Optimizer User Guide

Version N-2017.09, September 2017

SYNOPSYS®

Copyright and Proprietary Information Notice

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Contents

About This Guide

The Optimizer tool is part of Synopsys Sentaurus™ Workbench Advanced. It is an analysis tool designed for parametric studies in large-scale projects with hundreds of individual simulations, such as automatic iterative optimization, sensitivity analysis, and uncertainty analysis. This user guide describes the model tasks and experiments that can be performed using Optimizer. The scripting language, equations, optimization methods, and mathematical expressions used with Optimizer are described in detail.

Related Publications

For additional information, see:

- The TCAD Sentaurus release notes, available on the Synopsys SolvNet® support site (see Accessing SolvNet on page x).
- Documentation available on SolvNet at https://solvnet.synopsys.com/DocsOnWeb.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Blue text	Identifies a cross-reference (only on the screen).
Bold text	Identifies a selectable icon, button, menu, or tab. It also indicates the name of a field or an option.
`Courier font`	Identifies text that is displayed on the screen or that the user must type. It identifies the names of files, directories, paths, parameters, keywords, and variables.
Italicized text	Used for emphasis, the titles of books and journals, and non-English words. It also identifies components of an equation or a formula, a placeholder, or an identifier.
Menu > Command	Indicates a menu command, for example, **File > New** (from the **File** menu, select **New**).

Customer Support

Customer support is available through the Synopsys SolvNet customer support website and by contacting the Synopsys support center.

Accessing SolvNet

The SolvNet support site includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The site also gives you access to a wide range of Synopsys online services, which include downloading software, viewing documentation, and entering a call to the Support Center.

To access the SolvNet site:

1. Go to the web page at https://solvnet.synopsys.com.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register.)

If you need help using the site, click **Help** on the menu bar.

Contacting Synopsys Support

If you have problems, questions, or suggestions, you can contact Synopsys support in the following ways:

- Go to the Synopsys Global Support Centers site on synopsys.com. There you can find e-mail addresses and telephone numbers for Synopsys support centers throughout the world.
- Go to either the Synopsys SolvNet site or the Synopsys Global Support Centers site and open a case online (Synopsys user name and password required).

Contacting Your Local TCAD Support Team Directly

Send an e-mail message to:

- support-tcad-us@synopsys.com from within North America and South America.
- support-tcad-eu@synopsys.com from within Europe.
- support-tcad-ap@synopsys.com from within Asia Pacific (China, Taiwan, Singapore, Malaysia, India, Australia).
- support-tcad-kr@synopsys.com from Korea.
- support-tcad-jp@synopsys.com from Japan.

CHAPTER 1 Using Optimizer

This chapter provides an overview of the Optimizer tool.

Functionality of Optimizer

Optimizer is a batch mode tool that is used to perform the efficient extraction of general information about TCAD simulations. For example, it is used to determine parameter settings that satisfy design specifications and to analyze how parameter variations affect the device behavior. It provides the following set of tools:

- Design-of-experiments (DoE)

  Comprehensive selection of experimental designs, including full and fractional factorial, Box–Behnken, Plackett–Burmann, and Taguchi, as well as random and stochastic designs. These techniques allow models to be built with a minimal number of simulations.

- Response surface models (RSMs)

  Interactive building and evaluation of empirical models. These models are polynomial approximations or interpolations of multidimensional responses to different parameter values. The accuracy is determined automatically. Parameter and response domains of different orders of magnitude are handled transparently.

- Screening

  Model coefficients are displayed to illustrate the impact of different parameters on the simulation responses.

- Optimization

  Maximize or minimize a certain response or find a parameter setting such that a response is close to a given value. Multiple optimization goals for different responses can be combined; weights specify their relative importance. A nonlinear optimizer based on sequential quadratic programming (SQP) is used to solve optimization problems.

- Iterative optimization

  Iterative heuristic search process of a parameter setting that optimizes the device behavior. At each iteration, a new region of the parameter domain is explored with the aim of finding a globally satisfactory set of parameter values.

- Generic optimization

  Search for a parameter setting that optimizes the device behavior using generic optimization strategies. Both quasi-Newton and nonlinear simplex methods have been adapted for this purpose.

- Genetic algorithm

  The genetic algorithm optimization strategy is classified among the *evolutionary* approaches. The optimum is achieved after several iterations (called generations) in which a set of possible solutions is evaluated and, for each generation, a better set of experiments is examined.

- Sensitivity analysis

  Determine how small changes to a given parameter affect simulation responses.

- Uncertainty analysis

  Comprehensive analysis of how the variability of parameters affects the device behavior. Multidimensional stochastic RSMs are used to determine correlations between parameters and simulation responses and, ultimately, to extract an approximation of the probability density function of the simulation responses using the Monte Carlo method.

- Custom algorithm

  Optimizer allows the definition and use of custom algorithms that can address specific tasks that may not be available through the standard task types. The definition of the algorithm can use all of the structures and procedures of Optimizer, for example, building a DoE or an RSM.

These tools can be combined sequentially in a single Optimizer run. For example, a screening task can be combined with an iterative optimization task. The screening task would first identify which parameters have the strongest impact on the simulation responses and, then, the iterative optimization task would find values for the selected parameters that optimize the device behavior.

Optimizer interacts with batch tools of Sentaurus Workbench for setting up and running simulations in the context of TCAD simulation projects. It is possible to take advantage of the job scheduler of Sentaurus Workbench to speed up simulations using distributed, heterogeneous, corporate computing resources. The open architecture and tool interface of Sentaurus Workbench allow Optimizer to be used for a wide range of purposes.

Starting Optimizer

Optimizer can be started from the graphical user interface of Sentaurus Workbench by choosing **Optimization > Run** or from the command line using:

```
swbopt <options> <project_dir>
```

where `<project_dir>` is the name of the project to be executed. Usually, it is a project directory of Sentaurus Workbench. By default, Optimizer reads a command file called `gopt.cmd` from the directory `<project_dir>`. This file contains the tasks that Optimizer must perform; its syntax and definitions are discussed in Input Command File on page 8.

The following command-line options are available:

```
-h[elp]     : Displays this help message
-v[ersion]  : Displays the version number
-verbose    : Displays all messages
-silent     : Deactivates standard output display
-q[ueue]    : Allows to select a queue for running simulations
-expr       : Shows the polynomial representation of all models
                 created
-export     : Exports created model to a PCM XML file
-patch FILE: Allows an external Tcl patch file to be loaded
```

CHAPTER 2 Operations Guide

Optimizer is a batch tool designed to facilitate the analysis of simulations. This tool uses batch tools of Sentaurus Workbench for automatically running simulations in the context of TCAD simulation projects.

Basic Concepts

This section lists some common terms relevant to understanding Optimizer:

- Parameter

  A *parameter* is a scalar variable that modifies the simulation flow, which allows users to define families of similar simulations. It is a finite value that is defined inside a certain domain. Parameter definition is taken automatically from the project of Sentaurus Workbench. Therefore, only parameters of Sentaurus Workbench can be used by Optimizer.

  The three parameter categories are:

user-defined	Parameters with values that are specified by the user.
doe	Parameters with values that are set using a deterministic design-of-experiments.
sdoe	Parameters with values that are set using a stochastic design-of-experiments.

- Response

  A *response* is a scalar variable or simulation output that describes, for example, the device behavior.

- Experiment

  An *experiment* or *parameter setting* is a tuple that contains one value for each parameter of the project of Sentaurus Workbench. A *family of simulations* is defined as a set of simulations that have the same parameter values for all ‘doe’ and ‘sdoe’ parameters, but can have different values for ‘user-defined’ parameters. An *evaluation* is defined as all the simulations required to compute a given family of simulations.

- Scenario

  A *scenario* is a subtree of a simulation tree of Sentaurus Workbench that defines a particular subset of experiments. Scenarios can overlap, that is, a particular node or path can be part of more than one scenario. When Optimizer submits a new set of experiments to batch tools of Sentaurus Workbench, a new scenario is added to the simulation tree so as to add all new experiments. In Sentaurus Workbench, scenarios can be run and edited independently.

- Tasks

  A *task* is a sequence of actions used to obtain information about the relationship between the parameters and responses under consideration.

Optimizer Structures

The main object that links all relevant elements in Optimizer is the task. A task usually requires the evaluation of how several combinations of different parameter settings produce different response values. These evaluations can be performed through an external simulation process, mainly invoked and coordinated by the batch tools of Sentaurus Workbench, but they can also be performed by evaluating a tool command language (Tcl) function or a formula.

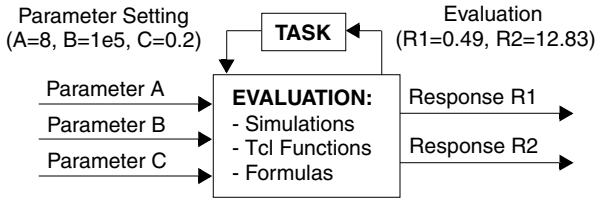

Figure 1 Tasks require the evaluation of multiple parameter settings

Sequencing Tasks

More than one task can be performed in a single execution. Tasks can use the evaluations and information obtained in previous tasks. For example, if a task requires the evaluation of a family of simulations that has already been evaluated by a previous task, that family is not re-evaluated and the values are recovered by the next task. That is, if a screening task selects only some parameters, only those are considered in subsequent tasks.

Task Interdependency

A task can use values obtained as results of other executed tasks. This scheme identifies ‘parents’ and ‘children’ tasks. Data is shared from parents to children tasks using an export-and-import mechanism, which allows for the declaration of which values of one task can be exported and which name is used to import them from another task.

For example, it is possible to combine an iterative optimization task with an uncertainty analysis task, allowing the goal of the optimization to be the minimization of the variance of the responses. This concept is also known as *robust design*.

A single run of Optimizer can be related to multiple projects because a different response can be obtained from different tasks that are related to different projects.

Several values can be imported and exported from one task to another (see Task Interdependency on page 76).

Reference Example

The examples in this user guide correspond to an NMOSFET project where device behavior is studied by analyzing the following simulation responses: threshold voltage (`VT1_WL`) and breakdown voltage (`VBR`). The analysis considers different values of the following parameters:

- p-well implantation dose (`PW_DOSE`)
- Gate oxide thickness (`TH_OX`)
- Channel dose (`CH_DOSE`)
- LDD implantation dose (`LDD_DOSE`)
- Spacer length (`SP_LENGTH`)

An example of the definition of the input command file is:

```
Start {
   nextTask = SCR1
}
Task {
   name = SCR1
   type = SCREENING
   Parameter {
      { name = PW_DOSE
      type = doe min = 1e12 max = 1e13 scale = logarithmic }
      { name = TH_OX
      type = doe min = 15 max = 20 }
      { name = CH_DOSE
```

```
      type = doe min = 4e11 max = 4e12 scale = logarithmic }
      { name = LDD_DOSE
      type = doe min = 5e12 max 5e13 scale = logarithmic }
      { name = SP_LENGTH
      type = doe min = 0 max = 300 }
   }
   Response {
      { name = VT1_WL
      model = standard degree = 1 }
      { name = Vbr
      model = standard degree = 1 }
   }
   doe = plackettBurmann
   screenRange = 2.0
   screenCriteria = average
   nextTask = end
}
```

The specified code in this example input file shows the following main aspects:

- Only one task is defined. The task is called `SCR1` and corresponds to a screening task. It is the first one to be executed according to the definition in the `Start` block.
- Five parameters are analyzed inside a given range (linearly or logarithmically scaled). Their values are generated using a Plackett–Burmann design-of-experiments.
- Two responses are considered.
- The main tool to be used is `SCREENING` that will analyze which parameters are relevant for the responses, considering how changes to the parameter values affect the response values. Those parameters that have an average effect (on both responses) greater than 2% are selected.

Input Command File

The command file `gopt.cmd` defines a set of Optimizer tasks. This file consists of a sequence of blocks that can be in any order and contain inner blocks. A block has always the following structure: `block_name { body }`. The body of a block is a sequence of either other blocks or data values.

A data value is an attribute and its corresponding value or an array of data values. An array of data values is a set of related data values.

Main blocks are separated into different lines. Inside the body of a block, data is separated by line feeds or multiple spaces. Keywords of Optimizer are case sensitive and syntax sensitive, for example, parentheses must be consistent. All lines starting with the symbol '#' are ignored.

This section describes and provides an example of each block. Optimizer Commands Reference on page 81 provides users with reference tables specifying the elements that can be used in each block.

Main Blocks

Start

The `Start` block allows users to define initial characteristics of parameters and responses, and to set global options. The `Start` block can be specified anywhere in the input command file, at the beginning or end, or in between task specifications.

```
# Define initial conditions and start first task
Start {
   Parameter {
      { name = VARIABLE  type = ud  values = {0 1} }
   }
   nextTask = 1
}
```

In this example, the `Start` block specifies a user-defined parameter `VARIABLE`, which is not used by any task and takes the values 0 and 1. The initial task is set to 1.

Global Options

Optimizer stops if one of the following global stopping criteria is reached:

`maxGlbNumEvaluations`	Global maximum number of evaluations.
`maxGlbTime`	Maximum wallclock time that Optimizer is allowed to run.
`maxGlbTimeUnit`	(seconds, minutes, hours, or days) Global time unit. The default unit is second [s].

Other stopping criteria are task dependent and must be specified in the corresponding task definition. The following global options are also available:

`nextTask`	Alphanumeric identifier of the first task that Optimizer runs.
`exportTable`	This option is used to export a table containing the different parameter settings and all response values to a file (in tab-delimited format). The argument of this option is the file name.

Global options can be modified in any task. `nextTask` must be set in all tasks to indicate which task is executed next. `exportTable` can be set differently to export the results of each task to different files.

Task

The `Task` block allows users to define Optimizer tasks. These tasks are executed according to a user-specified order. Each task definition contains a block that states the selected parameters and responses. Additionally, some task-specific options can be set. The following sections describe the different tasks and their specific options that are available in Optimizer.

The main attributes of a task are:

`name`	Alphanumeric identifier of the task.
`type`	Type of task to be performed, including `DOE`, `SDOE`, `SCREENING`, `OPTIMIZATION`, `GEN_OPTIMIZATION`, `ITER_OPTIMIZATION`, `SEN_ANALYSIS`, `UNC_ANALYSIS`, `GENETIC_OPTIMIZATION`.
`project`	Name of Sentaurus Workbench project. Each task can be linked to a different Sentaurus Workbench project. Parameter names must match the parameters of the corresponding project. The task attribute 'project' is optional and its default value is " ", which references the directory where the input command file is located.
`Parameter`	See Parameter on page 11 for their corresponding declarations.
`Response`	See Response on page 13 for their corresponding declarations.

A simple example of a task declaration is:

```
Task {
   name = SCR1
   type = SCREENING
   Parameter {
      { name = PW_DOSE
      type = doe min = 1e12 max = 1e13 scale = logarithmic }
      { name = TH_OX
      type = doe min = 15 max = 20 }
      { name = CH_DOSE
      type = doe min = 4e11 max = 4e12 scale = logarithmic }
      { name = LDD_DOSE
      type = doe min = 5e12 max 5e13 scale = logarithmic }
      { name = SP_LENGTH
      type = doe min = 0 max = 300 }
   }
   Response {
```

```
        { name = VT1_WL
        model = standard degree = 1 }
        { name = Vbr
        model = standard degree = 1 }
    }
    doe = plackettBurmann
    screenRange = 2.0
    screenCriteria = average
    nextTask = end
}
```

Inner Blocks

This section describes and provides examples of some inner blocks that are common to all tasks.

Parameter

A parameter is characterized by the attributes:

`name` Unique identifier.

`type` (`ud`, `doe`, `sdoe`, or `scen`) Parameter type. The default is user-defined (`ud`).

The remaining parameter attributes are type dependent.

User-Defined Parameters

`values` If the parameter is user defined, these values are used to create the different family of simulations.

Design-of-Experiments (DoE) Parameters

`max` Highest possible value for DoE generation.

`min` Lowest possible value for DoE generation.

`scale` (`linear` or `logarithmic`) Interpolation method for creating intermediate values between the lower and upper bounds. A logarithmic scale can be used only when all possible parameter values are positive or negative. The default is `linear`.

`doeArgs` Some DoEs require certain parameter-dependent attributes to be specified (see the DoE used in the task example in Design-of-Experiments Task on page 68).

`selValue` Selected value. It can be used as the nominal value of a sensitivity analysis task or the starting point of an optimization task. If its value is set to `autoValue`, it takes the value from the previous executed task. In addition, the import capacity between tasks can be used for parameters to obtain a value from another parameter. When `selValue` is set to `import.X1`, this means that the value of `X1`, defined in another task, will be inherited by this task.

Stochastic Design-of-Experiments (SDoE) Parameters

`SDoEModel` Probability distribution. This is a block compound of two attributes:
`type`: (`normal`, `uniform`, `expon`, `beta`, or `gamma`) Probability distribution type. Default distribution is `normal`.
`args`: Arguments of the probability distribution.

For example, if a given parameter has a normal distribution, two arguments must be provided – mean and variance. The default is `{0 1}`, which corresponds to the mean and variance of the standard normal distribution.

`sdoeArgs` Closely analogous to the deterministic case. Some stochastic DoEs require certain parameter-dependent attributes to be specified (see the DoE used in the uncertainty analysis task example in Example on page 61).

Scenario-Originated Parameters

It is also possible to use a previously defined scenario to set the values of some parameters. In that case, the required parameter is defined as a *scenario* type, which means that this parameter inherits all values defined in the specified scenario.

`values` This attribute specifies the scenario to be used to set these parameter values. The declaration has two parts: the path to the referenced project and the scenario name in that project:
`{ [path-to-example] [scenario-name] }`

Response

A response is characterized by the following attributes:

`name`	Unique identifier.
`model`	(`standard`, `kriging`, or `stochastic`) Type of RSM.
`transformation`	(`exp`, `log`, `sqrt`, or `sqr`) Transformation of the RSM.
`degree`	(`1`, `2`, or `3`) If a polynomial model is used, this attribute represents the degree of the RSM.
`crit`	(`minimal`, `maximal`, or `closeto`) Optimization goal.
`target`	If the optimization criterion `closeto` is used, this argument represents the target value.
`lowerBound`	Minimum limit related to the algorithm stopping range (used for optimizations).
`upperBound`	Maximum limit related to the algorithm stopping range (used for optimizations).
`perc_range`	Percentage related to the target value for which the stopping range is defined.
`weight`	Relative importance assigned to the simulation response.
`modelArgs`	Stochastic RSM specification. Although, there is always the possibility of declaring each term of the stochastic specification, some keywords help their automatic internal generation. These keywords are `linear`, `interaction`, `quadratic`, and `cubic`. By specifying one or more of these keywords, they are replaced by the requested parameter combinations.
`source`	Task responses can be computed by calling other tasks (particularly simulation tasks that perform actual simulations), evaluating a formula, or calling a function. This attribute allows for specifying how to compute the response values. This attribute is optional, but when declared, it must be the name of an existing task, a previously defined formula, or a function.

Sequencing Tasks

For a specific run of Optimizer, the order in which tasks are executed can be determined and controlled by the user using a specific command file definition. The sequence in which tasks are executed can be useful, for example, to perform a screening task before an optimization task, which will use only those parameters selected by the screening process to be optimized.

This concept involves two considerations: declaring the order in which tasks are executed and declaring how the results of the first task affect the start and evaluation of the second task.

Determining Order of Tasks

The starting task is defined by using its identifier on the `nextTask` attribute of the `Start` block:

```
# Define initial conditions and start first task
Start {
   ...
   nextTask = SCR1
}
```

In this example, the task identified by `SCR1` is executed first. Then, each following task to be executed is defined in the `nextTask` attribute of the previous task:

```
Task {
   ...
   nextTask = OPT
}
```

The final task sets the `nextTask` attribute to `end`.

Using Previous Parameter Values

To use values of parameters from a previously executed task, the `selValue` attribute in the `Parameter` statement must be set to `autoValue`:

```
Task {
   ...
   Parameter {
      { name = A selValue = autoValue }
      { name = B selValue = autoValue }
   }
   ...
}
```

Evaluating Tasks

To obtain values for each of the responses declared in a task, their source attribute defines how the task will be evaluated. Response values can be set as the result of a simulation process, a mathematical formula, or a tool command language (Tcl) function.

Using Simulation Processes

To use a simulation process, the source attribute of a response can be set to a specific task. Simulation tasks perform the actual interaction with the proper simulation tools. A simulation task can be declared as:

```
Task {
   name = "Simulation"
   type = SIMULATION
   project = "MOS"
   Parameter {
      { name = project_PA type = doe selValue = import.A }
      { name = project_PB type = doe selValue = import.B }
   }
   Response {
      { name = delta }
   }
   Export {
      { name = delta_1 value = delta }
   }
}
```

Here, the project name explicitly references a project of Sentaurus Workbench where a simulation process is defined. The response `delta` will be obtained from the simulation.

A second task defined to recover the values generated by this simulation task can be declared as:

```
Task {
   name = SCR1
   type = SCREENING
   Parameter {
      { name = A type = doe min = -1000 max = 1000 }
      { name = B type = doe min = -1000 max = 1000 }
   }
   Response {
      { name = delta_1 model = standard degree = 1 source = Simulation }
   }
```

```
   doe = plackettBurmann
   nextTask = end
}
```

In the task `SCR1` declaration, the values of the response `delta_1` are obtained from the task named `Simulation`.

NOTE If not declared, a simulation task is always created internally by default. If no simulation task is defined as the source for responses defined in a task, then internally, these responses are obtained by executing the default simulation task, which is related to the Sentaurus Workbench project of the directory where the input command file is located.

Using Formulas and Functions

In some cases, the evaluation of a specific response can be obtained by simply applying a mathematical formula or some calculation function that can use all or some of the declared parameters. In this case, no simulation is performed, thereby obtaining response values almost immediately.

For example, a task can compute `R1` and `R2` using simulation and calculate `RTotal` as `R1+R2`. This feature is also useful when the simulation can be replaced by a previously researched formula and a complete Optimizer run can finish in a fraction of the time involved in simulation runs.

To use a function, it must be declared in the command file. Usually, the definition of the function is declared at the beginning of the command file, using standard syntax for a Tcl code function declaration. An example of this type of declaration is:

```
proc RosenFunction { x1 x2 } {
   set rosen [expr 10 + 100 * pow($x2-$x1*$x1,2) + pow(1-$x1,2)]
   return $rosen
}
```

The task response that is evaluated using this function can be declared as the following, using the keyword `function` in the source value specification:

```
Task {
   ...
   Response {
      { name = Rosen source = "function RosenFunction X1 X2" }
   }
}
```

Moreover, a second response can be obtained as a mathematical formula involving the other response, thereby making both responses dependent on the function evaluation.

This declaration must use the keyword `formula` in the source value specification:

```
Task {
   ...
   Response {
      { name = Rosen source = "function RosenFunction X1 X2" }
      { name = Rosen2 source = "formula Rosen*X1+X2" }
   }
}
```

Response or parameter names can be used directly in a formula.

Importing Partial Results From Previous Tasks

Occasionally, it may be useful to retrieve partial values from either parameter or response values obtained in tasks already performed. They can be used, for example, to define new boundaries for an optimization task based on the results of previous optimization tasks.

The syntax to gain access to these values is:

```
<task_name>:<par|res>:<item_name>
```

where:

- `<task_name>` is the valid name for a task in the project. The indicated task should have been performed before the one in which the user needs to use it.
- `<par|res>` defines whether a parameter (`par`) or a response (`res`) is to be imported.
- `<item_name>` is the name of the parameter or response to be imported.

A valid name for this syntax can be, for example, `Iter1:par:A` (which can be thought of as 'the resulting value for parameter `A` in the task `Iter1`'). There must be no spaces between the composing elements.

This simple syntax can be embedded between more complex expressions by enclosing the entire expression with the `@` character, for example:

```
Parameter {
   { name = A type = doe min = @Iter1:par:A - Iter1:res:Z@
     max = @(Iter1:par:B * 0.05) + 1@ }
   ...
}
```

The value that will be imported into and be replaced in these expressions is the optimum value found for the required parameter or response in the indicated task.

The following example illustrates how to import the results of task `Iter1` for defining the parameter domain in task `Iter2`:

```
Task {
   name = Iter1
   type = ITER_OPTIMIZATION
   project = ""
   Parameter {
      { name = A type = doe min = -5 max = 5 }
      { name = B type = doe min = -20 max = 20 }
      { name = C type = doe min = -100 max = 100 }
   }
   ...
   nextTask = Iter2
}
Task {
   name = Iter2
   type = ITER_OPTIMIZATION
   project = ""
   Parameter {
      { name = A type = doe min = @0.8*Iter1:par:A@ max = @Iter1:par:A*1.2@ }
      { name = B type = doe min = -20 max = 20 }
      { name = C type = doe min = -100 max = 100 }
   }
   ...
}
```

In task `Iter2`, the values for both the `min` and `max` boundaries for parameter `A` are imported as the optimum values obtained for the same parameter (`A`), multiplied by 0.8 and 1.2, respectively.

Design-of-Experiments

The main concept for all tasks is the generation of satisfactory scenarios to run the required tools. This generation is called *design-of-experiments* (DoE). DoE techniques are methods to create a well-defined subset of the parameter domain according to the simulation goal. When multiple parameters (p_1, ..., p_n) exist, the parameter domain is defined as the set of all possible combinations of parameter values. If $V(p_i)$ is the set of possible values for parameter p_i, the parameter domain can be described as the set of possible tuples:

$$PS = \{(v_1, \ldots, v_n) \mid v_i \in V(p_i)\} \quad (1)$$

A DoE is a small set of *special* values that are used to explore relevant subsets of the parameter domain to achieve a simulation goal with the least number of experiments.

Deterministic Design-of-Experiments

Deterministic designs-of-experiments (DoEs) usually consider a subset of the parameter domain where each parameter is inside a given range. The user must define only a minimum and maximum value. Parameter values are then automatically computed according to the selected DoE:

$$DOE = \{(v_1, \ldots, v_n) \mid v_i \in [min_i, max_i]\} \subseteq PS \quad (2)$$

where min_i and max_i are the minimum and maximum values for the parameter p_i.

Optimizer provides some of the most common DoEs [1][2]:

- Full factorial

  For each parameter, a subset of values is selected. The values can be selected either equidistantly or randomly. Simulations run for all combinations formed from these values. A special case is the full factorial design at two levels, where each parameter accepts its minimum and maximum values.

- Half factorial at two levels (+), Half factorial at two levels (–)

  These are two halves of a full factorial design at two levels.

  Figure 2 shows full-factorial and half-factorial designs for three parameters.

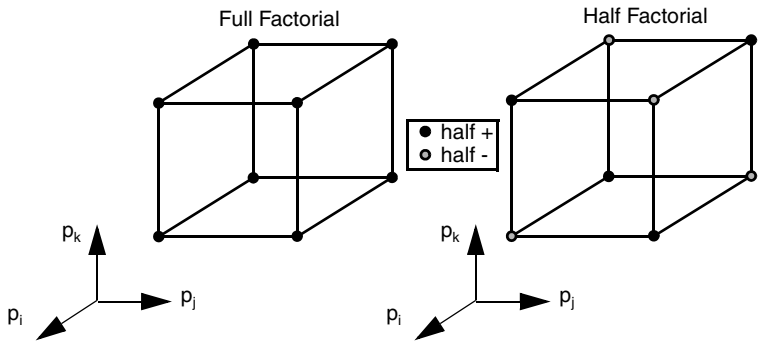

Figure 2 Factorial design-of-experiments

- Fractional factorial at two levels

  These are fractions of a full-factorial design at two levels and are used to reduce the number of simulations. Three criteria are used to select a given fractional factorial design:

  - Number of times the full-factorial design is divided
  - Resolution, which is the type of interaction that must be estimated
  - Number of experiments

Figure 3 shows a fractional-factorial design of resolution III for three parameters.

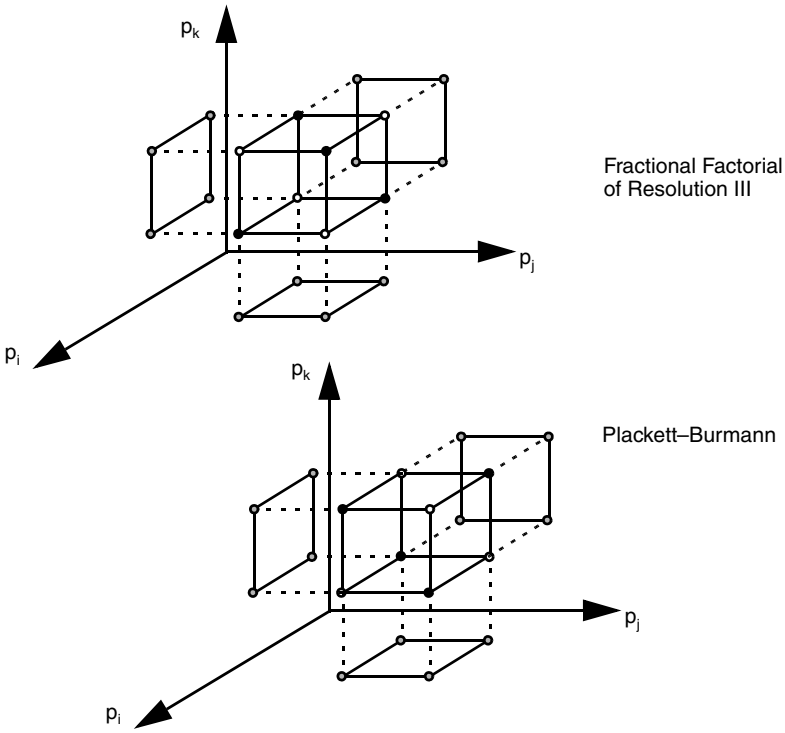

Figure 3 First-order design-of-experiments

Table 1 shows the different combinations of two-level fractional-factorial designs-of-experiments that Sentaurus Workbench supports.

Table 1 Two-level fractional-factorial designs

Parameters	Fraction	Resolution	Experiments
3	1	III	4
4	1	IV	8
5	1	V	16
5	2	III	8
6	1	VI	32
6	2	IV	16
6	3	III	8
7	1	VII	64
7	2	IV	32
7	3	IV	16

Table 1 Two-level fractional-factorial designs

Parameters	Fraction	Resolution	Experiments
7	4	III	8
8	1	VIII	128
8	2	V	64
8	3	IV	32
8	4	IV	16
9	2	VI	128
9	3	IV	64
9	4	IV	32
9	5	III	16
10	3	V	128
10	4	IV	64
10	5	IV	32
10	6	III	16

- Plackett–Burmann

  These designs are special cases of two-level fractional factorial designs for studying $K = N - 1$ parameters in N simulations, where N is a multiple of four. Figure 3 on page 20 shows a Plackett–Burmann design for three parameters.

- Face-centered central composite, small central composite, Box–Behnken

  These designs are special cases of three-level (minimum-center-maximum values) fractional factorial designs, used to fit response surfaces of second-order, running a small number of simulations. Figure 4 on page 23 shows a Box–Behnken and a face-centered central composite design for three parameters.

- Central composite inscribed, central composite circumscribed, orthogonal central composite, custom central composite

  These are special cases of composite designs that construct a design with five levels for each parameter. The custom central composite design allows users to enter the distance of the star point from the center of the design.

- Taguchi

  These designs are based on orthogonal arrays, which are a set of standard fractional factorial experiments.

- Latin square

  For three parameters, a family of simulations is generated, so that all levels of a given parameter are combined in one simulation with all levels of the other parameters.

- Greco-Latin square

  This is analogous to the Latin square, but for four parameters.

- Diagonal

  A number of experiments is selected along the diagonal, across the parameter domain.

- Center points

  Only one experiment is selected. Each parameter takes the middle value between its minimum and maximum values.

- Random

  A number of experiments is selected randomly.

When a DoE is selected, it is generally important to consider the simulation goal. The most common goals are:

- Screening

  Many parameters are usually considered in order to identify the parameters (if any) that have a strong impact on the simulation responses. Fractional factorial designs of resolution III and Plackett–Burmann designs can be used for this purpose. Figure 3 on page 20 shows two screening designs-of-experiments for three parameters: fractional factorial of resolution III and Plackett–Burmann. In this case, both designs are symmetric and equal to a half factorial at two levels (+) design and a half factorial at two levels (–) design, respectively.

- Optimization

  Usually, first-order or second-order approximations of simulation responses are used for optimization. Fractional factorial designs of resolution III are used to estimate first-order models. Face-centered central composite, small central composite, or Box–Behnken designs are used to estimate second-order models. Figure 4 on page 23 shows two DoEs suitable to create second-order RSMs: Box–Behnken and face-centered central composite.

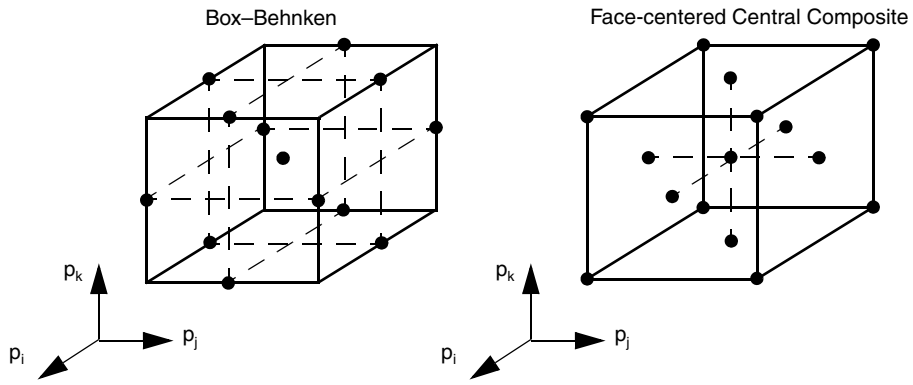

Figure 4 Second-order design-of-experiments

Table 2 lists DoEs used to create second-order polynomial models.

Table 2 Second-order designs

Parameters	Face-centered central composite	Small central composite	Box–Behnken
1	3	3	3
2	9	9	6
3	15	15	13
4	25	17	25
5	43	23	41
6	77	33	49
7	143	43	56
8	273	49	–
9	531	–	–
10	1045	–	–

Stochastic Design-of-Experiments

Stochastic RSMs are used to understand how the variability of parameters affects device behavior. These models are built using a family of simulations that sample those regions of the parameter domain that correspond to events that are more likely to happen. Parameter values are automatically computed when a stochastic design-of-experiments (SDoE) is selected. Only the probability distribution of the parameters must be specified.

The following designs are implemented:

- Monte Carlo

  Experiments are based on a sequence of independent random values for each parameter. The sequences are combined so that the first experiment uses the first value of each independent sequence, the second experiment uses the second value of each independent sequence, and so on. The sequences of random values must be uncorrelated.

- Corner

  Experiments are the full combination of the boundary values of each parameter. The number of simulations is 2^n, where n is the number of parameters.

- Boundary

  This design is based on the mean and boundary values of each parameter. In each simulation, all but one of the parameters are equal to their corresponding mean. The remaining parameter takes one of its boundary values. The number of experiments is $2*n$, where n is the number of parameters.

The mean and boundary values required by corner and boundary designs are arbitrary and depend on the probability distribution associated with each parameter. These values are defined for the two most common distributions: normal and uniform.

Normal distribution is defined by specifying two attributes: mean (μ) and variance (σ^2). Uniform distribution is characterized by the lower bound (a) and upper bound (b) of the considered domain. Table 3 defines the mean and boundary values for these distributions.

Table 3 Mean and boundary values

Probability distribution	Lower bound	Mean	Upper bound
Normal	$\mu - \sigma$	μ	$\mu + \sigma$
Uniform	a	$(a + b) \div 2$	b

- Probabilistic collocation

  To build stochastic RSMs that approximate the simulation responses on the highest probability region of the parameter domain (see Response Surface Models (RSMs) on page 25), it is necessary to have a DoE that samples that specific region. For each parameter p_i, a subset of $n_i + 1$ values is selected. These values are the roots of the orthogonal polynomial of order $n_i + 1$ associated with the probability density function of the parameter. Simulations are run for all combinations of these values. The simulation results are used to build stochastic RSMs for all simulation responses. These models can contain any terms that combine orthogonal polynomial of order up to n_i on the parameter p_i.

Figure 5 shows a probabilistic design for two parameters. One (X_1) has a uniform probability distribution; the other (X_2) has a normal (Gaussian) probability distribution.

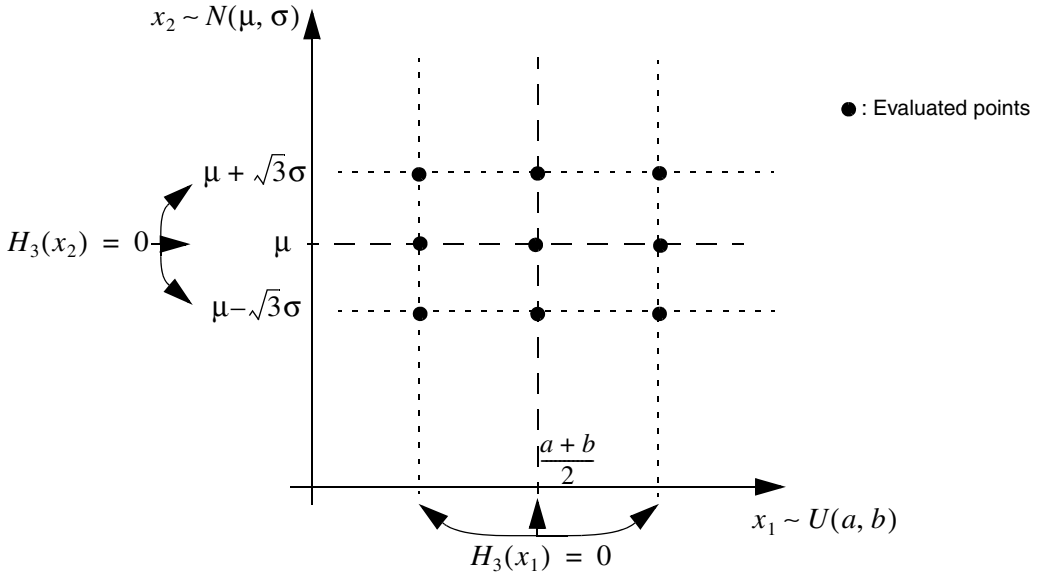

Figure 5 Stochastic design-of-experiments

Response Surface Models (RSMs)

Response surface modeling [3][4] is a technique for creating approximated mathematical models (RSMs) of the simulation responses. These models are used to:

- Determine which parameters have a significant effect on simulation responses (see Screening Task on page 31).
- Determine how parameters affect the behavior of simulation responses.
- Evaluate different parameter settings.
- Find a parameter setting that optimizes a weighted function of the simulation responses (see Optimization Task on page 33, Iterative Optimization Task on page 38, and Generic Optimization Task on page 49).
- Perform sensitivity or uncertainty analysis (see Sensitivity Analysis Task on page 57 and Uncertainty Analysis Task on page 59, respectively).

Model Definition

The options used to customize RSM formation are model, transformation, and degree.

Model

Both deterministic and stochastic models can be defined. Deterministic models are used mainly for optimization and to determine how parameter values affect the simulation responses. Two kinds of deterministic models are provided: standard polynomial models and Kriging models.

Standard polynomial models are computed using the least square method. A Kriging model is a standard polynomial model with an extra function that ensures the RSM passes exactly through all simulated response values. If the RSM is evaluated for a parameter setting that was used to create the model, the result is exactly the same as the simulated response value for that particular parameter setting.

Stochastic RSMs are used for uncertainty analysis. A probability density function must be provided for each parameter that is used to create the stochastic RSM. The terms of the stochastic polynomial model must be specified. These terms are not based on the parameters, as in the deterministic case, but on the orthogonal polynomial associated with the probability density function of the parameters (see Uncertainty Analysis Task on page 59).

Transformation

NOTE This option is considered only for deterministic models.

Usually, the metrics in which data is recorded are chosen for convenient measurement; however, they are not those in which the system is most simply modeled. To achieve the most appropriate scaling, a transformation can be applied to the simulation response. Nonlinear transformations, such as the square root, logarithm, and reciprocal of some (necessarily positive) response Y, expand the scale at one part of the range and contract it at another. Transformations $y = Y^{alpha}$, in which $alpha$ is less than one, contract the range at high values and can be called contractive transformations. Power transformations with $alpha$ greater than one have the reverse effect and can be called expansive. The supported transformations are `log`, `exp`, `sqrt`, and `sqr`. By default, no transformation is used.

Degree

NOTE This option is considered only for deterministic models.

In practice, it is often assumed that a polynomial of first or second degree adequately approximates the true function over a limited region of the parameter domain. First-order, second-order, and third-order polynomial models are supported.

A model cannot be created if there are less than $(n+1)$ linear-independent simulation experiments for a first-degree polynomial model or $(n+1)(n+2)/2$ linear-independent simulation experiments for a second-degree polynomial model.

A general rule is that at least $(n+1)$ or $(n+1)(n+2)/2$ different experiments are required, respectively; this is the only necessary condition. It is strongly recommended to use a family of simulations created by DoE techniques because they consider this condition (see Deterministic Design-of-Experiments on page 19).

Model Information

If an RSM is built, the following information can be computed for each response.

Model Accuracy

Three statistics measure the model accuracy:

- The coefficient of determination (R^2) measures the predictive capacity of the model. It represents the proportion of the variation in the simulation response that can be predicted by changes in the values of the parameters. It ranges from 0 to 1. Expressed as a percentage, it represents the proportion of the real values that can be predicted by the model. It is often used as an overall measure of the fit obtained. The value of R^2 must be as close to 1 as possible.
- A large value of R^2 does not necessarily imply that the regression model is satisfactory. Reducing the number of simulations always increases R^2. To deal with this problem, it is recommended to use an adjusted version of the R^2 statistic (R^2_{adj}). When R^2 and R^2_{adj} differ dramatically, there is a high risk that the model is not sufficiently accurate.
- The estimated variance of the error (S^2) is a measure of the model variability. Clearly, a low value is preferred.

Model Accuracy on page 95 describes the computation of these statistics.

Model Coefficients

The coefficients can be interpreted as either the coefficients of the polynomial function or a measure of the effect of the parameters on the simulation response. In the model information section, the following data is displayed:

Coefficients

The values of the coefficients indicate how much the simulation response differs if a parameter is varied in one unit. These coefficients are also used to measure the effect of each parameter on the simulation responses.

Normalized coefficients

As each parameter has a different domain, it is not possible to compare the previous coefficients directly. It is more effective to work with a normalized range, in which the lower bound and upper bound of the parameter domain are normalized to –1 and 1, respectively. The normalized coefficients are used to determine, by direct inspection, which parameter or parameter interaction has the most influence on the simulation response.

Rank

To simplify the identification of the most significant normalized coefficients, the influences of the different parameters are ranked on a percentage scale.

Model Variance

It is possible to compute which fraction of the total variance is due to each term of the model:

Variance

These values reflect how much of the simulation response variance is associated with each polynomial term.

Standard deviation

The square root of the variance.

Rank

To simplify the identification of the most significant terms, the influences of each polynomial term over the total variance are ranked on a percentage scale.

ANOVA Table

The analysis of variance (ANOVA) table shows standard information on the quality of the model and levels of variability. It also forms a basis for tests of significance. ANOVA Table on page 96 provides a full description of this table.

Histogram

NOTE This information is available for stochastic models only.

A histogram is a discrete description of the estimated probability density function of the simulation response.

Moments

NOTE This information is available for stochastic models only.

Four factors that are based on the first four moments of the estimated probability density function of the simulation response are computed. Moments on page 98 presents formal definitions of these factors.

Model Expressions

Response surface models are polynomial approximations of the simulated function. When the `-expr` option is used, Optimizer shows the polynomial expression in different formats: Tcl expression, Tcl procedure, and Visual Basic function.

Specific Tasks

A task is a sequence of actions used to obtain information about the relationship between the parameters and responses under consideration. More than one task can be performed in a single execution. Each task uses the information obtained in previous tasks. For example, if a task requires the evaluation of a family of simulations that has already been evaluated by a previous task, that family is not re-evaluated.

Different tasks can be defined by describing the following aspects:

- Parameters and responses
- Iterations
- Final analysis tool

Parameters and Responses

The first step is to determine which parameters and responses will be considered. The attributes of parameters and responses can be modified at the beginning of each task. For example, before starting an optimization task, the parameter domain can be modified by redefining the lower and upper bounds of the parameters.

Iterations

A task can consist of one or more iterations. An iteration is defined by the application of one or more of the following consecutive steps:

- Design-of-experiments (DoE)

  At the beginning of any iteration and depending on the task goal, it may be necessary to explore new subregions of the parameter domain by adding new families of simulations. Multiple DoEs can be used to generate the set of parameter settings. After new parameter settings are created, Optimizer automatically calls the batch tools of Sentaurus Workbench to run the simulations and obtain the corresponding simulation responses. If no DoE is selected, Optimizer proceeds directly to the analysis using pre-existing results.

- Response surface models (RSMs)

  Most analyses are performed using RSMs that approximate the real function in a small region of the parameter domain. There are different kinds of RSMs for different tasks.

- Analysis

  Using the RSMs created for each simulation response, the analysis tool obtains information about the relationship between parameters and responses.

- Update

  At the end of each iteration, it may be necessary to update the task information, for example, the region of interest (subset of the parameter domain) or to store the best parameter values that are found.

Final Analysis Tool

Some analyses are performed after all iterations have finished. Usually, such analyses require information collected from all iterations.

Screening Task

Overview

Parameter screening determines how much the different parameters affect the simulation responses. Parameters are ranked according to their influence on each response. Usually, this method is applied in the early stages of analysis. Many parameters are considered in order to identify which ones have the strongest impact or negligible impact on the simulation responses.

This is a single iteration task:

DoE	The designs that are used most often for screening are fractional factorial of resolution III and Plackett–Burmann (see Figure 3 on page 20). These designs create RSMs that are used to obtain the first-order effect of each parameter on the simulation responses.
RSM	First-order polynomial models.
Analysis	When all simulations have been performed, an RSM is built for each response. The normalized coefficients of the RSMs are used to rank parameters according to their effect on the responses.

Command Description

To specify a screening task, parameters and responses are chosen as previously outlined. Specific regions of interest are set for each parameter. The screening method is mainly valid inside the selected region of interest. Linear or logarithmic scale can be specified for each parameter. RSMs are usually set to standard first-order polynomial models.

The screening task has three properties: `screenCrit`, `screenRange`, and `screenBest`.

- `screenCrit`

  The screening criterion indicates how the ranking of parameters is computed. If `screenCrit` is set to `average`, parameters are ranked according to the average impact they have on all responses under consideration. This is the default criterion.

  The influence of a parameter on a response is computed using the normalized coefficient of the corresponding RSM (see Response Surface Models (RSMs) on page 25).

The `average` screening criterion ranks parameters according to the expression:

$$rank_i = \sum_{k \in Responses} I_{ik} \qquad \forall i \in Parameters \tag{3}$$

where $rank_i$ is the parameter p_i ranking and I_{ik} is the influence of parameter p_i on the simulation response R_k.

If `screenCrit` is set to `local_strong`, a high importance is assigned to parameters that have a strong impact on, at least, one response, even if they do not have a strong impact on the remaining parameters.

This criterion ranks parameters according to the expression:

$$rank_i = max(I_{ik}) \qquad \forall i \in Parameters, \forall k \in Responses \tag{4}$$

where $rank_i$ is the parameter p_i ranking and I_{ik} is the influence of parameter p_i on the simulation response R_k.

- `screenRange`

  Screening range is the threshold level that defines whether a parameter is relevant enough to be selected. Parameters whose impact is less than this threshold level are redefined as user-defined parameters and set to their nominal values. This option is given as a percentage of the parameter domain and its default value is 10%.

- `screenBest`

  The screening best criterion is useful for selecting a certain number of parameters after the screening task is performed. Therefore, the number of parameters with the highest impact set for this option are selected for the subsequent tasks, and the remaining ones are redefined as user-defined parameters and are set to their nominal values.

An example of the definition of a screening task is:

```
# Screening task
Task {
   name = 1
   type = SCREENING
   Parameter {
      { name = PW_DOSE
      type = doe min = 1e12 max = 1e13 scale = logarithmic }
      { name = TH_OX
      type = doe min = 15 max = 20 }
      { name = CH_DOSE
      type = doe min = 4e11 max = 4e12 scale = logarithmic }
      { name = LDD_DOSE
      type = doe min = 5e12 max 5e13 scale = logarithmic }
      { name = SP_LENGTH
      type = doe min = 0 max = 300 }
```

```
   }
   Response {
      { name = VT1_WL
      model = standard degree = 1 }
      { name = Vbr
      model = standard degree = 1 }
   }
   doe = plackettBurmann
   screenRange = 2.0
   nextTask = 2
}
```

Output

Screening parameters are ranked using a percentage scale according to their influence on all responses.

The following is an excerpt of output from Optimizer that illustrates how parameters are ranked:

```
-------------------------------------------------------------
Screening result:
-------------------------------------------------------------
Name             Value   Selected
PW_DOSE          1.685      0
TH_OX            0.481      0
CH_DOSE          12.341     1
LDD_DOSE         73.601     1
SP_LENGTH        11.892     1
-------------------------------------------------------------
```

Optimization Task

The key values of a semiconductor device (simulation responses) are influenced by parameters that can be varied within given ranges. The goal of a simulation is often to determine how to set parameters to create a device that meets certain design requirements. This task determines a parameter setting that optimizes a weighted function of the simulation responses. Each task defines an independent optimization problem. Different optimization goals and different parameter domains can be specified in different tasks. If the parameter domain is not modified between the different tasks, no further simulations are required.

This is a single iteration task:

DoE	For optimization purposes, second-order polynomial models are usually used. They require at least three levels for each parameter. The most often employed DoEs are small central composite, face-centered central composite, and Box–Behnken.
RSM	Second-order polynomial models.
Analysis	After all simulations have been performed, a second-order polynomial model is built for each response. Then, an optimization tool is used to obtain parameter values that optimize a weighted function of the response targets.

Optimization Criteria

Multiple optimization criteria can be set, even if they are antagonistic. In such cases, a sensible compromise must be found. Optimization literature describes many approaches to the analysis of multiple responses. The approach of Optimizer gives users control over the importance of all criteria and allows a mathematical optimization using nonlinear optimization solvers. The method uses a global desirability function in which:

- A normalization is applied to each simulation response.
- Different desirability functions are used depending on whether the simulation response is to be minimized, maximized, or has an assigned target value.
- A user-defined weight is assigned to each simulation response.

Each function is normalized using the expression:

$$R'_k = \frac{\left(R_k - \frac{m_k + M_k}{2}\right)}{\left(\frac{M_k - m_k}{2}\right)}, \forall (k \in Results) \quad (5)$$

where:

- R_k is the simulation response k.
- R'_k is the normalized simulation response k.
- m_k is the minimum of the simulation response k.
- M_k is the maximum of the simulation response k.

Using this normalization function, all simulation responses range from –1 to 1 and are considered equally important in the global desirability function. In this way, the global desirability function to be minimized can be expressed as:

$$F = \sum_{k \in minR} w_k \cdot R'_k + \sum_{k \in maxR} -1 \cdot w_k \cdot R'_k + \sum_{k \in targR} T \cdot w_k \cdot (R'_k - T'_k)^2 \quad (6)$$

where:

- w_k is the weight of the simulation response k.
- $minR$ is the set of simulation responses to be minimized.
- $maxR$ is the set of simulation responses to be maximized.
- $targR$ is the set of simulation responses that are required to be as close as possible to a given target.
- T'_k is the normalized target for the simulation response k.
- T is the target constant.

The adjust constant T corrects the anomaly produced by the square in the third sum expression of the desirability function. The range of the terms of the desirability function, corresponding to each response, varies depending on the optimization goal. For minimization and maximization, terms range from –1 to 1. For approximation of a given target, they range from 0 to $max\{ (-1 - T'_k)^2, (1 - T'_k)^2 \}$.

Additionally, these last terms approach zero quadratically, while the other terms decrease linearly. In order that all terms have a similar impact on the global desirability function, the adjust constant T is introduced. T is set to 100 in this implementation.

Optimization Method

The optimization method implemented to solve the general constrained nonlinear problems is based on the sequential quadratic programming (SQP) approach [5][6]. SQP is also known as the projected Lagrangian method or successive (or recursive) quadratic programming. It solves the nonlinear problem using an iterative approach. At each iteration, the problem is approximated by a quadratic problem, which is solved more easily.

This problem is solved by employing the Newton (or quasi-Newton) method to directly find a solution to the Karush–Kuhn–Tucker (KKT) conditions of the original problem. The subproblem solved at each iteration is a minimization of a quadratic approximation to the Lagrangian function, optimized over a linear approximation of the constraints.

These SQP methods work by moving from one feasible point of the parameter domain to another; the second one is a better solution of the optimization problem. The typical strategy is

that at a feasible point x_k, a direction d_k is determined such that for a sufficiently small $\lambda > 0$, the following two conditions are true:

- $x_k + \lambda d_k$ is feasible.
- The objective function value at $x_k + \lambda d_k$ is better than the objective function value at x_k.

After such a direction is found, a one-dimensional optimization is solved to determine how far to proceed along d_k. This leads to a new point x_k and the process is repeated.

At each iteration, the method generates a feasible improving direction and, then, optimizes along that direction. Therefore, the main problem is how to determine this direction. The SQP method creates a quadratic problem to determine this feasible direction by adopting the Newton or quasi-Newton method to solve the KKT optimality conditions. The quadratic problem is solved using the KKT method or the active set method. Whenever possible, the former is used because it is faster and non-iterative.

Nonlinear programming is the general case of optimization problems, in which both the objective function and constraint functions can be nonlinear. This type of problem is the most difficult of the smooth optimization problems. There is no consensus on the best approach; however, the SQP methods have been reported to be among the best [6].

Command Description

An optimization task is defined by selecting which parameters and responses are considered. Regions of interest are set for each parameter. Linear or logarithmic scale can be specified for each parameter. For each response, the polynomial degree of the RSM is selected. Second-order polynomial models are usually used.

The target function is defined according to the following response properties:

`crit`	The three possible criteria are to attain a given target (`closeto`), to minimize the simulation response (`minimal`), or to maximize the simulation response (`maximal`). For the criterion `closeto`, the optimization process tries to find a value for this response that is as close as possible to the required target.
`target`	If the `closeto` criterion is used, this is the optimization goal.
`weight`	It controls the relative importance of a given response.

An example of the definition of an optimization task is:

```
# Optimization task

Task {
   name = 1
   type = OPTIMIZATION
   Parameter {
      { name = PW_DOSE
      type = doe min = 1e12 max = 1e13 scale = logarithmic }
      { name = TH_OX
      type = doe min = 15 max = 20 }
      { name = CH_DOSE
      type = doe min = 4e11 max = 4e12 scale = logarithmic }
      { name = LDD_DOSE
      type = doe min = 5e12 max 5e13 scale = logarithmic }
      { name = SP_LENGTH
      type = doe min = 0 max = 300 }
   }
   Response {
      { name = VT1_WL
      model = standard degree = 2
      crit = closeto target = 0.5 weight = 10
   }
      { name = Vbr
         model = standard degree = 2
         crit = maximal weight = 1
      }
   }
   doe = facedCentralComposite
   nextTask = end
}
```

Output

An excerpt of output from Optimizer that shows the best parameter setting and the estimated value for each simulation response is:

```
-----------------------------------------------------------
Best parameter setting:
----------------------
Evaluation: 4.5405e+2
----------------------------------------
# 1:
----
Parameter values:
PW_DOSE : 1.00e+12
```

```
TH_OX : 15.00
CH_DOSE : 7.82e11
LDD_DOSE : 1.67e+13
SP_LENGTH : 300.00
----
Response values:
VT1_WL : 0.4983
VBR : 8.5721
----
Nodes: 24
-----------------------------------------
```

Iterative Optimization Task

The optimization approach described in Optimization Task on page 33 is satisfactory whenever the parameter domain is relatively small or it is correctly approximated by a second-order polynomial model. In practice, however, the parameter domain is often not small enough, either because of insufficient knowledge of the parameter domain or because the geometry of the parameter domain is not well suited to first-order polynomial models.

A multiple iteration approach can be used in this case. A heuristic search process controls this iterative approach. At each iteration, a new region of the parameter domain is explored to guide the search to the most adequate parameter setting. The heuristic process converges to a local optimum. As in any heuristic process, finding the global optimum is not guaranteed. The optimizer is used to determine a parameter setting that satisfies the constraints on the parameters and optimizes the responses under consideration.

To stop the iterative process, a set of stopping criteria is defined. The criteria consider the quality of the solution, the use of computational resources, and the closeness to a local optimum.

Search Heuristic

The iterative heuristic algorithm for optimization is an approach to cover the parameter domain in search of the optimum. In each step or iteration as described in Basic Concepts on page 5, a section of the whole domain is analyzed and an optimum is searched for. This search consists of a full optimization task, as described in Optimization Task on page 33. The process is based on the response surface methodology. It aims to obtain in each iteration step the most information about the mathematical model presented by the response surface model (RSM). The heuristic component is how the ‘region of interest,’ a subset of the domain for building the RSM, is updated for the next iteration.

Definitions

Current optimum	The best, most current parameter setting. Each iteration that produces a better solution also updates the current optimum.
Search domain	All possible parameter combinations considered for the analysis. It is also known as the parameter domain.
Range	A percentage of the whole search domain.
Region of interest (RoI)	A subset of the search domain used for one iteration. A new RSM is generated for each new RoI, based on the current optimum and range.
Subregion of interest	An expansion of the RoI used to compare new possible local optimum and to determine if the RoI was useful. No new RSM is generated for each subregion.
$\boldsymbol{R}^2_{adj}$	An adjusted version of the coefficient of determination ($\boldsymbol{R}^2$), which measures the predictive capacity or accuracy of the model (RSM).

Algorithm

Initial Setting

The algorithm uses two elements before starting any iteration. They must be initialized for the first iteration:

- Current optimum

  To start, a DoE for the full search domain is created, and it is evaluated to find the best parameter combination. This combination is set as the first current optimum.

  If a previous optimization task already produced simulation results, the starting point is taken from the optimum of those results.

- Range

  It is initialized to 10, representing 10% of the search domain. The value of range is currently fixed to 10 (that is, 10%) in the algorithm and cannot be changed.

Iteration

The following steps are necessary for an iteration:

1. Set region of interest (RoI).

   The current optimum and range are used to build a new RoI with the current optimum as the center and range as the size. If the RoI crosses the search domain border, it is clipped. This means that the DoE and RSM creation consider only valid points within the search domain and the RoI.

2. Select DoE.

   For the RoI, use one of the following DoEs (that is, the first DoE that is feasible as decided by the algorithm and that is currently not being used):

   - Face-centered central composite
   - Box–Behnken
   - Small central composite

3. Select the RSM.

   A second-order polynomial model, built for each simulation response, approximates the simulation responses inside the RoI. The quality of the RSM is indicated by the average of $\boldsymbol{R}^2_{adj}$.

   If the quality of the model is satisfactory ($\boldsymbol{R}^2_{adj} > 0.9$), proceed to Step 4 and try to use it to find a new optimum. Otherwise, go to Step 5.

4. Search for a better current optimum.

   This process searches progressively for the best solution inside the defined RoI for the iteration:

   a) Define a subregion of interest (initially, it is equal to the RoI of the iteration).

   b) Optimize within this subregion using the original RSM, in the same way as the single optimization task also available in Optimizer.

   c) Evaluate the optimal parameter setting obtained and compare it to the current optimum:

      If the new solution is better, update the current optimum and increase the range to 5 (to create a new subregion enlarged by 5%), using the same RSM, then return to Step 4b.

      If the solution is not better, the current optimum is not updated. Go to Step 5.

5. Update the range at the end of the iteration.

   The range can be increased or decreased depending on the following conditions. This can result in enlarging or shrinking the region in the next iteration. If range is increased beyond the search domain, it is automatically corrected to fit it.

   - If no optimization was performed (Step 4 was omitted) due to the insufficient accuracy of the model ($R^2_{adj} \leq 0.9$), then:

     If model is not that inaccurate ($0.7 < R^2_{adj} \leq 0.9$), decrease the range to 10 (in order to try a new subregion with 10% less surface than previously used).

     Otherwise, if $R^2_{adj} < 0.7$, decrease the range to 20.

   - Otherwise, if the iteration did include at least one optimization, then:

     If the best found solution in the iteration is better than the previous best, then:
     – If the model accuracy is high ($R^2_{adj} > 0.95$), increase the range to 3.
     – Otherwise, increase the range to 1.5.

     Otherwise (if the best solution of the iteration was not better than the current optimum), then:
     – If the RSM model is extremely accurate ($R^2_{adj} > 0.98$), it is assumed that no better solution will be achieved for this region. Therefore, increase the range to 2.
     – Otherwise, if $0.9 < R^2_{adj} \leq 0.98$, decrease the range to 10.

6. Check the stopping criteria.

   If any stopping criterion has been reached, stop the iterative optimization; otherwise, go to Step 1.

This algorithm is shown in Figure 6.

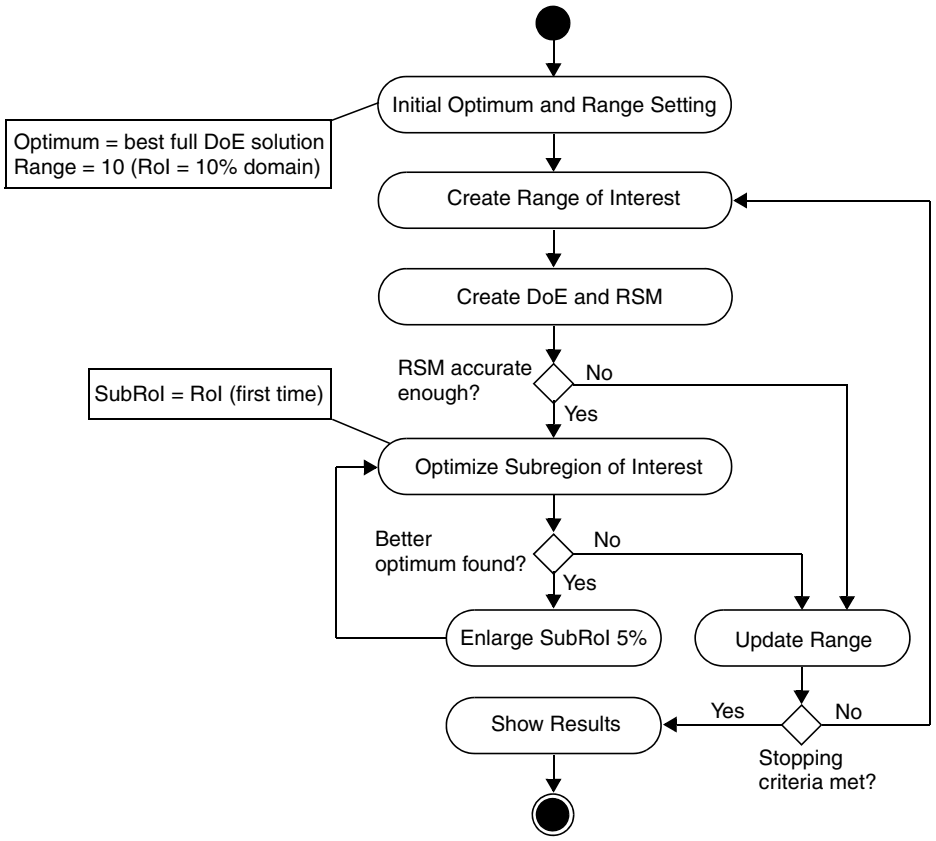

Figure 6 Unified Modeling Language (UML) activity diagram

Stopping Criteria

Stopping criteria include several conditions that stop the heuristic optimizer when any criterion is met. The concept behind these conditions is to avoid iterations that are too long when attempting to find a better solution. In other words, some criteria relate to computational resources and others focus on the quality of the solution found so far.

Computational Resources

`maxTime`	Maximum running time (in seconds) in which to search for an optimum. Default is 3600 = 1 hour.
`maxNumEvaluations`	Maximum number of evaluations before stopping.
`maxNumIterations`	Maximum number of iterations before stopping.

Quality of the Solution

`maxWoImprove`	Maximum number of iterations without improvement before stopping.

Closeness to a Local Optimum

`LocalOpt`	This block defines conditions assumed to be related to a local optimum. A local optimum is defined as an optimum found inside a region of interest that is greater than a particular percentage of the global domain (for example, 10%) and where the average accuracy of the model (measured by the adjusted coefficient of determination) exceeds a given value (for example, 0.98).
Response value range termination	Definition of a range of possible values for any response, which will stop the iterative algorithm if all those responses reach a value within the declared range. For this range declaration, specific keywords are needed: `lowerBound`, `upperBound`, and (alternatively) `perc_range`. A more precise and complete declaration of the range termination feature is in Response Value Range Termination on page 44.

Another stopping condition that is not explicitly declared is finding a solution equal to zero, which means finding a global optimum.

Evaluation Sequence

Currently, the heuristic algorithm executes the evaluation of stopping criteria in the following order. The algorithm stops when it finds the first of these conditions to be true:

1. Is the current number of evaluations equal to or greater than `maxNumEvaluations`?
2. Is the current number of iterations equal to or greater than `maxNumIterations`?
3. Is the current number of iterations without finding a better solution equal to or greater than `maxWoImprove`?

4. Was the maximum process time reached?
5. Is the current solution a local optimum, that is, is $\boldsymbol{R}^2_{adj}$ greater than the maximum defined and is the range greater than defined?
6. Is the best value equal to 0.0 (global optimum)?
7. After checking all of the responses, are they all within the value termination range?

Response Value Range Termination

Optimizer allows users to define a range of possible values for any response, which will stop the iterative algorithm if all those responses reach a value within the declared range. For this declaration, three new keywords have been added to the `Response` syntax: `lowerBound`, `upperBound`, and `perc_range`.

`lowerBound` and `upperBound` represent, respectively, the lower limit and upper limit for the values of the response for stopping the algorithm. Both keywords can replace `target` if it is not declared.

`perc_range` can replace `lowerBound` and `upperBound`, thereby defining a range related to the target value for the optimization.

An example of a response value declaration is:

```
Response {
   { name = delta crit = closeto
      target = 10 lowerBound = 5 upperBound = 10
      weight = 1 }
}
```

This stopping condition takes into consideration the following issues:

- The stopping condition relies on all of the range-defined responses to have values within their respective ranges. Otherwise, the algorithm does not stop.
- The declaration for a range termination depends on the following concepts:
  - `target`: If `target` is specified, two other optional attributes can be used: `lowerBound` and `upperBound`. These represent the lower limit and upper limit for the values of the response for stopping the algorithm. `target` is related to the `closeto` optimization goal.
  - A percentage can also be used for a range (`perc_range`). For example, when defining `target` as 10, and a percentage range as 50%, the actual range is 5 (lower) and 15 (upper). This attribute is declared instead of upper and lower bounds because `perc_range` and its values are expressed in normal percentage values. For example, `perc_range = 70` means a 70% variation from the absolute value of the target.

  - `target` can be omitted. However, if lower and upper bounds are defined, then `target` is set as the average value of both:

    ```
    target = (lowerBound + upperBound) / 2
    ```

  - The range concept can also be used for minimization and maximization. If a response is maximized, `lowerBound` can define the minimal value of that response for which the algorithm can stop. This is analogous for minimization and `upperBound`.
- Validations on declaration:
  - If `target` is omitted when defining a `closeto` optimization goal, `upperBound` and `lowerBound` must be declared.
  - `lowerBound` ≤ `target` ≤ `upperBound`.
  - If `lowerBound = upperBound`, the declaration is still valid but a warning will be sent to the output.
  - If `crit = minimal`, only `upperBound` can be declared for that response. In addition, if `crit = maximal`, only `lowerBound` can be declared for that response.
- If none of `lowerBound`, `upperBound`, and `perc_range` is declared, the standard searching approach will be used and no stopping criterion will be related to the values of those responses.
- The `weight` attribute for each response will still be valid for the optimization approach, but it will have no effect on the evaluation of the stopping criteria.

Command Description

An iterative optimization task is defined by selecting the parameters and responses to be considered. Regions of interest are set for each parameter. Linear or logarithmic scale can be specified for each parameter. For each response, an optimization criterion, a target (when it is required), and a weight are defined (see Command Description on page 36). Specific options of this task are the stopping criteria (see Iterations on page 30).

Additional options are available to store the status of the optimization process before starting an iteration and, if Optimizer is interrupted, to reload that status and continue the optimization process:

`dumpFile`	Name of the file where the status is dumped.
`loadFile`	Name of the file from where the status is reloaded.

The load and dump files can be the same. An example definition of an iterative optimization task is:

```
# Iterative Optimization task
Task {
   name = 1
   type = ITER_OPTIMIZATION
   Parameter {
      { name = PW_DOSE
      type = doe min = 1e12 max = 1e13 scale = logarithmic }
      { name = TH_OX
      type = doe min = 15 max = 20 }
      { name = CH_DOSE
      type = doe min = 4e11 max = 4e12 scale = logarithmic }
      { name = LDD_DOSE
      type = doe min = 5e12 max 5e13 scale = logarithmic }
      { name = SP_LENGTH
      type = doe min = 0 max = 300 }
   }
   Response {
      { name = VT1_WL
      crit = closeto target = 0.5 weight = 10
      }
      { name = Vbr
      crit = maximal weight = 1
      }
   }
   Stop {
      maxTime = 3600
      maxNumEvaluations = 100
      maxNumIterations = 20
      maxWoImprove = 5
      LocalOpt { r2Adj = 0.99 range = 20 }
   }
   loadFile = 1.dump
   dumpFile = 2.dump
   nextTask = end
}
```

Declaration Examples for Range Termination

Target and Bounds Declaration

```
Response {
   { name = VT1_WL crit = closeto
     lowerBound = 0.1 target = 0.35 upperBound = 0.6
     weight = 3 }
   { name = Vbr lowerBound = 10 crit = maximal weight = 1 }
}
```

Target and Percentage Declaration

```
Response {
   { name = VT1_WL crit = closeto
     target = 0.35 perc_range = 71
     weight = 3 }
   { name = Vbr lowerBound = 10 crit = maximal weight = 1 }
}
```

Target Is Omitted, but Both Bounds Are Declared (target = (upper – lower) / 2)

```
Response {
   { name = VT1_WL crit = closeto
     lowerBound = 0.1 upperBound = 0.6
     weight = 3 }
   { name = Vbr lowerBound = 10 crit = maximal weight = 1 }
}
```

Maximization of a Response With a Minimum Limit Declared as Stopping Criterion

```
Response {
   { name = VT1_WL crit = maximal
     lowerBound = 10.0
     weight = 3 }
   { name = Vbr lowerBound = 10 crit = maximal weight = 1 }
}
```

Minimization of a Response With a Maximum Limit Declared as Stopping Criterion

```
Response {
   { name = VT1_WL crit = minimal
     upperBound = 10.0
     weight = 3 }
   { name = Vbr lowerBound = 10 crit = maximal weight = 1 }
}
```

Output

The best parameter setting and the value of each simulation response are returned at the end of this task. Additional information is displayed during the whole optimization process, describing which simulations are performed and how the heuristic moves through the parameter domain.

An excerpt of output is:

```
------------------------------------------------------------
Automatic optimization finished
------------------------------------------------------------
| Local optimum found
---------------------------------------
Model statistics:
----------------
Name               r2     r2Adj  Variance     MSE
VT1_WL             1      1      2.5012       0.869983
VBR                1      1      3.5631       0.754212
------------------------------------------------------------
Best parameter setting:
----------------------
Evaluation: 9.83216e-09
----------------------------------------
# 1:
----
Parameter values:
PW_DOSE : 1.5e+12
TH_OX : 15.00
CH_DOSE : 7.82e11
LDD_DOSE : 1.63e+13
SP_LENGTH : 298.75
----
Response values:
VT1_WL : 0.5002
VBR : 8.9621
----
Nodes: 113
----------------------------------------
```

Generic Optimization Task

Optimization is the search of an optimal value for the objective function within given ranges of the parameters. There are several methods for solving optimization problems. In general, the best-performing method depends on the behavior of the target function. The range of available methods spans from iterative methods based on the derivatives of the target function to local search heuristics (for example, genetic algorithms or random search algorithms). Optimization Problem on page 100 briefly describes optimization problems and the different mathematical methods that have been developed to solve them.

The approaches described in Optimization Task on page 33 and Iterative Optimization Task on page 38 are based on RSMs, which approximate the simulation responses. Since these models are usually first-order or second-order polynomial models whose first and second derivatives can be easily computed, it is possible to use well-known methods for solving general constrained optimization problems that require the computation of the gradient of the target function.

However, Optimizer provides generic methods without using RSMs. To select an adequate method, two facts are considered:

- The problem described in Optimization Task on page 33 can be classified as a bound-constrained optimization problem (see Bound-Constrained Optimization Methods on page 109).
- It is not possible to find analytic expressions or compute the first derivatives of the target function.

The general optimization methods available in Optimizer are:

- Quasi-Newton method applied to the bound-constrained optimization problem
- Nonlinear simplex method

Quasi-Newton Method Applied to Bound-Constrained Optimization Problems

The Newton method has generated a diverse and important class of algorithms that requires the computation of the gradient vector and Hessian matrix (see Mathematical Expressions on page 94) for different parameter settings. If the target function is given by an analytic formula, the first and second derivatives can be calculated directly. However, for a model function formed by results of several simulations, it is impossible to find analytic expressions for the derivatives.

Quasi-Newton methods (see Quasi-Newton Methods on page 107) are based on the Newton method, but approximate the Hessian matrix by recording the gradient differences along each step taken by the algorithm. Optimizer uses finite-difference approximations for computing the gradient.

Nonlinear Simplex Method

The nonlinear simplex method (see Nongradient-Based Methods on page 108) does not require gradient or Hessian evaluations. It performs a pattern search based only on function values. As it makes little use of information about the target function, it typically requires many iterations to find a reasonable solution.

Stopping Criteria

To control these generic optimization processes, the following stopping criteria are set:

`tolerance` — This defines when a local optimum is found. For the quasi-Newton method, if each element of the gradient vector is smaller than the tolerance, it means a local optimum has been found. For the nonlinear simplex method, it represents the fractional convergence tolerance to be achieved by the function value, that is, the smallest difference allowed between the best and worst solutions that integrate the simplex.

`maxNumIterations` — Maximum number of iterations. Several simulations are performed at each iteration of the quasi-Newton method, while only one simulation is run at each iteration of the nonlinear simplex method. Therefore, a larger number of iterations must be allowed for the latter method.

Command Description

A generic optimization task is defined by selecting the parameters and responses to be considered. Regions of interest are set for each parameter. Linear or logarithmic scale can be specified for each parameter. For each response, an optimization criterion, a target (when it is required), and a weight are defined (see Command Description on page 36).

Specific options of this task are the stopping criteria (see Stopping Criteria on page 50). The options for storing and retrieving the status of the optimization process are available (see Command Description on page 45). Additionally, the next option is used to select a particular generic optimization method:

`solver`	Generic optimization method. The available methods are: `bcopt` – Quasi-Newton method for bound-constrained optimization problems. `simplex` – Nonlinear simplex method.

An example of the definition of a generic optimization task is:

```
# Generic optimization task
Task {
   name = 1
   type = GEN_OPTIMIZATION
   Parameter {
      { name = PW_DOSE
      type = doe min = 1e12 max = 1e13 scale = logarithmic }
      { name = TH_OX
      type = doe min = 15 max = 20 }
      { name = CH_DOSE
      type = doe min = 4e11 max = 4e12 scale = logarithmic }
      { name = LDD_DOSE
      type = doe min = 5e12 max = 5e13 scale = logarithmic }
      { name = SP_LENGTH
      type = doe min = 0 max = 300 }
   }
   Response {
      { name = VT1_WL
      crit = closeto target = 0.5 weight = 10 }
      { name = Vbr
      crit = maximal weight = 1}
   }
   Stop {
      tolerance = 1e-5
      maxNumIterations = 100
   }
   solver = bcopt
   nextTask = end
}
```

Output

The best parameter setting and the value of each simulation response are returned. Additional information is displayed during the whole optimization process, describing which simulations are performed and how the heuristic moves through the parameter domain.

```
------------------------------------------------------------
Optimization completed unsuccessfully : Maximum iterations exceeded
------------------------------------------------------------
Best parameter setting:
----------------------
Evaluation: 7.2634e-07
----------------------------------------
# 1:
----
Parameter values:
PW_DOSE : 1.45e+12
TH_OX : 15.00
CH_DOSE : 6.72e11
LDD_DOSE : 3.32e+13
SP_LENGTH : 275.14
----
Response values:
VT1_WL : 0.484
VBR : 8.641
----
Nodes: 672
----------------------------------------
```

Genetic Algorithm Task

This particular version of the genetic algorithm or genetic optimization is based on the standard definition of this algorithm, enhancing a few elements so that you can parameterize the behavior of the searching process according to the specific conditions of the model being optimized.

Some definitions for this context and problem are considered.

- *Chromosome*: In this context, a chromosome represents an experiment, a tuple with one value for each parameter.
- *Fitness function*: This is a particular type of objective function that quantifies the optimality of a solution. The fitness function evaluates the goal (originating from a TCAD simulation) to obtain a normalized range.

- *Mutation*: This is the process of producing a new chromosome from a previous experiment by changing one of the parameters.
- *Crossover*: This is the process of producing a new chromosome by computing a weighted average of each parameter value from two chromosomes, using the fitness function results as its weights.
- *Population*: A population is a set of chromosomes that belong to the same generation, which also is defined as a scenario. Each new generation generates a full population. During the optimization process, the population size is fixed and is defined by the initial scenario.

Main Steps of the Algorithm

The main steps of the algorithm are:

1. Initialization of population (first generation). This is performed using the DoE indicated by the user. Alternatively, this implementation offers a random initialization.
2. Evaluation of each chromosome as a TCAD simulation to determine its fitting quality. This fitting value is the result of a TCAD simulation for a parameter combination and the values of the responses evaluated with the goal function.
3. Application of the selection method on the population related to mating rights. Candidates are chosen whose genetic mix leads to improved candidate solutions in the next generation. In this case, the result is an elite population (by default, only the best experiment is found, but there may be more).
4. Breed the next generation. Randomly select chromosomes from this generation to produce a single child for each pair mated. In this implementation, the child will be more influenced by the parent whose fitting is better. In other words, the child has a closer resemblance to the parent whose fitness is higher.
5. If the `"use mutants"` option is enabled, produce a percentage of the total population (next generation) by mutation. In other words, change the value of a random parameter of a random parent chromosome (selecting an option, the pool of selection for mutation may be of either the elite chromosomes or the total population).
6. In addition, if the next generation has not filled up to complete the defined population size and if the `"use mutants"` option is enabled, produce more chromosomes until the required population is completed.
7. Repeat from Step 2 until finished (see Stopping Criteria on page 54).

Stopping Criteria

The algorithm stops for either of these conditions:

- A maximum number of defined generations is reached.
- A satisfactory response value for all responses with a range definition is reached. This feature is inherited from the iterative optimization method available in Optimizer.

Command Description

To use a genetic algorithm optimization task, the command input file must be defined using the standard command file syntax of Optimizer, as shown in the example at the end of this section, with some mandatory and optional attributes of the task.

Mandatory Attributes

The mandatory attributes are:

- `doe`: Design-of-experiments (DoE) used to generate the first population.
- `iterationsFile`: Output file in CSV format (compatible with Sentaurus PCM Studio) that contains all the experiments for all iterations.
- `nextTask`: The next task to be executed by Optimizer. Declared as in any other Optimizer task.

Optional Attributes

The optional attributes (default values in parentheses) are:

- `maxNumberOfGenerations` (`15`): Maximum number of generations to produce.
- `fitnessFunction` (`sigma`): Fitness function used to evaluate the goal results of the optimization process. The possibilities are:

```
simple: f(X) = 1 - X
sigma : f(X) = 1 / (1 + exp( K*X ))
lineal: f(X) = 1 / (1 + K*X)
stair : f(X) = 1 % ((1/PopulationSize)*index)
```

  where:

  - `X` is the normalized goal value, with (or without) the minimal goal difference (`setFitnessBase`).
  - `K` is a constant of convergence.
  - `index` is the row of the sorted table.

- `elitism` (`1`): The number of chromosomes to be considered elite (`0` for no elitism).
- `constantK` (`5`): Constant of convergence (used by `sigma` and `lineal` fitness functions).
- `setFitnessBase` (`1`): If set to `1`, it normalizes the goal only with the difference between goals (`0` is the best). If set to `0`, it normalizes the goal values as they are.
- `nMutants` (`20`): Percentage of the total population to be generated by mutation (not crossover).
- `percMutation` (`10`): Percentage of mutation of the values.
- `eliteMutants` (`1`): If set to `1`, it generates mutated chromosomes only from elite members. If set to `0`, it generates mutated chromosomes from any chromosome of the population.
- `maxFalseChildAttempts` (`100`): The maximum number of attempts to generate a new chromosome (child) by crossover. This condition is used because the algorithm always attempts to generate new chromosomes and, by limiting the number of attempts to produce new children, it is possible to avoid infinite loops.
- `velocity` (`1`): This parameter is used to make different combinations between the fitness function evaluation and the selection crossover method. This parameter controls how quickly (in how many generations) the population converges to an optimum. If set too fast, some potentially good areas (but incidentally, with poor experiments) may be omitted too early. The options are:

  - `1`:

    Fitness function: Smallest goal value → biggest fitness value.

    Crossover: Change more those that have the biggest fitness value.

  - `2`:

    Fitness function: Smallest goal value → biggest fitness value.

    Crossover: Change more those that have the smallest fitness value.

  - `3`:

    Fitness function: Biggest goal value → smallest fitness value.

    Crossover: Change more those that have the smallest fitness value.

  - `4`:

    Fitness function: Biggest goal value → smallest fitness value.

    Crossover: Change more those that have the biggest fitness value.

An example definition of a genetic optimization task is:

```
Task {
   name = Task1
   type = GENETIC_OPTIMIZATION
   project = ""
   Parameter {
      { name = A type = doe min = -100 max = 100 }
      { name = B type = doe min = -100 max = 100 }
      { name = C type = doe min = -100 max = 100 }
      { name = D type = doe min = -100 max = 100 }
   }
   Response {
      { name = delta model = standard degree = 1 crit = minimal }
   }

   doe = facedCentralComposite
   iterationsFile = generations.csv
   maxNumberOfGenerations = 10
   nextTask = end
}
```

Files Generated by Genetic Algorithm

Two major output files are generated.

First, there is the standard output file of Optimizer: `gopt.log`. This file includes a description (including numeric data) that explains the iterative optimization process that the genetic algorithm performs and how a generation evolves to the next, finishing with an absolute best solution found.

NOTE Due to the nature of the genetic algorithm and the types of problem it addresses, it is not possible to guarantee that the best solution found is the absolute optimum.

The second and more interesting output file is `generations.csv`. This data file contains all the experiments analyzed (and simulated) during the optimization process. For each experiment, an iteration number references the generation in which the experiment was used. The format of this file makes it easy to view the results using CSV-compatible data software (such as Microsoft® Excel) or Sentaurus PCM Studio.

Sensitivity Analysis Task

Two main types of uncertainty affect confidence in the results of a simulation tool: structural uncertainty (inaccurate models) and parametric uncertainty. Parametric uncertainty arises from incomplete knowledge of model parameters such as empirical quantities, defined constants, and stochastic parameters (for example, due to manufacturing variations). Refining measurements of input parameters reduce parametric uncertainty. Both sensitivity analysis and uncertainty analysis are used to understand how the variability of parameters affects device behavior.

Sensitivity analysis aims at analyzing the model outputs as a function of very small changes of a single parameter, with all of the other parameters fixed. Therefore, sensitivity analysis only reveals the local gradient of the response surface of the model with regard to a given parameter.

This is a multiple iteration task. One iteration is required for each parameter of interest. At each iteration, these actions are performed:

DoE	All but one of the parameters are fixed to a given nominal value. For the remaining parameter, a simple DoE is used. This design selects some values inside a small range around its nominal value. It corresponds to a diagonal design of n levels. For three levels, the design is formed by the lower bound, upper bound, and nominal value.
RSM	RSMs are not required.
Analysis	On completion of all simulations, the influence of the parameter on each simulation response (response variation) is computed.

The influence of a certain parameter p_i on a particular response R_k is computed using the expression:

$$V_{ik} = \frac{\sum_{j=1}^{n} \left\{ \left| RV_{ikj} - RC_{ik} \right| \times \left| \frac{c_i - m_i}{c_i - v_{ij}} \right| \right\}}{\left| RC_{ik} \right| \times \sum_{j=1}^{n} \left| \frac{c_i - m_i}{c_i - v_{ij}} \right|} \times 100 \qquad \forall i \in Parameters, \forall k \in Responses \tag{7}$$

where:

- n is the number of levels.
- RV_{ikj} is the value of the simulation response R_k when parameter p_i is set to the level j.
- RC_{ik} is the value of the simulation response R_k when parameter p_i is set to the nominal value.

- c_i is the nominal value of parameter p_i.
- m_i is the lower bound of parameter p_i.
- v_{ij} is the value of parameter p_i for the level j.

For the simple case of only two levels (lower bound and upper bound), the expression becomes the well-known expression:

$$V_{ik} = \frac{|Rm_{ik} - RC_{ik}| + |RM_{ik} - RC_{ik}|}{2|RC_{ik}|} \times 100 \qquad \forall i \in Parameters, \forall k \in Responses \tag{8}$$

where:

- Rm_{ik} is the value of the simulation response R_k when parameter p_i is set to its lower bound.
- RM_{ik} is the value of the simulation response R_k when parameter p_i is set to its upper bound.

Command Description

A sensitivity analysis task is defined by selecting which parameters and responses are considered. The nominal set of parameter values and the range of variation around that particular parameter setting are specified. Linear or logarithmic scale can be specified for each parameter. The following specific options are available:

`nPoints`	Number of considered points inside the sensitivity range. Default is 3.
`range`	Percentage range of the whole domain used for sensitivity analysis. Default is 10%.

An example of the definition of a sensitivity analysis task is:

```
# Sensitivity Analysis task
Task {
   name = 1
   type = SENS_ANALYSIS
   Parameter {
      { name = PW_DOSE
      selValue = 5e12 type = doe min = 1e12 max = 1e13 scale = logarithmic }
      { name = TH_OX
      selValue = 17.5 type = doe min = 15 max = 20 }
      { name = CH_DOSE
      selValue = 9e11 type = doe min = 4e11 max = 4e12 scale = logarithmic }
      { name = LDD_DOSE
      selValue = 1e13 type = doe min = 5e12 max = 5e13 scale = logarithmic }
```

```
      { name = SP_LENGTH
      selValue = 150 type = doe min = 0 max = 300 }
   }
   Response {
      { name = VT1_WL weight = 10 }
      { name = Vbr    weight = 1 }
   }
   nPoints = 3
   range = 1.0
   nextTask = end
}
```

Output

An excerpt of the final output of the sensitivity analysis task for the response `VT1_WL` is:

```
-------------------------------------------------------------
Sensitivity Analysis     - range (1.0%)
---------------------------------------------
sensitivity parameter : PW_DOSE
response VT1_WL: 0.096 : 2.73 %
------------------------------
sensitive parameter : TH_OX
response VT1_WL: 0.061 : 1.75 %
------------------------------
sensitive parameter : CH_DOSE
response VT1_WL: 0.053 : 1.50 %
------------------------------
sensitive parameter : LDD_DOSE
response VT1_WL: 0.075 : 2.14 %
------------------------------
sensitive parameter : SP_LENGTH
response VT1_WL: 0.124 : 3.53 %
```

Uncertainty Analysis Task

Uncertainty analysis [7][8] is used to understand how the variability of parameters affects device behavior. It employs a multidimensional RSM, called stochastic RSM, which reflects the collective uncertainty of all parameters under consideration. Therefore, uncertainty analysis yields a more complete picture of the uncertainty propagation in a model. Further, sensitivity analysis is regarded as a special case of uncertainty analysis.

Given the uncertainty descriptions of the parameters (their probability density functions), any simulation response can be approximated by a polynomial function that is mathematically

tractable. To determine this function, a sequence of polynomials is associated with each parameter. These polynomials are mutually orthogonal when integrated using the probability density function as weight function. The model function is constructed as a selected sum of the products of orthogonal polynomials, where each product is multiplied by an undetermined coefficient. The coefficients are estimated by least squares, using the results of a family of simulations. The corresponding set of parameter values is given by the zeros of selected orthogonal polynomials.

After the model function is computed, it is used to determine correlations between parameters and simulation responses, to perform variance analysis and sensitivity analysis, to extract statistical moments, and, ultimately, to extract the probability density functions of the simulation responses.

Some empirical benchmarks [7] have shown that this approach is potentially a factor of 25 to 60 times faster than the pure Monte Carlo method. A more computationally demanding simulation process results in a larger factor. This method converges exponentially with increasing orders of polynomial expansions.

Mathematical Background

Each parameter p_i has an associated probability density function $f_i(p_i)$ defined over the interval $(a_i \le x_i \le b_i)$ that can be used as a weight function to generate an associated sequence:

$$\left\{ H_i^{r_1}(p_i), \ldots, H_i^{r_k}(p_i) \right\} \tag{9}$$

of polynomials of degree $r_i \ge 0$ that are mutually orthogonal when integrated over the domain $a_i \le x_i \le b_i$. That is, the polynomials satisfy the orthogonality condition:

$$\int_{a_i}^{b_i} f_i(x_i) H_i^{r_j}(p_i) H_i^{r_k}(p_i) dp_i = C_i \delta_{r_j, r_k} \qquad \forall r_j \forall r_k \tag{10}$$

where C_i is a positive constant and δ_{r_j, r_k} is the Kronecker delta defined as:

$$\delta_{r_i, r_j} = \begin{cases} 0 & \text{if} \quad r_i \neq r_j \\ 1 & \text{if} \quad r_i = r_j \end{cases} \tag{11}$$

The sequence of orthogonal polynomials depends on the domain $(a_i \le x_i \le b_i)$ and the weight function $f_i(x_i)$. In this case, the domain depends on the probability density function used as weight function.

For a given simulation response $Y(p_1, \ldots, p_n)$, a stochastic RSM $\hat{Y}(p_1, \ldots, p_n)$ is constructed as a sum of polynomial products:

$$H_i^{r_n}(p_1) \cdot \ldots \cdot H_i^{r_n}(p_n) \text{ where } 0 \le r_i \le D_i, \forall i = 1,\ldots,n \tag{12}$$

each multiplied by a coefficient, which is to be computed:

$$\hat{Y}(p_1, \ldots, p_n) = \sum_{\forall(r_1, \ldots, r_n) \mid (0 \le r_i \le D_i, \forall i = 1, \ldots, n)} A_{r_1, \ldots, r_n} \prod_{i=1}^{n} H_i^{r_i}(p_i) \tag{13}$$

The coefficients are estimated by least squares, using the results of a family of simulations in which the parameter values are determined by the zeros of the orthogonal polynomials of order $D_i + 1$.

If the built stochastic RSM is satisfactorily accurate, according to the model statistics described in Response Surface Models (RSMs) on page 25, an approximation of the probability density function of the simulation response can be obtained using the Monte Carlo method. Optimizer computes a histogram of the probability density function and some coefficients that depend on the first four moments of the probability density function (see Model Information on page 27).

Example

Consider an example with two parameters, p_a and p_b, and one simulation response $Y(p_a, p_b)$. p_a has a uniform probability density function over the domain (l_a, u_a) and p_b has a normal (Gaussian) probability density function with mean μ_b and standard deviation σ_b (see Figure 7).

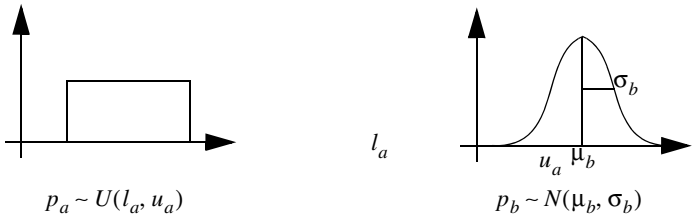

Figure 7 Stochastic parameters

If a stochastic RSM with only first-order and second-order orthogonal polynomials ($D_1 = 2$ and $D_2 = 2$) is created for the response Y, a family of simulations is defined by selecting the zeros of the third-order orthogonal polynomials for each parameter and computing all the possible combination of those parameter values:

$$\begin{aligned} &H_a^3(p_a) = 0 \rightarrow \{p_a^1, p_a^2, p_a^3\} \\ &H_b^3(p_b) = 0 \rightarrow \{p_b^1, p_b^2, p_b^3\} \end{aligned} \quad \left| \begin{array}{l} \text{Family of Simulations} \\ (p_a^1, p_b^1),(p_a^1, p_b^2), (p_a^1, p_b^3) \\ (p_a^2, p_b^1),(p_a^2, p_b^2), (p_a^2, p_b^3) \\ (p_a^3, p_b^1),(p_a^3, p_b^2), (p_a^3, p_b^3) \end{array} \right. \tag{14}$$

After running the corresponding simulations, the stochastic RSM can be constructed as:

$$\begin{aligned} \hat{Y}(p_a, p_b) = A_{0,0} + \ldots \\ A_{1,0}H_a^1(p_a) + A_{0,1}H_b^1(p_b) + \ldots \\ A_{2,0}H_a^2(p_a) + A_{1,1}H_a^1(p_a)H_b^1(p_b) + A_{0,2}H_b^2(p_b) + \ldots \\ A_{2,1}H_a^2(p_a)H_b^1(p_2) + A_{1,2}H_a^1H_b^2(p_b) + \ldots \\ A_{2,2}H_a^2(p_a)H_b^2(p_b) \end{aligned} \tag{15}$$

where $A_{0,0}, A_{1,0}, A_{0,1}, A_{2,0}, A_{1,1}, A_{0,2}, A_{2,1}, A_{1,2}, A_{2,2}$ are the still undetermined coefficients.

The coefficients result from a least square fit. Figure 8 shows the family of simulations and some isolines of the stochastic RSM.

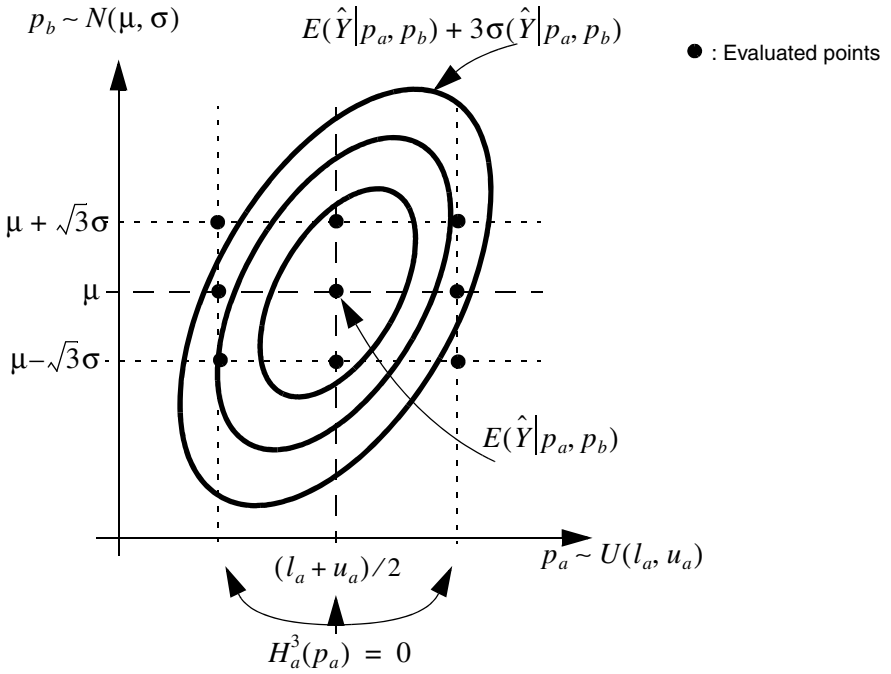

Figure 8 Stochastic response surface model

This is a single iteration task:

DoE	A stochastic DoE is a full factorial design at a given number of levels for each parameter. The number of levels used for each parameter is equal to the maximum orthogonal polynomial degree that is required for use with this parameter plus one. For example, if orthogonal polynomials of first-order and second-order are used for a given parameter, the stochastic DoE considers three levels for such a parameter.
RSM	When all simulations have been performed, a stochastic RSM is built for each response.
Analysis	Using the Monte Carlo method for each stochastic RSM, a histogram and the first four moments of the estimated probability density function are computed.

Command Description

Uncertainty analysis requires parameters and responses. For each parameter, a probability density function is provided. The arguments of this function must be set. For example, if the normal (Gaussian) probability density function is selected, two arguments are required: mean and variance. An additional argument specifies the number of considered split points.

For each response, a stochastic RSM, which is a sum of terms, is created. Each term contains one orthogonal polynomial per parameter. The orthogonal polynomial of order zero is always equal to 1. Therefore, the constant term is a special case where the orthogonal polynomial of order zero is used for all parameters.

Although, all stochastic designs-of-experiments can be used, only the probabilistic stochastic DoE is actually satisfactory for uncertainty analysis. This design is a full factorial DoE where the parameter values are determined by the roots of the orthogonal polynomial, whose order is equal to the number of required levels for each parameter. It is also possible to import a stochastic design from a comma-separated value (CSV) file (see Stochastic Design-of-Experiments Task on page 69).

The following options are specific for this task:

`sampleSize`	Monte Carlo sample size.
`exportStat`	This option is used to export information about the stochastic RSMs to a file (in tab-delimited format). The argument of this option is the file name. For each simulation response, the following data is exported: all values generated using the Monte Carlo method, coefficients of the stochastic model, some statistics that measure its accuracy, the histogram, and the first moments of the estimated response probability distribution.

The option `save`, which is available for stochastic DoE tasks (see Stochastic Design-of-Experiments Task on page 69), is also available for this task. An example of the definition of an uncertainty analysis task is:

```
# Uncertainty Analysis task
Task {
   name = 1
   type = UNC_ANALYSIS
   Parameter {
      { name = PW_DOSE
      type = sdoe sdoeArgs = 2
      SDoEModel { type = normal args = {5.5e+12 3.025e+23} }
      }
      { name = TH_OX
```

```
      type = sdoe sdoeArgs = 2
      SDoEModel { type = normal args = {17.5 3.0625} }
      }
      { name = CH_DOSE
      type = sdoe sdoeArgs = 2
      SDoEModel { type = normal args = {2.2e+12 4.84e+22} }
      }
      { name = LDD_DOSE
      type = sdoe sdoeArgs = 2
      SDoEModel { type = normal args = {2.75e+13 7.5625e+24} }
      }
      { name = SP_LENGTH
      type = sdoe sdoeArgs = 2
      SDoEModel { type = normal args = {150 225} }
      }
   }
   Response {
      { name = VT1_WL
        model = stochastic
        modelArgs = { {} {PW_DOSE} {TH_OX} {CH_DOSE} {LDD_DOSE} {SP_LENGTH}
                    {PW_DOSE TH_OX} {PW_DOSE CH_DOSE} {PW_DOSE LDD_DOSE}
                    {PW_DOSE SP_LENGTH} {TH_OX CH_DOSE} {TH_OX LDD_DOSE} {TH_OX
SP_LENGTH}
                    {CH_DOSE LDD_DOSE} {CH_DOSE SP_LENGTH} {LDD_DOSE
SP_LENGTH} }
      }
      { name = Vbr
        model = stochastic
        modelArgs = { {} {PW_DOSE} {TH_OX} {CH_DOSE} {LDD_DOSE} {SP_LENGTH}
                    {PW_DOSE TH_OX} {PW_DOSE CH_DOSE} {PW_DOSE LDD_DOSE}
                    {PW_DOSE SP_LENGTH} {TH_OX CH_DOSE} {TH_OX LDD_DOSE} {TH_OX
SP_LENGTH}
                    {CH_DOSE LDD_DOSE} {CH_DOSE SP_LENGTH} {LDD_DOSE
SP_LENGTH} }
      }
  }
  sdoe = probabilisticDesign
  sampleSize = 10000
  Save {
    file = uaDOE.tab
    elems = {values experiments moments corr_matrix}
  }
  exportStat = uaStTable.tab
  exportTable = uaTable.tab
  nextTask = end
}
```

Output

An excerpt of the final output of an uncertainty analysis task for the response `VT1_WL` is:

```
------------------------------------------------------------
Response : VT1_WL
----------------------------------------
Model Coefficients:
-------------------
Term                            Coefficient           Norm. Coeff.         Rank
constant                         +3105140.000000     +3105140.000000       90.766
H1(PW_DOSE)                        -12531.100000       -12531.100000        0.366
H1(SP_LENGTH)                     -289157.000000      -289157.000000        8.452
H1(TH_OX)                            -345.420000         -345.420000        0.010
H1(LDD_DOSE)                       -10841.000000       -10841.000000        0.317
H1(PW_DOSE)*H1(SP_LENGTH)             +11.758300          +11.758300        0.000
H1(PW_DOSE)*H1(TH_OX)               +2526.110000        +2526.110000        0.074
H1(PW_DOSE)*H1(LDD_DOSE)               +0.166421           +0.166421        0.000
H1(SP_LENGTH)*H1(TH_OX)                +0.497907           +0.497907        0.000
H1(SP_LENGTH)*H1(LDD_DOSE)           +502.516000         +502.516000        0.015
H1(TH_OX)*H1(LDD_DOSE)                 +0.007071           +0.007071        0.000

----------------------------------------
Model statistics:
-----------------
Coeff. of determination: 0.995487
Adjusted coeff. of deter.:0.994842
Variance of error:        3.38875e+009
MSE:                      2.92855e+009
----------------------------------------
Model Variance:
---------------
Term                             Variance              Standard Deviation    Rank
constant                            +41836500.000000         +6468.114099     44.354
H1(PW_DOSE)                         +20918300.000000         +4573.652807     22.177
H1(SP_LENGTH)                       +20918100.000000         +4573.630943     22.177
H1(TH_OX)                              +85380.400000          +292.199247      0.091
H1(LDD_DOSE)                           +10459.100000          +102.269741      0.011
H1(PW_DOSE)*H1 (SP_LENGTH)          +10459000.000000         +3234.037724     11.088
H1(PW_DOSE)*H1(TH_OX)                  +42690.300000          +206.616311      0.045
H1(PW_DOSE)*H1 (LDD_DOSE)               +5229.560000           +72.315697      0.006
H1(SP_LENGTH)*H1 (TH_OX)               +42689.900000          +206.615343      0.045
H1(SP_LENGTH)*H1 (LDD_DOSE)             +5229.510000           +72.315351      0.006
H1(TH_OX)*H1 (LDD_DOSE)                   +21.345100            +4.620076      0.000
```

```
----------------------------------------
ANOVA table:
-----------
Source Deg.   Freedom    Sum of Squares   Mean Square   F
Model         10         +5.23e+013       +5.23e+012    +1544.060000
Error         70         +2.37e+011       +3.39e+009
Total         80         +5.26e+013       +6.57e+011
----------------------------------------

Model Histogram:
---------------
       Min                       Max                          Frequency
              +1298080.000000          +1668442.727273                37
              +1668442.727273          +2038805.454546               499
              +2038805.454546          +2409168.181819              1346
              +2409168.181819          +2779530.909092              1719
              +2779530.909092          +3149893.636365              1696
              +3149893.636365          +3520256.363638              1680
              +3520256.363638          +3890619.090911              1556
              +3890619.090911          +4260981.818184              1027
              +4260981.818184          +4631344.545457               371
              +4631344.545457          +5001707.272730                62
              +5001707.272730          +5372070.000003                 7
----------------------------------------
Model moments:
-------------
Moment     Value
Mean       +0.5210321
Variance   +4.80e+011
Skewness   +0.100766
Kurtosis   -0.783261
----------------------------------------
```

Design-of-Experiments Task

Command Description

To specify a DoE task, parameters must be chosen, and a DoE type and its corresponding arguments must be set. DoE arguments can be set for either each parameter (as in the following example) or the whole task.

The following option must be set:

`doe` — This argument is used to select a particular DoE. The available designs are `fullFacNLev`, `halfFactorialMinus`, `halfFactorialPlus`, `fractFactorial2`, `plackettBurmann`, `boxBehnken`, `facedCentralComposite`, `smallCentralComposite`, `taguchi`, `latinSquare`, `grecoLatinSquare`, `diagonalDesign`, `midPointDesign`, `randomDesign`.
See Deterministic Design-of-Experiments on page 19.

The following option is also available:

`tree` — This option is used to specify whether the simulation tree is modified. The following keywords are added to the list if any of these actions are required:
`none` – Do not add the parameter settings to either the experimental table or simulation tree.
`add` – Add the parameter settings to the simulation tree.
`eval` – Run simulations to evaluate the new branches. This keyword is redundant if the keyword add is not included in the list of arguments.

The default behavior is to add the new experiments to the experiment table and leave the simulation tree unchanged. An example of the definition of a DoE task is:

```
# DoE task: full factorial
#
# doe arguments for each parameter:
# first argument: selection mode (equidistant or random)
# second argument: whether include interval boundaries
# third argument: whether include interval center point
# fourth argument: number of levels
Task {
   name = 1
   type = DOE
```

```
    Parameter {
       { name = A type = doe doeArgs = {random 1 0 5} }
       { name = B type = doe doeArgs = {random 0 1 3} }
       { name = C type = doe doeArgs = {random 0 0 4} }
       { name = D type = doe doeArgs = {random 0 1 2} }
    }
    doe = fullFacNLev
    nextTask = 101
}
```

Output

Optimizer outputs a list of all the different parameter settings that belong to the families of simulations that were created using the specified DoE technique.

Stochastic Design-of-Experiments Task

Command Description

To specify an SDoE task, parameters are selected and their probability distribution functions are set in terms of the corresponding arguments. Additionally, the following argument is available for each parameter:

`sdoeArgs` — The SDoE argument must be set for the design:

`montecarloDesign` – A value that specifies the number of split points to be generated for each parameter.

The following option must be set:

`sdoe` — This argument is used to select a particular stochastic DoE. The available designs are `montecarloDesign`, `cornerDesign`, `boundaryDesign`, `probabilisticDesign` (see Stochastic Design-of-Experiments on page 23). It is also possible to import the design from a CSV file using the keyword `fromTable`.

The following options are also available:

`sdoeArgs` This argument is used to specify some design-specific information for the following design:

`montecarloDesign` – A value that indicates the sample size of the sequence of random values that are generated for each parameter.

`fromTable` – A value that indicates the sample size of the sequence of random values that are generated for each parameter. A list contains the file name and the character used in the file to separate values. For example, the list `{"mydoe.csv" ","}` indicates the design will be imported from the CSV file `mydoe.csv`.

NOTE The CSV file must have column names in the first row, but the names can be empty. Each parameter or response name in the project must match the name of only one column in the CSV file.

`save` This option is used to export SDoE information to a file (in tab-delimited format). The first argument of this option is the file name; the second one is a list of keywords that select the output data:

`values` – List of parameter values.

`experiments` – List of parameter settings created using the DoE.

`moments` – First moments (mean, variance, skewness, and kurtosis) of the sequence of values created for each parameter.

`corr_matrix` – Both the correlation matrix and a normalized version of the correlation matrix are exported to the file. The normalized correlation matrix is created by transforming all sequences of values as if all parameters have standard normal distributions.

The option `tree`, which is available for the DoE task, is also available for this task (see Design-of-Experiments Task on page 68). An example of the definition of an SDoE task is:

```
# SDoE task: Monte Carlo
#
# arguments:
# number of experiments
Task {
   name = 1
   type = SDOE
   Parameter {
```

```
      { name = A type = sdoe }
      { name = B type = sdoe }
      { name = C type = sdoe }
      { name = D type = sdoe }
   }
   sdoe = montecarloDesign
   sdoeArgs = 1000
   tree = {none}
   Save {
      file = sdoeMC.tab
      elems = {values experiments moments corr_matrix}
   }
   nextTask = 2
}
```

Output

Optimizer outputs a list with all the different parameter settings that belong to the families of simulations that were created using the specified SDoE technique.

Custom Task

The custom task is a simple task type that allows the specification of a particular algorithm, which can rely on the internal structures and procedures of Optimizer to perform a particular type of optimization or other data analysis.

This task is useful when it is necessary to explore and implement a different optimization or analysis process, which is not available in the standard task types.

Command Description

To specify a custom task, standard parameter and response declarations are valid, and a few other arguments are available for the task declaration:

`doe`	This argument defines the design to be used if necessary.
`Algorithm`	This is the most important section. Within the `Algorithm` block, any standard Tcl script is allowed. Usually, to take advantage of the internal structures and procedures of Optimizer, a few namespaces are available.

An example of a custom task declaration is:

```
Task {
   name = 1
   type = CUSTOM
   project = ""
   Parameter {
      { name = A type = doe min = -100 max = 100 }
      { name = B type = doe min = -100 max = 100 }
      { name = C type = doe min = -100 max = 100 }
      { name = D type = doe min = -100 max = 100 }
   }
   Response {
      { name = delta model = standard degree = 1 }
   }

   Algorithm {
      upvar $TABLE Table

      set taskId $id
      set parId [Cget $taskId parId]
      set doeType [Cget $taskId doe]
      set doeArgs [Cget $taskId doeArgs]
      set parNames [parData::Cget $parId names]

      OPT:WriteLog "======================================================="
      OPT:WriteLog " DoE creation, using $doeType:"

      set doeId [doe::New $parNames]
      Task::CallDoE $parId $doeId "DOE" $doeType $doeArgs
      set experimentList [doe::Cget $doeId Expers]

      OPT:WriteLog " $experimentList"
      OPT:WriteLog "======================================================="

      foreach pname $parNames {
         foreach pvalue $experimentList {
            parData::Config $parId autoValue $pname $pvalue
         }
      }
   }
   doe = facedCentralComposite
   nextTask = end
}
```

Output

Output and behavior depend on the specific instructions included in the `Algorithm` section.

Integration of Sentaurus Workbench

Sentaurus Workbench and Optimizer are closely related. Assuming Optimizer is the main analysis tool, Sentaurus Workbench provides some features that allow users to edit and view command files and output files, and to run Optimizer directly using explicit menu options. These features are accessible through the **Optimization** menu of Sentaurus Workbench.

Sentaurus Workbench Scenarios

Some naming conventions have been adopted for the generation of scenarios, which allow users to control the execution of the different tasks by dynamically editing scenarios while Optimizer is running.

Optimizer creates scenarios using the naming scheme `<task_id>_<sc_counter>` where `<task_id>` is a task-dependent identifier as shown in Table 4 and `<sc_counter>` is a nonnegative integer that is increased dynamically so as not to overwrite previously created scenarios.

Table 4 Task-dependent identifiers for scenario naming

Task	Task identifier	Task	Task identifier
Design-of-experiments	`doe`	Uncertainty analysis	`unc`
Stochastic design-of-experiments	`sdoe`	Optimization	`opt`
Screening	`screen`	Iterative optimization	`itropt`
Sensitivity analysis	`sen`	Generic optimization	`genopt`

In addition, Optimizer creates the scenario `optimum` that stores all experiments that reached the same optimal value. For iterative optimizations, Optimizer also creates the scenario `roi` that retains those solutions that are inside the last-visited region of interest.

Optimizer creates the `genopt_<task>_optimum_family_<family_number>` scenario for generic optimizations. This scenario stores the family with the minimum error (the average error between all the experiments belonging to the family).

Reusing All Simulation Results

Optimizer uses all of the simulation results that already exist in the Sentaurus Workbench project before Optimizer is run.

Most tasks provided by Optimizer require several simulations. To take advantage of simulations previously run (either by the user using Sentaurus Workbench or automatically by Optimizer):

- All previously computed simulation results are loaded before starting to execute any new task.
- All optimization methods start from the best-existing solution.
- The tasks based on RSMs reuse all existing results to build the models, improving the representation of the region of interest.

Advanced Features

Restarting

All iterative optimization methods can store their status after each iteration. If Optimizer is interrupted, the last status can be restored. The following information is stored:

- Iterative heuristic method

  Current best solution, region of interest, number of simulations, number of iterations, number of iterations without improvements, and execution time.

- Quasi-Newton method

  Current best solution, gradient of the current best solution and current Hessian, number of simulations, number of iterations, and execution time.

- Nonlinear simplex method

  Current best solution, simplex composition, number of simulations, number of iterations, and execution time.

Sequencing of Tasks

Optimizer allows the execution of two or more tasks in a sequence and it automatically uses the information computed in previous tasks. For example, this feature can be used to reduce the number of parameters used in an optimization task by performing a screening task before optimization.

The parameter attribute `selValue` allows users to control whether the values computed for a given parameter in previous tasks are used in a different task. This attribute can be used as the nominal value of a sensitivity analysis task or the starting point of an optimization task (see Parameter on page 11).

If this value is set to `autoValue`, its value is taken automatically from previous tasks, using the following criteria:

- If any optimization task has been performed before the task, where `selValue` has been set to `autoValue`, `selValue` is set to the optimal value found for the corresponding parameter in the previous task.
- If a screening task has been performed before the task, where `selValue` has been set to `autoValue`, if the parameter is not considered relevant in the screening task, it is discarded and not used in the current task.

Example: Screening and Iterative Optimization

In a project with five parameters and a given response, the following input file will perform a screening task before an iterative optimization task, so as to discard all irrelevant parameters before starting the optimization process:

```
#
# Define a screening task
Task {
   name = 1
   type = SCREENING
   Parameter {
      { name = A doeArgs = {equidist 1 0 2} }
      { name = B doeArgs = {equidist 1 0 2} }
      { name = C doeArgs = {equidist 1 0 2} }
      { name = D doeArgs = {equidist 1 0 2} }
      { name = E doeArgs = {equidist 1 0 2} }
   }
   Response {
      { name = delta model = standard degree = 1 }
   }
   doe = fullFacNLev
   screenRange = 0.2
```

```
   nextTask = 2
}
#
# Define an iterative optimization task
Task {
   name = 2
   type = ITER_OPTIMIZATION
   Parameter {
      { name = A selValue = autoValue}
      { name = B selValue = autoValue}
      { name = C selValue = autoValue}
      { name = D selValue = autoValue}
      { name = E selValue = autoValue}
   }
   Response {
      { name = delta crit = minimal weight = 1 }
   }
   Stop {
   maxTime = 3600
   }
   nextTask = end
}
#
# Define initial conditions and start first task
#
Start { nextTask = 1 }
```

Task Interdependency

A task can use values obtained as results of other executed tasks. This scheme identifies ‘parents’ and ‘children’ tasks. Data is shared from parents to children tasks using an export-and-import mechanism, which allows for the declaration of which values of one task can be exported and which name is used to import them from another task.

Any numeric attribute of a parameter can be replaced by `import` followed by a dot and a parameter name of the parent task.

```
...
Parameter {
   { name = X1 type = sdoe sdoeArgs = 3
   SDoEModel { type = normal args = {import.mediaX1 import.varX1} } }
   { name = X2 type = sdoe sdoeArgs = 3
   SDoEModel { type = normal args = {import.mediaX2 import.varX2} } }
}
...
```

Another section allows for setting explicitly which responses or attribute responses should be exported to the calling task. This section can be specified using the `Export` block within the task definition:

```
Task {
   ...
   Export {
      { name = mediaRos value = Rosen.media }
      { name = varRos value = Rosen.variance }
      { name = skewRos value = Rosen.skewness }
      { name = kurtRos value = Rosen.kurtosis }
   }
...
```

Exportable Information

Besides the general attributes of each response, additional information – according to the task – can also be exported.

Screening Task

It is possible to export the screening result corresponding to any parameter. Usage possibilities are:

- `value = parname`
- `value = result.parname`

where `parname` is the parameter name.

Sensitivity Analysis Task

It is possible to export the sensibility result for a response with regard to any parameter. For the export, use:

```
value = resname.parname
```

or:

```
value = result.resname.parname
```

where `resname` is the name of the response.

Uncertainty Analysis Task

It is possible to export any of the first four moments of the distribution. To export these values, use:

```
value = resname.media
value = resname.variance
value = resname.skewness
value = resname.kurtosis
```

or:

```
value = result.resname.media
value = result.resname.variance
value = result.resname.skewness
value = result.resname.kurtosis
```

Optimization, Iterative Optimization, and Generic Optimization Tasks

It is possible to export the value of any parameter for the best parameter setting or the value of any response at the optimum found. To export these values, use:

```
value = parname
value = resname
```

or:

```
value = result.parname
value = result.resname
```

Simulation Task

It is possible to export all the results of the simulations (standard behavior). To export these values, use:

```
value = resname
```

or:

```
value = result.resname
```

Convergence Plot

When Optimizer executes some optimization tasks, the goal is achieved by an iterative process that may be relevant for the user to analyze. The iteration evolution is available in the `gopt.log` file and some of the output files are available in the form of alphanumeric information. In addition, a convergence plot file is generated.

You can monitor the progress of the execution of Optimizer by viewing this plot file with Inspect. Using the Inspect auto-reload feature, this plot can be viewed dynamically adding more iterations to the chart, displaying a convergence curve.

The file name of the convergence plot varies depending on the task being executed, for example, an iterative optimization task produces an `itropt.plt` file and a generic optimization task produces a `genopt.plt` file.

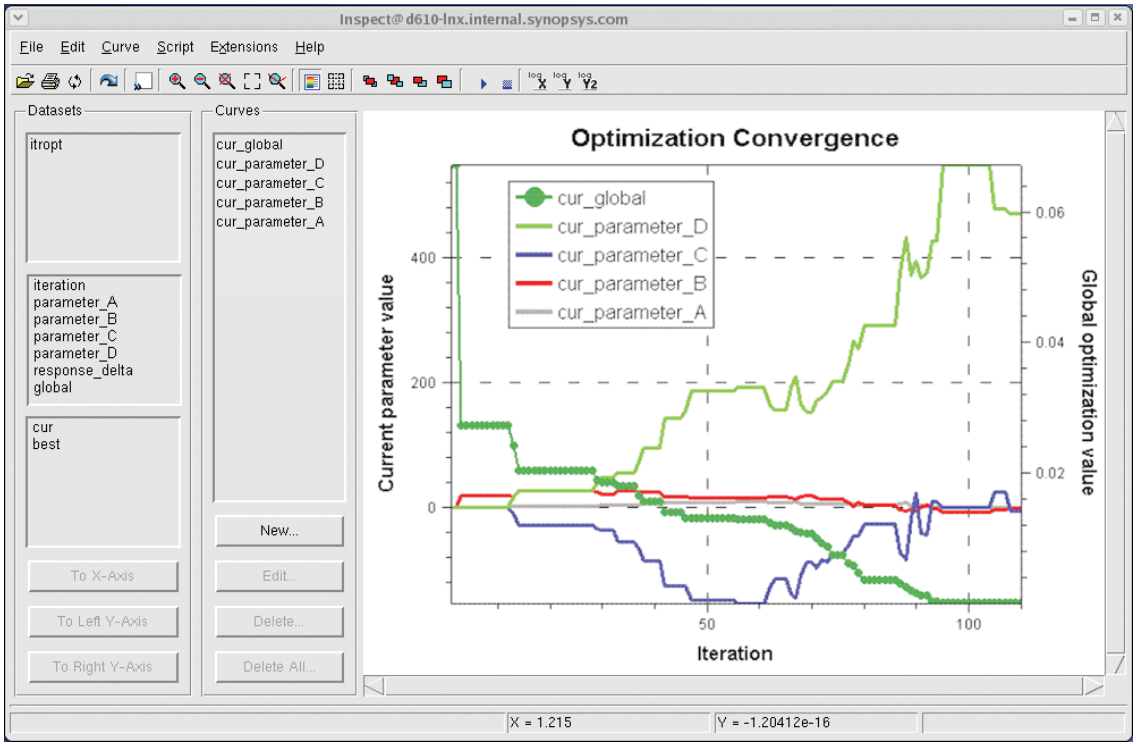

Figure 9 Inspect window showing a convergence plot

To open the convergence plot in Inspect:

1. **File > Load Datasets**.
2. Select the `itropt.plt` or `genopt.plt` file, depending on the type of task executed or being executed.
3. Select the dataset `iteration` for the x-axis and the dataset `global>cur` for the left y-axis.

To update the plot during an Optimizer run:

1. **File > Automatically Update Datasets**.
2. Select the option.
3. Set the time between updates.
4. Click **OK**.

References

[1] D. C. Montgomery, *Design and Analysis of Experiments,* New York: John Wiley & Sons, 1997.

[2] G. E. P. Box and N. R. Draper, *Empirical Model-Building and Response Surfaces*, New York: John Wiley & Sons, 1987.

[3] R. Myers and D. Montgomery, *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, New York: John Wiley & Sons, 1995.

[4] A. I. Khuri and J. A. Cornell, *Response Surfaces: Designs and Analyses*, STATISTICS: Textbooks and Monographs, vol. 81, New York: Marcel Dekker, 1987.

[5] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, New York: John Wiley & Sons, 1993.

[6] R. Fletcher, *Practical Methods of Optimization*, New York: John Wiley & Sons, 1987.

[7] M. A. Tatang *et al*., "An efficient method for parametric uncertainty analysis of numerical geophysical models," *Journal of Geophysical Research - Atmospheres*, vol. 102, no. D18, pp. 21925–21932, 1997.

[8] M. A. Tatang, *Direct Incorporation of Uncertainty in Chemical and Environmental Engineering Systems*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, February 1995.

CHAPTER 3 Reference Guide

This chapter describes all of the commands and file formats, the methodology, and the mathematics pertinent to Optimizer.

Optimizer Commands Reference

Names and Symbols

Every task described here has a two-letter designator as set out in Table 5.

Table 5 Designators

Designator	Definition	Designator	Definition
SC	Screening	UA	Uncertainty analysis
OP	Optimization	DO	Design-of-experiments
IO	Iterative optimization	SD	Stochastic design-of-experiments
GA	Genetic algorithm task	ST	Start block
GO	Generic optimization	CU	Custom task
SA	Sensitivity analysis		

Attributes are either mandatory in the input file specification or optional, depending on which tasks are being referenced. The syntax checker verifies that the attributes used in each block correspond to the ones allowed for such a block. An error message is displayed when a particular mandatory attribute is not defined for a given task. In the following tables, the symbols used to designate each type of attribute are:

- ! – Mandatory attribute
- x – Optional attribute

Global Options

Table 6 lists the global options. For more information about global options, see Global Options on page 9.

Table 6 Global options

Parameter name	Value	Default	SC	OP	IO	GA	GO	SA	UA	DO	SD	ST	CU	Description
magGlbNumEvaluations	<number>			x	x	x	x					x	x	Global maximum number of evaluations
maxGlbTime	<number>			x	x	x	x					x	x	Maximum wallclock time that Optimizer is allowed to run
maxGlbTimeUnit	sec | min | hrs | days	sec		x	x	x	x					x	x	Global time unit
nextTask	<alphanum> | end	end	x	x	x	x	x	x	x	x	x		x	Name of next task
name	<alphanum>		!	!	!	!	!	!	!	!	!		!	Task name, required
type	SCREENING | UNC_ANALYSIS OPTIMIZATION | SDOE | ITER_OPTIMIZATION | GEN_OPTIMIZATION | GENETIC_OPTIMIZATION | SENS_ANALYSIS | DOE		!	!	!	!	!	!	!	!	!		!	Task type, required
exportTable	<filename>		x	x	x	x	x	x	x	x	x		x	Export a table containing all parameter settings and all responses in tab-delimited format

Inner Blocks for Parameters

Table 7 lists the inner blocks for parameters. For more information about parameters, see Parameter on page 11.

Table 7 Inner blocks for parameters

Parameter name	Subname	Value	Default	SC	OP	IO	GA	GO	SA	UA	DO	SD	ST	CU	Description
parameters				!	!	!	!	!	!	!	!	!	x	!	Block start for parameters
name		<alphanum>		!	!	!	!	!	!	!	!	!	x	!	Sentaurus Workbench parameter name
type		ud | doe | sdoe | scen		!	!	!	!	!	!	!	!	!	x	!	Type of parameter
deflection		<number>						x							Amount of step increment
ud parameters only:															
values		<number> | {<number>}		x	x	x	x	x	x	x	x	x	x	x	Values for user-defined parameters
doe parameters only:															
min		<number>		x	x	x	x	x	x		x		x	x	Minimum range value
max		<number>		x	x	x	x	x	x		x		x	x	Maximum range value
scale		linear | logarithmic	linear	x	x	x	x	x	x		x		x	x	Parameter interpolation
doeArgs		<alphanum> | {<alphanum>}		x	x	x	x	x	x		x		x	x	Arguments of DoE that are specific for each parameter
selValue		<number>		x	x	x	x	x	x		x		x	x	Nominal value for sensitivity analysis or starting point for optimization
sdoe parameters only:															
SDoEModel										x		x	x		Block starts for SdoE models
	type	normal | beta | uniform | expon | gamma	normal							x		x	x		Probability density function (PDF)

Table 7 Inner blocks for parameters

Parameter name	Subname	Value	Default	SC	OP	IO	GA	GO	SA	UA	DO	SD	ST	CU	Description
	`args`	`<number>` `<number>`	`0 1`							x		x	x		Probability density function arguments
`sdoeArgs`		`<alphanum>` | `{<alphanum>}`								x		x	x		Arguments of stochastic DoE that are specific for each parameter
scen parameters only:															
`values`		`{ [`*`path-to-example`*`] [`*`scenario-name`*`] }`		x	x	x		x	x	x	x	x	x	x	Scenario reference as source for the parameter values

Inner Blocks for Responses

Table 8 lists the inner blocks for responses. For more information about responses, see Response on page 13.

Table 8 Inner blocks for responses

Parameter name	Value	Default	SC	OP	IO	GA	GO	SA	UA	DO	SD	ST	CU	Description
`responses`			!	!	!	!	!	!	!	!	!	x	!	Block start for responses
`name`	`<alphanum>`		!	!	!	!	!	!	!	!	!	x	!	Sentaurus Workbench variable name
`model`	`standard` | `kriging` | `stochastic`	`standard`	x	x	x	x	x	x	x	x		x	x	RSM type
`transformation`	`exp` | `log` | `sqrt` | `sqr`		x	x	x	x	x	x	x	x	x	x	x	RSM transformation
`degree`	`1` | `2` | `3`	`2`	x	x	x	x	x			x		x	x	RSM degree
`crit`	`minimal` | `maximal` | `closeto`			x	x	x	x					x	x	Optimization criterion
`target`	`<number>`	`" "`		x	x	x	x					x	x	Optimization target when the criterion `closeto` is used, required

Table 8 Inner blocks for responses

Parameter name	Value	Default	SC	OP	IO	GA	GO	SA	UA	DO	SD	ST	CU	Description
lowerBound	<number>	" "			x	x							x	Lower limit of the range valid for optimization termination
upperBound	<number>				x	x							x	Upper limit of the range valid for optimization termination
perc_range	<number>				x	x							x	Percentage relative to the target value used for defining the termination range
weight	<number>	1		x	x	x	x	x				x	x	Relative importance assigned to the response
modelArgs	{<alphanum>}	{}							x			x		Stochastic RSM specification

Inner Blocks for Stopping Criteria

Table 9 lists the inner blocks for stopping criteria. For more information about stopping criteria, see Stopping Criteria on page 50.

Table 9 Inner blocks for stopping criteria

Parameter name	Subname	Value	Default	SC	OP	IO	GA	GO	SA	UA	DO	SD	ST	CU	Description
Stop															Block start for local (single task) stopping criteria
	maxTime	<number> (sec | min | hrs | days)	3600 sec			x									Time
	maxNumEvaluations	<number>	100			x									Number of evaluations
	maxNumIterations	<number>	1			x		x							Number of iterations
	maxWoImprove	<number>	5			x									Iterations without improvements

Table 9 Inner blocks for stopping criteria

Parameter name	Subname	Value	Default	SC	OP	IO	GA	GO	SA	UA	DO	SD	ST	CU	Description
LocalOpt {						x									Block start for local optimum arguments
r2Adj		<number>	0, 99			x									Local optimum: r2Adj
range		<number>	30			x									Local optimum: range
}															
tolerance		<number>	1e-05					x							Termination tolerance

Specific Task Parameters for Screening

Table 10 lists the specific task parameters for screening. For more information about screening, see Screening Task on page 31. For detailed information on screening-specific options, see Command Description on page 31.

Table 10 Specific task parameters for screening

Parameter name	Value	Default	SC	Description
screenCrit	average | local_strong	average	x	Screening criterion
screenRange	<number>	10.0	x	Threshold level for parameter selection
screenBest	<number>	3	x	Number of parameters to be selected

Specific Task Parameters for Iterative Optimization

Table 11 lists the specific task parameters for iterative optimization. For more information about iterative optimization, see Iterative Optimization Task on page 38. For detailed information on iterative optimization–specific options, see Command Description on page 45.

Table 11 Specific task parameters for iterative optimization

Parameter name	Value	Default	IO	Description
dumpFile	<filename>	" "	x	Dump file for iterative optimization
loadFile	<filename>	" "	x	Load file to restart iterative optimization
lowerRoI	<number>	1	x	Granularity parameter: minimum RoI allowed

Specific Task Parameter for Generic Optimization

Table 12 lists the specific task parameter for generic optimization. For more information about generic optimization, see Generic Optimization Task on page 49. For detailed information on generic optimization–specific commands, see Command Description on page 50.

Table 12 Specific task parameter for generic optimization

Parameter name	Value	Default	GO	Description
`solver`	`bcopt | simplex`	`" "`	!	Solver for generic optimization, required

Specific Task Parameters for Genetic Algorithm Optimization

Table 13 lists the specific task parameters for genetic algorithm optimization. For more information about the genetic algorithm, see Genetic Algorithm Task on page 52. For detailed information on genetic algorithm–specific commands, see Command Description on page 54.

Table 13 Specific task parameters for genetic algorithm optimization

Parameter name	Value	Default	GA	Description
`constantK`	`<number>`	`5`	x	Constant of convergence (used by `sigma` and `lineal` fitness functions).
`eliteMutants`	`0|1`	`1`	x	If set to `1`, it generates mutated chromosomes only from elite members. If set to `0`, it generates mutated chromosomes from any chromosome of the population.
`elitism`	`0|1`	`1`	x	Number of chromosomes to be considered elite (`0` for no elitism).
`fitnessFunction`	`simple | sigma | lineal | stair`	`sigma`	x	For a description of `fitnessFunction`, see Optional Attributes on page 54.
`iterationsFile`	`<filename>`	`" "`	!	Required. Valid file name, for example, `iterations.csv`.
`maxFalseChildAttempts`	`<integer>`	`100`	x	The maximum number of attempts to generate a new chromosome (child) by crossover. This condition is used because the algorithm always attempts to generate new chromosomes and, by limiting the number of attempts to produce new children, it is possible to avoid infinite loops.
`maxNumberOfGenerations`	`<integer>`	`15`	x	Maximum number of generations to produce.

Table 13 Specific task parameters for genetic algorithm optimization

Parameter name	Value	Default	GA	Description
nMutants	<number>	20	x	Percentage of the total population to be generated by mutation (not crossover).
percMutation	<number>	10	x	Percentage of mutation of the values for the next generation.
setFitnessBase	0|1	1	x	If set to 1, it normalizes the goal only with the difference between goals (0 is the best). If set to 0, it normalizes the goal values as they are.
velocity	1|2|3|4	1	x	This parameter is used to make different combinations between the fitness function evaluation and the selection crossover method. This parameter controls how quickly (in how many generations) the population converges to an optimum. If set too fast, some potentially good areas (but incidentally, with poor experiments) may be omitted too early. The options are described in Optional Attributes on page 54.

Specific Task Parameters for Sensitivity Analysis

Table 14 lists the specific task parameters for sensitivity analysis. For more information about sensitivity analysis, see Sensitivity Analysis Task on page 57. For detailed information on sensitivity analysis–specific commands, see Command Description on page 58.

Table 14 Specific task parameters for sensitivity analysis

Parameter name	Value	Default	SA	Description
nPoints	<integer>	3	x	Number of points inside sensitivity range
range	<number>	10	x	Percentage range of the whole domain used for sensitivity analysis

Specific Task Parameters for Uncertainty Analysis

Table 15 lists the specific task parameters for uncertainty analysis. For more information about uncertainty analysis, see Uncertainty Analysis Task on page 59. For detailed information on uncertainty analysis–specific commands, see Command Description on page 64.

Table 15 Specific task parameters for uncertainty analysis

Parameter name	Value	Default	UA	Description
exportStat	<filename>	" "	x	Export statistical data to tab-delimited file
sampleSize	<integer>	10000	x	Monte Carlo sample size
uaSeed	<integer>		x	Seed for the generation of random numbers

Specific Task Parameters for Design-of-Experiments

Table 16 lists the specific task parameters for design-of-experiments. For detailed information on design-of-experiments–specific commands, see Design-of-Experiments Task on page 68.

Table 16 Specific task parameters for design-of-experiments

Parameter name	Subname	Value	Default	SC	OP	UA	DO	SD	CU	Description
doe		fullFacNLev | halfFactorialMinus | halfFactorialPlus | fractFactorial2 | plackettBurmann | facedCentralComposite | smallCentralComposite | boxBehnken | taguchi | latinSquare | grecoLatinSquare | diagonalDesign | midPointDesign | randomDesign	" "	!	!		!		x	DoE design
	doeSeed	<integer>		x	x		x			Seed for generation of random numbers used in the DoE

Table 16 Specific task parameters for design-of-experiments

Parameter name	Subname	Value	Default	SC	OP	UA	DO	SD	CU	Description
`tree`		`none | add | {add eval}`	`none`				x	x		Whether the DoE is mapped to the family tree: `add` – Experiments are added to tree. `{add eval}` – Experiments are added to family tree and evaluated.
`Save`						x	x	x		Block start for saving DoE information
	`file`	`<filename>`								File name for saving DoE information
	`elems`	`[values] [experiments] [moments] [corr_matrix]`								DoE information that can be saved

Specific Task Parameters for Stochastic Design-of-Experiments

Table 17 lists the specific task parameters for stochastic design-of-experiments. For detailed information on stochastic design-of-experiments–specific commands, see Stochastic Design-of-Experiments Task on page 69.

Table 17 Specific task parameters for stochastic design-of-experiments

Parameter name	Value	Default	UA	SD	Description
`sdoe`	`probabilisticDesign | montecarloDesign | boundaryDesign | cornerDesign | fromTable`	`" "`	!	!	Stochastic design
`sdoeArgs`	`<number> | { filename separator }`		x	x	Block start for stochastic DoE arguments. This option must be set only for `montecarloDesign` or `fromTable`. In the first case, it specifies the number of split points to be generated for each parameter. In the second case, it specifies the file name of the CSV file to be read and the character that separates the values in it.
`sdoeSeed`	`<integer>`		x	x	Seed for generation of random numbers used in SDoE.

Custom Task Parameters

Table 18 lists the specific parameters for a custom task. For detailed information on custom tasks, see Custom Task on page 71.

Table 18 Specific task parameters for custom tasks

Parameter name	Value	Default	CU	Description
`Algorithm { }`	Valid Tcl script.	`" "`	!	It is a block enclosed by braces and contains a Tcl script that sequences the instructions that perform the intended optimization process. Optimizer does not validate this script.

Output Files

Optimizer generates several different files when running a task. Each file includes specific output information that may be useful to the user. Part of this output information is also sent to the standard output.

The notation used for the file names depends on the names given to the parameters and responses, which are enclosed in a pair of angle brackets. For example, assume that a simulation yields a `<response_name>_anova.dat` file. If there is a response named `delta1`, there will be a `delta1_anova.dat` file after running Optimizer for that particular task. The output files for various tasks are described.

Uncertainty Analysis Task

For each response, the following files are generated (see Output on page 66), containing data in a tab-delimited format (TDF) plain text format, which can be imported to other tools such as Microsoft® Excel:

`<response_name>_coeff.dat`	Model coefficients for the response.
`<response_name>_stat.dat`	Model statistics for the response.
`<response_name>_var.dat`	Model variance for the response.
`<response_name>_anova.dat`	ANOVA table for the response.
`<response_name>_histo.dat`	Model histogram for the response.

`<response_name>_mom.dat`	Model moments for the response.
`values.dat`	Details of all the values generated to perform the experiment.
`uaTable.tab`	Structurally, it is equal to the `unc_<n>.tab` file showing values for all scenarios.
`unc_<n>.tab`	This file holds the values of the experiments inside the whole domain where `n` stands for the evaluated scenario. Its columns show all parameters, all responses, a node number from the simulation tree and, finally, a goal column.
`uaStTable.tab`	This contains the same data as in the `.dat` files previously described, that is, correlatively and for each response: the model coefficients, model statistics, model variance, ANOVA table, model histogram, and model moment files. This information is followed by a list of values generated to run the experiment, as stored in the values `.dat` file.

Sensitivity Analysis Task

`sen_<n>.tab`	Values of the experiments inside the whole domain where `n` stands for the evaluated scenario. Its columns show all parameters, all responses, a node number from the simulation tree and, finally, a goal column.

Screening Task

`screen_<n>.tab`	Values of the experiments inside the whole domain where `n` stands for the evaluated scenario. Its columns show all parameters, all responses, a node number from the simulation tree and, finally, a goal column.

Generic Optimization Task

`genopt_<n>.tab` — Optimum parameter and response values obtained with the optimization process where `n` is the evaluated scenario. Its columns show all parameters, all responses, a node number from the simulation tree and, finally, a goal column.

`genopt.plt` — This contains the data of a convergence plot for a simulation. For tools such as Inspect, iteration is put on the x-axis and the behavior of any particular parameter (minimum, current, best, or maximum) or response (current or best, for any response or a global one) is put on the y-axis.

Optimization Task

`opt_<n>.tab` — Values of the experiments inside the whole domain where `n` stands for the evaluated scenario. Its columns show all parameters, all responses, a node number from the simulation tree and, finally, a goal column.

Iterative Optimization Task

`itropt_<n>.tab` — Optimum parameter and response values obtained with the optimization process where `n` stands for the evaluated scenario. Its columns show all parameters, all responses, a node number from the simulation tree and, finally, a goal column.

`itropt.plt` — This contains the data of a convergence plot for a simulation. For tools such as Inspect, iteration is put on the x-axis and the behavior of any particular parameter (minimum, current, best, or maximum) or response (current or best, for any response or a global one) is put on the y-axis.

Design-of-Experiments Task

`doe_<n>.tab` — Values of the experiments inside the whole domain where `n` stands for the evaluated scenario. Its columns show all parameters, all responses, a node number from the simulation tree and, finally, a goal column.

Stochastic Design-of-Experiments Task

`sdoe_<n>.tab` — Values of the experiments inside the whole domain where `n` stands for the evaluated scenario. Its columns show all parameters, all responses, a node number from the simulation tree and, finally, a goal column.

Mathematical Expressions

Gradient Vector

The gradient vector $\nabla f(\vec{x})$ of a scalar function $f(\vec{x})$ ($\vec{x} \in R^n$) is defined by the $n \times 1$ vector that is built from the first partial derivatives:

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(\vec{x}) \\ \dots \\ \frac{\partial}{\partial x_n} f(\vec{x}) \end{bmatrix} \tag{16}$$

Hessian Matrix

The Hessian matrix $\nabla^2 f(\vec{x})$ of a scalar function $f(\vec{x})$ $(\vec{x} \in R^n)$ is defined by the $n \times n$ matrix that is built out of the second partial derivatives:

$$\nabla^2 f(\vec{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} f(\vec{x}) & \cdots & \frac{\partial^2}{\partial x_1 x_n} f(\vec{x}) \\ \cdots & \cdots & \cdots \\ \frac{\partial^2}{\partial x_n x_1} f(\vec{x}) & \cdots & \frac{\partial^2}{\partial x_n^2} f(\vec{x}) \end{bmatrix} \tag{17}$$

Equations for Response Surface Models

For all of the equations here, the notation is:

- n : Number of experiments
- h : Number of unknown coefficients (or terms) in the model
- Y_i : Actual value of the experiment i
- $\hat{Y}_i$: Estimated value of the experiment i
- $\bar{Y} = \frac{\sum_{i=1}^{n} Y_i}{n}$: Actual mean value

Model Accuracy

Three statistics that measure the model accuracy are provided:

- $\boldsymbol{R}^2$: The coefficient of determination is a statistic that measures the predictive capacity of the model (see Model Information on page 27). The relevant equations are:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n} (Y_i - \bar{Y})^2} \tag{18}$$

$$R^2 = \frac{\sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2}{\sum_{i=1}^{n} (Y_i - \bar{Y})^2} \tag{19}$$

- $\boldsymbol{R}^2_{adj}$: Model Information on page 27 discusses the adjusted $\boldsymbol{R}^2$ statistic ($\boldsymbol{R}^2_{adj}$). The relevant equation is:

$$R^2\text{-adj} = 1 - \left(\frac{n-1}{n-h}\right)(1 - R^2) \tag{20}$$

- $\boldsymbol{S}^2$: The estimated variance of the error is a measure of the model variability. It is discussed in Model Information on page 27. The relevant equation is:

$$S^2 = \frac{\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2}{n-h} \tag{21}$$

ANOVA Table

The analysis of variance (ANOVA) consists of calculations that provide information about levels of variability within a regression model and form a basis for tests of significance. The basic regression concept, *DATA* = *FIT* + *RESIDUAL*, can be rewritten as:

$$(Y_i - \bar{Y}) = (\hat{Y}_i - \bar{Y}) + (Y_i - \hat{Y}_i) \tag{22}$$

The first term is the total variation in the simulation response Y, the second term is the variation of the mean result, and the third term is the residual value. Squaring each of these terms and adding over all of the n observations gives:

$$\sum_{i=1}^{n} (Y_i - \bar{Y})^2 = \sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2 + \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \tag{23}$$

This equation can also be written as $SST = SSM + SSE$, where SS is notation for sum of squares, and T, M, and E are notations for total, model, and error, respectively.

The coefficient of determination ($\boldsymbol{R}^2$) is equal to the ratio of the model sum of squares to the total sum of squares:

$$R^2 = \frac{SSM}{SST} = \frac{\sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2}{\sum_{i=1}^{n} (Y_i - \bar{Y})^2} \tag{24}$$

This formalizes the interpretation of $\boldsymbol{R}^2$ as explaining the fraction of variability in the data described by the RSM.

For an RSM with h terms, the model degrees of freedom (*DFM*) are equal to $h - 1$; the error degrees of freedom (*DFE*) are equal to $n - h$; and the total degrees of freedom (*DFT*) are equal to $n - 1$, which is the sum of *DFM* and *DFE*.

The sample variance (S_Y^2 or *MST*) is equal to:

$${S_Y}^2 = MST = \frac{SST}{DFT} = \frac{\sum_{i=1}^{n} (Y_i - \bar{Y})^2}{n-1} \tag{25}$$

The mean square model (*MSM*) is equal to:

$$MSM = \frac{SSM}{DFM} = \frac{\sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2}{h-1} \tag{26}$$

The mean square error (*MSE*), which is an estimate of the variance with respect to the population regression line, is equal to:

$$MSE = \frac{SSE}{DFE} = \frac{\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2}{n-h} \tag{27}$$

Table 19 displays analysis of variance calculations.

Table 19 ANOVA table definitions

Source	Degrees of freedom	Sum of squares	Mean square	*F*
Model	$h-1$	SSM	SSM/DFM	MSM/MSE
Error	$n-h$	SSE	SSE/DFE	
Total	$n-1$	SST	SST/DFT	

The F column provides a statistic for testing the hypothesis that some coefficients are not null against the null hypothesis, which is that all the coefficients of the model equal zero. The test statistic is the ratio *MSM/MSE*, the mean square model divided by the mean square error. When *MSM* is large relative to *MSE*, the ratio is large and there is evidence against the null hypothesis. The F test does not indicate which coefficients are not equal to zero. It only indicates that at least one of them is linearly related to the response variable.

The test statistic *MSM/MSE* has a probability density function $F(p, n-p-1)$.

Moments

A real-valued random variable $x(\omega)$ is a function that maps the probability space Ω into the real line, such that [1]:

1. The set $\{\omega \in \Omega | x(\omega) \leq X\}$ is an event for any real number X.
2. The probabilities of the events $\{x(\omega) = -\infty\}$ and $\{x(\omega) = \infty\}$ equal zero.

$$Pr(x(\omega) = -\infty) = 0 \tag{28}$$

$$Pr(x(\omega) = \infty) = 0 \tag{29}$$

$$F_x(x) = Pr(x(\omega)| \leq x) = \int_{-\infty}^{x} f_x(u)du \qquad \forall x \in \Re \tag{30}$$

$$Pr(x(\omega)|\omega \in A) = \int_A f_x(x)dx \tag{31}$$

The i-th moment of a continuous real-valued random variable $x(\omega)$, with probability density function $f_x(x)$, is defined by:

$$m_i(x(\omega)) = \int_{-\infty}^{\infty} x^i f_x(x)dx \tag{32}$$

Moments define some properties of the random variable (see ‘Mean’ in this section). Similar to the concept of ‘moment’ in mechanic engineering, it is also possible to have moments of a random variable based on a nonzero point of reference:

$$m_i(x-p) = \int_{-\infty}^{\infty} (x-p)^i f_X(x)dx \tag{33}$$

where p is a nonzero real number. If the point p is the mean value, there are central moments:

$$c_i(x) = m_i(x-\mu(x)) \tag{34}$$

Several measures of a random variable that are based on its second, third, and fourth central moments are known to be important (see ‘Variance,’ ‘Skewness,’ and ‘Kurtosis’ in this section).

- Mean

  The first moment $m_1(x)$, which is also known as the mean or expected value $\mu(x)$, defines the central mass of the probability density function or the average of the random variable.

- Variance

  The second central moment $c_2(x)$ defines the degree of dispersion of the corresponding random variable from its mean value. It is also called variance, denoted by $\sigma^2(x)$. Its square root is known as the standard deviation.

- Skewness

  The third central moment affects the degree of asymmetry of the corresponding probability density function around its mean value. The skewness factor is a coefficient used to measure this degree of asymmetry. It is defined by:

  $$\gamma_1(x) = \frac{c_3(x)}{\sigma^3(x)} \tag{35}$$

  A positive value of the skewness factor relates to a probability density function with a greater tail area towards more positive values. A negative value indicates a probability density function with a greater tail area towards more negative values. A zero value indicates a symmetric probability density function.

- Kurtosis

  The kurtosis factor measures the relative peakedness or flatness of a probability density function with respect to the Gaussian probability density function. It is derived from the fourth central moment:

$$\gamma_2(x) = \frac{c_4(x)}{\sigma^4(x)} \tag{36}$$

  The kurtosis factor for a Gaussian probability density function is equal to three. Therefore, a kurtosis factor that is less than three defines a less peaked or flatter probability density function than the Gaussian. A kurtosis factor that is more than three means a more peaked or less flat probability density function than the Gaussian. Platykurtic and leptokurtic are synonyms for probability density functions with less than three and more than three kurtosis factors, respectively.

Optimization Problem

An optimization problem [2] can be expressed as a maximization or minimization task, for example, maximizing profits or minimizing costs. The objective function $f(\vec{x})$ reflects whether a particular set of input parameters (a vector $\vec{x}_i$) produces a good or bad result for the analyzed model function.

The optimization problem is usually formulated as a minimization of the objective function $f(\vec{x})$ that depends on $\vec{x}$:

$$\min_{\vec{x} \in R^n} f(\vec{x}) \tag{37}$$

$f(\vec{x})$ is also called the target function or cost function. If the dimension of $\vec{x}$ is one (it is a scalar), it is a univariate optimization problem. Otherwise, it is a multivariate optimization problem.

When no additional conditions are supplied for the input parameter vector, it is an unconstrained optimization problem, where any value of $\vec{x} \in R^n$ is a feasible point.

In constrained optimization problems, equality or inequality constraints reduce the input parameter space. These can be expressed as:

$$\begin{aligned} &\min_{\vec{x} \in R^n} f(\vec{x}) \\ &h_i(\vec{x}) = 0 \qquad i = 1, \ldots, m \\ &g_j(\vec{x}) \leq 0 \qquad j = 1, \ldots, p \end{aligned} \tag{38}$$

where m is the number of equality constraints and p is the number of inequality constraints.

A particular constrained problem is bound-constrained, which can be expressed as:

$$\begin{aligned} &\min_{\vec{x} \in R^n} f(\vec{x}) \\ &\vec{l} \leq \vec{x} \leq \vec{u} \end{aligned} \tag{39}$$

where $\vec{l}$ and $\vec{u}$ are the respective lower and upper bounds for the vector $\vec{x}$.

An extreme is a global optimum that is definitely the highest or lowest function value, as opposed to a local optimum, which is the highest or lowest function value within a finite neighborhood.

Optimizer solves a bound-constrained problem, determines a parameter setting that optimizes a weighted function of the simulation responses, and satisfies bound constraints to the parameter domain.

Optimization

Optimization is the search for an optimal value of the objective function within given ranges of the input variables or parameters.

There are several methods for solving optimization problems. In general, the method used depends on the behavior of the target function. The range of available methods spans from iterative methods (based on the derivatives of the target function) to local search heuristics (for example, genetic algorithms or random search algorithms).

The unconstrained optimization problem is central to the development of optimization software. Constrained optimization algorithms are often extensions of unconstrained algorithms, while nonlinear least square and nonlinear equation algorithms tend to be specializations.

Global optimization algorithms try to find a vector $\vec{x}_*$ that minimizes $f(\vec{x})$ over all possible vectors $\vec{x}$. This is a more difficult problem to solve. There is no efficient algorithm to perform this task. For many applications, local minima are adequate, particularly when users utilize their own experiences to provide a good starting point for the algorithm.

The Newton method generates a varied and important class of algorithms that require the computation of the gradient vector and Hessian matrix (see Mathematical Expressions on page 94).

Although the computation or approximation of the Hessian matrix is time consuming, it is invaluable for many problems. The following sections describe algorithms in which users explicitly supply the Hessian before discussing algorithms that do not require the Hessian.

Gradient-Based Optimization Methods

The majority of optimization algorithms are based on methods that use the gradient and higher derivatives of the target function. These methods approximate the target function $f(\vec{x})$ by a Taylor series around $\vec{x}_0$:

$$f(\vec{x}_0 + \vec{p}) \approx f(\vec{x}_0) + (\nabla f(\vec{x}_0))^T \cdot \vec{p} + \frac{1}{2} \cdot \vec{p}^T \cdot \nabla^2 f(\vec{x}_0) \cdot \vec{p} \tag{40}$$

Step Direction

Many optimization algorithms iteratively improve the solution. Each iteration consists of four basic steps:

1. If the convergence conditions are satisfied, stop the algorithm with the actual point $\vec{x}_k$ as the solution.
2. Compute a direction $\vec{p}_k$.
3. Compute a step length α_k.
4. Set the new actual point $\vec{x}_{k+1} = \vec{x}_k + \alpha_k \cdot \vec{p}_k$ and return to Step 1.

For multidimensional optimization problems, selecting the step direction for the next iteration in order to decrease the scalar function $f(\vec{x})$ is a major consideration. Usually, the directions used for the next iteration are the steepest descent direction, the Newton direction, or a combination of both.

Steepest Descent Direction

The modeling of the target function is performed by a linear approximation of $f(\vec{x}_0)$ based on a Taylor series expansion of first order:

$$f(\vec{x}_0 + \vec{p}) \approx f(\vec{x}_0) + (\nabla f(\vec{x}_0))^T \cdot \vec{p} \tag{41}$$

where $\vec{p}$ denotes a small step away from the current point $\vec{x}$. A reduction of the function $f(\vec{x})$ is achieved by a step $\vec{p}$ where $(\nabla f(\vec{x}_0))^T \cdot \vec{p}$ is a negative number with a large absolute value. The direction is found by solving the following minimization problem:

$$\min_{\vec{p} \in R^n} \frac{(\nabla f(\vec{x}_0))^T \cdot \vec{p}}{\|\vec{p}\|_2} \tag{42}$$

that gives direction $\vec{p}$:

$$\vec{p} = -\nabla f(\vec{x}_0) \tag{43}$$

along the negative gradient of the target function $f(\vec{x})$.

Newton Direction

For the Newton direction, a quadratic approximation of the target function is used. For existing second derivatives of $f(\vec{x})$, the minimum of the target function is found by:

$$\nabla f(\vec{x}_0) + \nabla^2 f(\vec{x}_0) \cdot \vec{p} = 0 \tag{44}$$

Therefore, the Newton search direction is given by:

$$\nabla^2 f(\vec{x}_0) \cdot \vec{p} = -\nabla f(\vec{x}_0) \tag{45}$$

using the gradient and Hessian of the target function.

Step-Length Method

Step-length or line-search methods are iterative methods, where the parameter vector of the next iteration is calculated by:

$$\vec{x}_{k+1} = \vec{x}_k + \alpha_k \cdot \vec{p}_k \tag{46}$$

where $\vec{x}_k$ is the parameter vector of the current iteration, $\vec{p}_k$ is the step direction, and α_k is the step length.

The step direction is calculated by the steepest descent direction or Newton direction previously described. Finding the step length is a univariate minimization problem:

$$\min_{\alpha_k} \; f(\vec{x}_k + \alpha_k \cdot \vec{p}_k) \tag{47}$$

Eq. 13 is used to find the minimum of the objective function along the search direction. For this type of optimization method, the selection of the direction does not depend on the step length.

Trust-Region Method

Trust-region methods approach the selection of the next iteration step differently than the step length–based methods. The next point is calculated by first selecting some tentative step length, and then applying the quadratic model to determine a direction and an actual step.

The trust region step is calculated by:

$$(\nabla^2 f(\vec{x}) + \lambda I) \cdot \vec{p} + \nabla f(\vec{x}) \; = \; 0 \tag{48}$$

where λ is a nonnegative scalar and I is the identity matrix.

The value of λ and, consequently, the size of the trust region are adapted by the quality of the previous evaluations.

Having solved the trust region problem, the next stage is to decide whether to accept the step or change the trust region radius and calculate a new solution.

In general, an improvement of the target function is reflected in a decrease of λ and vice versa. Therefore, if the new step does not reduce the value of the target function, the value of λ is modified and a new step is calculated from Eq. 48.

For $\lambda = 0$, the solution of Eq. 48 is simply the Newton direction and, for $\lambda \to \infty$, the step $\vec{p}$ becomes parallel to the steepest descent direction.

The Levenberg–Marquardt algorithm is a specialized trust-region method that is used to solve nonlinear least square problems by exploiting the specific structure of these particular problems.

Comparison

Steepest Descent and Newton Directions

These two strategies for the search direction, steepest descent direction and Newton direction, have different convergence properties and are chosen with regard to the target function.

If the quadratic model of the target function is accurate, the Newton direction converges quadratically to the optimum value. If the Hessian matrix is positive definite, the start value is sufficiently close to the optimum, and the step length converges to 1.

Using the steepest descent algorithm, the step of the next iteration is always a descent direction for a nonvanishing gradient. In practice, the steepest descent typically requires a large number of iterations to make progress toward the solution.

Trust-region methods, however, enable a smooth transition from the steepest descent direction to the Newton direction in a way that gives the global convergence properties of steepest descent and the fast local convergence of the Newton direction. The Levenberg–Marquardt algorithm uses these properties to solve least square problems.

Step-Length and Trust-Region Methods

For both step-length and trust-region algorithms, the next iteration point is selected according to a scalar value. In step-length algorithms, this scalar is the step length α_k; in trust-region methods, this scalar is the size of the trust region. The major difference is how the second-order information influences the search direction. Step-length algorithms leave the Hessian unchanged; trust-region methods use a so-called modified Hessian.

Derivative Approximations and Optimization Methods

The previous algorithms require the derivatives of the model function $f(\vec{x})$. If this model function is given by an analytic formula, the first and second derivatives can be calculated directly. For a model function formed by results of several simulations, it is impossible to find analytic expressions for the derivatives. For this reason, approximation methods must be used.

So far, the availability of the Hessian matrix has been assumed, but the algorithms are unchanged if the Hessian matrix is replaced by a reasonable approximation.

Two different methods use approximate Hessians in place of the real one:

- The first possibility is to use difference approximations to the exact Hessian by exploiting the fact that each column of the Hessian can be approximated by the difference between two instances of the gradient vector evaluated at two nearby points. For sparse Hessians, many columns of the Hessian can often be approximated with a single gradient evaluation by choosing the evaluation points judiciously.
- Quasi-Newton methods build up an approximation to the Hessian by recording the gradient differences along each step taken by the algorithm. Various conditions are imposed on the approximate Hessian. For example, its behavior along a step just taken is forced to mimic the behavior of the exact Hessian and it is usually kept positive definite.

Finite-Difference Approximations

The most common method for obtaining an approximation of the Hessian matrix is to use differences of gradient values.

Hessian Approximation

If forward differences are used, the i-th column of the Hessian matrix is replaced by:

$$\frac{\nabla f(\vec{x} + h_i \vec{e}_i) - \nabla f(\vec{x})}{h_i} \tag{49}$$

for some suitable choice of the difference parameter h_i. Here, $\vec{e}_i$ is the vector with 1 in the i-th position and zeros elsewhere. Similarly, if backward differences are used, the i-th column of the Hessian matrix is replaced by:

$$\frac{\nabla f(\vec{x}) - \nabla f(\vec{x} - h_i \vec{e}_i)}{h_i} \tag{50}$$

If central differences are used, the i-th column is replaced by:

$$\frac{\nabla f(\vec{x} + h_i \varepsilon_i) - \nabla f(\vec{x} - h_i \varepsilon_i)}{2h_i} \tag{51}$$

An appropriate choice of the difference parameter h_i is difficult. Rounding errors hide the calculation if h_i is too small, while truncation errors dominate if h_i is too large. Newton codes rely on forward differences because they often yield sufficient accuracy for reasonable values of h_i. Central differences are more accurate but they require twice the work ($2 \cdot n$ gradient evaluations against n evaluations).

Variants of the Newton method for problems with a large number of variables cannot use the above techniques to approximate the Hessian matrix because the cost of n gradient evaluations is prohibitive. For problems with a sparse Hessian matrix, it is possible to use specialized techniques based on graph coloring that allow difference approximations to the Hessian matrix to be computed efficiently. For example, if the Hessian matrix has the bandwidth $2b + 1$, only $b + 1$ gradient evaluations are required.

Gradient Approximation

For the approximation of the first derivatives, finite-difference approximations can also be used. Finite differences are calculated from a number of function evaluations of particular $\vec{x}$ values.

If forward differences are used, the i-th element of the gradient vector $\nabla f(\vec{x})$ is replaced by:

$$\frac{f(\vec{x} + h_i \vec{e}_i) - f(\vec{x})}{h_i} \tag{52}$$

Similarly, the backward-difference approximation of the i-th element of the gradient vector $\nabla f(\vec{x})$ can be calculated by:

$$\frac{f(\vec{x}) - f(\vec{x} - h_i \vec{e}_i)}{h_i} \tag{53}$$

For higher accuracy, the central-difference approximation is used:

$$\frac{f(\vec{x} + h_i \vec{e}_i) - f(\vec{x} - h_i \vec{e}_i)}{2h_i} \tag{54}$$

When the target function has n independent parameters, $2 \cdot n$ function evaluations are required using a central-difference approximation. If a single evaluation takes from several minutes to several hours, the forward-difference formula is preferable to reduce the overall calculation time. To keep the 'approximation error' in an acceptable range, the step size is adapted for each input parameter, depending on the values of the previous iteration.

Quasi-Newton Methods

Quasi-Newton or variable metric methods can be used when the Hessian matrix is difficult or time-consuming to evaluate. The use of a finite-difference approximation of the first derivatives is not suitable for this purpose. This would require a large amount of function evaluations.

Instead of obtaining an estimate of the Hessian matrix at a single point, these methods gradually build up an approximate Hessian matrix by using gradient information from some or all of the previous iterates $\vec{x}_k$ visited by the algorithm.

A good approximation of the curvature of the nonlinear function, without additional function evaluations, can be computed by an iterative scheme, where the gradient of the current and last step, and the Hessian of the last step are used.

In the Broyden–Fletcher–Goldfarb–Shanno (BFGS) update, the approximative Hessian matrix β_{k+1} (the index of the matrix β_k is the approximative Hessian matrix in the iteration k) is calculated from the approximation of the last step β_k with:

$$\beta_{k+1} = \beta_k - \frac{\beta_k \cdot \vec{s}_k \cdot (\beta_k \cdot \vec{s}_k)^T}{\vec{s}_k^T \cdot \beta_k \cdot \vec{s}_k} + \frac{\vec{y}_k \cdot \vec{y}_k^T}{\vec{y}_k^T \cdot \vec{s}_k} \tag{55}$$

where the vectors $\vec{s}_k = \vec{x}_{k-1} - \vec{x}_k$ denote the last step and $\vec{y}_k = \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k)$ denote the difference of the gradient vectors. The search direction $\vec{p}_k$ is assumed to be the Newton direction.

The algorithm starts with the identity matrix as an initial value of the approximated Hessian matrix. Therefore, the first iteration of the Newton step is equivalent to an iteration of a steepest descent method.

Although, there are other iterative update formulas, for example, the Powell–symmetric–Broyden (PSB) and Davidon–Fletcher–Powell (DFP) updates, it is generally agreed that the most effective update is the BFGS update.

The availability of quasi-Newton methods renders steepest descent methods obsolete. Both types of algorithm require only first derivatives; both require a line search. The quasi-Newton algorithms require slightly more operations to calculate an iterate and more storage. However, in almost all cases, the advantage of superior convergence outweighs these additional costs.

Nongradient-Based Methods

There are two other approaches for unconstrained problems that are not so closely related to the Newton method:

- Nonlinear conjugate gradient methods

  Nonlinear conjugate gradient methods are motivated by the success of the linear conjugate gradient method in minimizing quadratic functions with positive definite Hessians. They use search directions that combine the negative gradient direction with another direction, which is chosen so that the search occurs along a direction not previously explored by the algorithm. At least, this property holds for the quadratic case, for which the minimizer is found exactly within n iterations. For nonlinear problems, performance is problematic, but these methods have the advantage that they require only gradient evaluations and do not use much storage.

- Nonlinear simplex method

  The nonlinear simplex method (not to be confused with the simplex method for linear programming) does not require gradient or Hessian evaluations. It performs a pattern search based only on function values. As it makes little use of information about $f(\vec{x})$, it typically requires many iterations to find a reasonable solution. It is useful when $f(\vec{x})$ is nonsmooth or contains noise, or when the gradient of $f(\vec{x})$ is hard or impossible to calculate. For an n-dimensional problem, this method maintains a simplex of $n + 1$ points (a triangle in two dimensions or a pyramid in three dimensions). The simplex moves, expands, contracts, and distorts its shape as it attempts to find a minimizer. This method is slow and can be applied only to problems in which n is small.

Bound-Constrained Optimization Methods

Newton-based methods for bound-constrained optimization use step-length and trust-region versions of unconstrained minimization algorithms. This discussion emphasizes the differences between the unconstrained and bound-constrained cases.

A step-length method for bound-constrained problems generates a sequence of iterates by setting:

$$\vec{x}_{k+1} = \vec{x}_k + \alpha_k \cdot \vec{p}_k \tag{56}$$

where $\vec{x}_k$ is a feasible approximation to the solution, $\vec{p}_k$ is a search direction, and $\alpha_k > 0$ is the step length.

The direction $\vec{p}_k$ is obtained as an approximate minimizer of the subproblem:

$$\min_{\vec{p} \in R^n} \left\{ \nabla f(\vec{x})^T \vec{p} + \frac{1}{2} \cdot \vec{p}^T \cdot \beta_k \cdot \vec{p} : p_i = 0, i \in W_k \right\} \tag{57}$$

where W_k is the working set and β_k is an approximation to the Hessian matrix $\nabla^2 f(\vec{x}_k)$ at $\vec{x}_k$. All variables in the working set W_k are fixed during this iteration; all other variables are in the free set F_k. This subproblem is expressed in terms of the free variables by noting that it is equivalent to the unconstrained problem:

$$\min_{\vec{p} \in R^n} \left\{ g_k{}^T \vec{w} + \frac{1}{2} \cdot \vec{w}^T \cdot A_k \cdot \vec{w} : \vec{w} \in R^{m_k} \right\} \tag{58}$$

where m_k is the number of free variables, A_k is the matrix obtained from β_k by taking the rows and columns whose indices correspond to the free variables, and g_k is obtained from $\nabla f(\vec{x}_k)$ by taking the components whose indices correspond to the free variables.

The main requirement of W_k is that $\vec{p}_k$ be a feasible direction, that is, $\vec{x}_k + \alpha \cdot \vec{p}_k$ satisfies the constraints for all $\alpha > 0$ sufficiently small. This is certainly the case if $W_k = A(\vec{x}_k)$, where:

$$A(\vec{x}) = \{ i : x_i = l_i \} \cup \{ i : x_i = u_i \} \tag{59}$$

is the set of active constraints at $\vec{x}$. As long as progress is made with the current W_k, the next working set W_{k+1} is obtained by merging $A(\vec{x}_{k+1})$ with W_k. This updating process is continued until the function cannot be reduced further with the current working set.

At this point, the classical strategy is to drop a constraint in W_k for which $\frac{\partial}{\partial x_i} f(\hat{x}_k)$ has the wrong sign, that is, $i \in W_k$ but $i \in W_k$, where the binding set:

$$B(\hat{x}) = \left\{ i : x_i = l_i, \frac{\partial}{\partial x_i} f(\hat{x}_k) \geq 0 \right\} \cup \left\{ i : x_i = u_i, \frac{\partial}{\partial x_i} f(\hat{x}_k) \leq 0 \right\} \tag{60}$$

is defined as before. In general, it is advantageous to drop more than one constraint, in the anticipation that the algorithm makes more rapid progress toward the optimal binding set. However, all dropping strategies are constrained by the requirement that the solution $\vec{p}_k$ of the subproblem be a feasible direction.

An implementation of a step-length method based on subproblem Eq. 57 must cater to the situation in which the reduced Hessian matrix A_k is indefinite, because in this case the subproblem does not have a solution. This situation can arise, for example, if β_k is the Hessian matrix or an approximation obtained by differences of the gradient. Here, it is necessary to specify $\vec{p}_k$ by other means. For example, the modified Cholesky factorization can be used.

Quasi-Newton methods for bound-constrained problems update an approximation to the reduced Hessian matrix since, as already noted, only the reduced Hessian matrix is likely to be positive definite. The updating process is not entirely satisfactory because there are situations in which a positive definite update that satisfies the quasi-Newton condition does not exist. Moreover, complications arise because the dimension of the reduced matrix changes when the working set W_k changes. Quasi-Newton methods are usually beneficial when the working set remains fixed during consecutive iterations.

The choice of the step-length parameter α_k is similar to the unconstrained case. If subproblem Eq. 57 has a solution $\vec{p}_k$ and $\hat{x}_k + \vec{p}_k$ violates one of the constraints, the largest $\mu_k \in (0, 1)$ is computed, such that:

$$\hat{x}_k + \mu_k \cdot \vec{p}_k \tag{61}$$

is feasible. A standard strategy for choosing α_k is to seek an $\alpha_k \in (0, \mu_k]$ that satisfies the sufficient decrease and curvature conditions.

The existence of such an α_k is guaranteed, unless μ_k satisfies the sufficient decrease condition, and:

$$\nabla f(\hat{x}_k + \mu_k \cdot \vec{p}_k)^T \cdot \vec{p}_k < 0 \tag{62}$$

This situation is likely to happen if, for example, $f(\hat{x})$ is strictly decreasing on the line segment $[\hat{x}_k, \hat{x}_k + \mu_k \cdot \vec{p}_k]$. In this case, $\alpha_k = \mu_k$ can be set.

References

[1] M. H. DeGroot, *Probability and Statistics*, Massachusetts: Addison-Wesley, 2nd ed., 1986.

[2] J. J. Moré and S. J. Wright, *Optimization Software Guide*, Frontiers in Applied Mathematics, vol. 14, Philadelphia: SIAM, 1993.