in general have the same structure as the objective function, and are as expensive to evaluate.

pointwise inequality constraint An inequality constraint in a PDE-constrained optimization problem involving an explicit expression in terms of the control variables. The constraint sets lower and upper bounds on the expression for node points in a set of geometric entities of the same dimension.

sensitivity problem The sensitivity problem determines the gradient of an objective function with respect to the control variables.

solution variables Designates variables that are not control variables, for example, field variables and global variables.

superbasic variable A variable is *superbasic* if it is not currently at one of its bounds.

Index