

---

# **SimOpt**

***Release 1.0***

**simopt-admin**

**Dec 06, 2022**



# CONTENTS

<b>1</b>	<b>Getting Started</b>	<b>3</b>
<b>2</b>	<b>Contents</b>	<b>5</b>
2.1	stable . . . . .	5
2.1.1	simopt package . . . . .	5
2.1.1.1	Subpackages . . . . .	5
2.1.1.2	Submodules . . . . .	135
2.1.1.3	simopt.GUI module . . . . .	135
2.1.1.4	simopt.base module . . . . .	140
2.1.1.5	simopt.data_farming_base module . . . . .	152
2.1.1.6	simopt.directory module . . . . .	156
2.1.1.7	simopt.experiment_base module . . . . .	156
2.1.1.8	Module contents . . . . .	175
<b>3</b>	<b>Acknowledgments</b>	<b>177</b>
	<b>Python Module Index</b>	<b>179</b>
	<b>Index</b>	<b>181</b>



The purpose of the SimOpt testbed is to encourage development and constructive comparison of simulation-optimization (SO) solvers (algorithms). We are particularly interested in the finite-time performance of solvers, rather than the asymptotic results that one often finds in related literature.

For the purposes of this site, we define simulation as a very general technique for estimating statistical measures of complex systems. A system is modeled as if the probability distributions of the underlying random variables were known. Realizations of these random variables are then drawn randomly from these distributions. Each replication gives one observation of the system response, i.e., an evaluation of the objective function. By simulating a system in this fashion for multiple replications and aggregating the responses, one can compute statistics and use them for evaluation and design.

The paper [Pasupathy and Henderson \(2006\)](#) explains the original motivation for the testbed, and the follow-up paper [Pasupathy and Henderson \(2011\)](#) describes an earlier interface for MATLAB implementations of problems and solvers. The paper [Dong et al. \(2017\)](#) conducts an experimental comparison of several solvers in SimOpt and analyzes their relative performance. The recent Winter Simulation Conference paper [Eckman et al. \(2019\)](#) describes in detail the recent changes to the architecture of SimOpt and the control of random number streams.

The [models](#) library contains the simulation logic to simulate a variety of systems and SO test problems built around these models. The [solvers](#) library provides users with the latest SO solvers to solve different types of SO problems. The two libraries are intended to help researchers evaluate and compare the finite-time performance of existing solvers.

The source code consists of the following modules:

- The [base.py](#) module contains class definitions for models, problems, and solvers.
- The [experiment\\_base.py](#) module contains class definitions and functions for running experiments with simulation-optimization solvers.
- The [data\\_farming\\_base.py](#) module contains class definitions and functions for running data-farming experiments.
- The [directory.py](#) module contains a listing of models, problems, and solvers in the library.



## **GETTING STARTED**

Please make sure you have the following dependencies installed: Python 3, numpy, scipy, matplotlib, seaborn, and mrg32k3a. Then clone the repo. To see an example of how to run an experiment on a solver and problem pair, please view or run `demo/demo_problem_solver.py`.





## CONTENTS

## 2.1 stable

### 2.1.1 simopt package

#### 2.1.1.1 Subpackages

#### simopt.models package

#### Submodules

#### simopt.models.amusementpark module

#### Summary

Simulate a single day of operation for an amusement park queuing problem. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.amusementpark.AmusementPark` (*fixed\_factors=None*)

Bases: *Model*

A model that simulates a single day of operation for an amusement park queuing problem based on a poisson distributed tourist arrival rate, a next attraction transition matrix, and attraction durations based on an Erlang distribution. Returns the total number and percent of tourists to leave the park due to full queues.

#### **name**

name of model

#### **Type**

*str*

#### **n\_rngs**

number of random-number generators used to run a simulation replication

#### **Type**

*int*

#### **n\_responses**

number of responses (performance measures)

#### **Type**

*int*

**factors**

changeable factors of the simulation model

**Type**

`dict`

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**check\_factor\_list**

switch case for checking factor simulatability

**Type**

`dict`

**Parameters**

**fixed\_factors** (*dict*) – fixed\_factors of the simulation model

**See also:**

`base.Model`

`check_arrival_gammas()`

`check_depart_probabilities()`

`check_erlang_scale()`

`check_erlang_shape()`

`check_number_attractions()`

`check_park_capacity()`

`check_queue_capacities()`

`check_simulatable_factors()`

Determine if a simulation replication can be run with the given factors.

**Notes**

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

**Returns**

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

**Return type**

`bool`

`check_time_open()`

`check_transition_probabilities()`

**replicate** (*rng\_list*)

Simulate a single replication for the current model factors.

**Parameters**

**rng\_list** (*[list]* [*rng.mrg32k3a.MRG32k3a*]) – rngs for model to use when simulating a replication

**Returns**

**responses** – performance measures of interest “total\_departed\_tourists”: The total number of tourists to leave the park due

to full queues

**”percent\_departed\_tourists”**: The percentage of tourists to leave the park due to full queues

**Return type**

dict

**class** simopt.models.amusementpark.**AmusementParkMinDepart** (*name='AMUSEMENTPARK-1', fixed\_factors=None, model\_fixed\_factors=None*)

Bases: *Problem*

Class to make amusement park simulation-optimization problems.

**name**

name of problem

**Type**

str

**dim**

number of decision variables

**Type**

int

**n\_objectives**

number of objectives

**Type**

int

**n\_stochastic\_constraints**

number of stochastic constraints

**Type**

int

**minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type**

tuple of int (+/- 1)

**constraint\_type**

**description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

str

**variable\_type**

**description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

str

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

base.Model

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

[list] [rng.mrg32k3a.MRG32k3a]

**factors****changeable factors of the problem****initial\_solution**

[tuple] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name of problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

bool

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters****factor\_dict** (*dict*) – dictionary with factor keys and associated values**Returns****vector** – vector of values associated with decision variables**Return type***tuple***get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters****rand\_sol\_rng** (*rng.mrg32k3a.MRG32k3a*) – random-number generator used to sample a new random solution**Returns****x** – vector of decision variables**Return type***tuple***response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters****response\_dict** (*dict*) – dictionary with response keys and associated values**Returns****objectives** – vector of objectives**Return type***tuple***response\_dict\_to\_stoch\_constraints** (*response\_dict*)Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$ **Parameters****response\_dict** (*dict*) – dictionary with response keys and associated values**Returns****stoch\_constraints** – vector of LHSs of stochastic constraint**Return type***tuple***vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters****vector** (*tuple*) – vector of values associated with decision variables

**Returns****factor\_dict** – dictionary with factor keys and associated values**Return type**

dict

**simopt.models.chessmm module****Summary**

Simulate matching of chess players on an online platform. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.chessmm.ChessAvgDifference` (*name='CHESS-1', fixed\_factors=None, model\_fixed\_factors=None*)

Bases: *Problem*

Base class to implement simulation-optimization problems.

**name**

name of problem

**Type**

string

**dim**

number of decision variables

**Type**

int

**n\_objectives**

number of objectives

**Type**

int

**n\_stochastic\_constraints**

number of stochastic constraints

**Type**

int

**minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type**

tuple of int (+/- 1)

**constraint\_type**

**description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**lower\_bounds**

lower bound for each decision variable

**Type**

tuple

**upper\_bounds**

upper bound for each decision variable

**Type**

tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

list of `mrg32k3a.mrg32k3a.MRG32k3a` objects



**factors****changeable factors of the problem****initial\_solution**

[list] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**prev\_cost**

[list] cost of prevention

**upper\_thres**

[float > 0] upper limit of amount of contamination

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name for problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

bool

**check\_upper\_time** ()**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives

- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**`tuple`**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters****vector** (*tuple*) – vector of values associated with decision variables**Returns****factor\_dict** – dictionary with factor keys and associated values**Return type**

dictionary

**class** `simopt.models.chessmm.ChessMatchmaking` (*fixed\_factors=None*)Bases: `Model`

A model that simulates a matchmaking problem with a Elo (truncated normal) distribution of players and Poisson arrivals. Returns the average difference between matched players.

**name**

name of model

**Type**

string

**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**`int`**n\_responses**

number of responses (performance measures)

**Type**`int`**factors**

changeable factors of the simulation model

**Type**`dict`**specifications**

details of each factor (for GUI and data validation)

**Type**`dict`**check\_factor\_list**

switch case for checking factor simulatability

**Type**`dict`**Parameters****fixed\_factors** (*nested dict*) – fixed factors of the simulation model

See also:

`base.Model`

`check_allowable_diff()`

`check_elo_mean()`

`check_elo_sd()`

`check_num_players()`

`check_poisson_rate()`

`replicate(rng_list)`

Simulate a single replication for the current model factors.

**Parameters**

**rng\_list** (*list of mrg32k3a.mrg32k3a.MRG32k3a objects*) – rngs for model to use when simulating a replication

**Returns**

- **responses** (*dict*) – performance measures of interest “avg\_diff” = the average Elo difference between all pairs “avg\_wait\_time” = the average waiting time
- **gradients** (*dict of dicts*) – gradient estimates for each response

## simopt.models.cntnv module

### Summary

Simulate a day’s worth of sales for a newsvendor. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.cntnv.CntNV` (*fixed\_factors=None*)

Bases: `Model`

A model that simulates a day’s worth of sales for a newsvendor with a Burr Type XII demand distribution. Returns the profit, after accounting for order costs and salvage.

**name**

name of model

**Type**

string

**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**

int

**n\_responses**

number of responses (performance measures)

**Type**

int

**factors**

changeable factors of the simulation model

**Type**

`dict`

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**check\_factor\_list**

switch case for checking factor simulatability

**Type**

`dict`

**Parameters**

**fixed\_factors** (*dict*) – fixed\_factors of the simulation model

**See also:**

`base.Model`

`check_Burr_c()`

`check_Burr_k()`

`check_order_quantity()`

`check_purchase_price()`

`check_sales_price()`

`check_salvage_price()`

`check_simulatable_factors()`

Determine if a simulation replication can be run with the given factors.

**Notes**

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

**Returns**

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

**Return type**

`bool`

**replicate** (*rng\_list*)

Simulate a single replication for the current model factors.

**Parameters**

**rng\_list** (*list of `mrg32k3a.mrg32k3a.MRG32k3a` objects*) – rngs for model to use when simulating a replication

**Returns**

**responses** – performance measures of interest “profit” = profit in this scenario “stockout\_qty” = amount by which demand exceeded supply “stockout” = was there unmet demand? (Y/N)

**Return type**

dict

```
class simopt.models.cntnv.CntNVMaxProfit (name='CNTNEWS-1', fixed_factors=None,  
                                         model_fixed_factors=None)
```

Bases: *Problem*

Base class to implement simulation-optimization problems.

**name**

name of problem

**Type**

string

**dim**

number of decision variables

**Type**

int

**n\_objectives**

number of objectives

**Type**

int

**n\_stochastic\_constraints**

number of stochastic constraints

**Type**

int

**minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type**

tuple of int (+/- 1)

**constraint\_type****description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**lower\_bounds**

lower bound for each decision variable

**Type**

tuple

**upper\_bounds**

upper bound for each decision variable

**Type**

tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

list of mrg32k3a.mrg32k3a.MRG32k3a objects

**factors**

**changeable factors of the problem**

**initial\_solution**

[tuple] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name for problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

bool

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints



**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrng32k3a.mrng32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

dictionary

## simopt.models.contam module

### Summary

Simulate contamination rates. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.contam.Contamination` (*fixed\_factors=None*)

Bases: *Model*

A model that simulates a contamination problem with a beta distribution. Returns the probability of violating contamination upper limit in each level of supply chain.

**name**

name of model

**Type**

string

**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**

int

**n\_responses**

number of responses (performance measures)

**Type**

int

**factors**

changeable factors of the simulation model

**Type**

dict

**specifications**

details of each factor (for GUI and data validation)

**Type**

dict

**check\_factor\_list**

switch case for checking factor simulatability

**Type**

dict

**Parameters**

**fixed\_factors** (*nested dict*) – fixed factors of the simulation model

**See also:**

`base.Model`

`check_contam_rate_alpha()`

`check_contam_rate_beta()`

`check_initial_rate_alpha()`

`check_initial_rate_beta()`

`check_prev_cost()`

`check_prev_decision()`

`check_restore_rate_alpha()`

`check_restore_rate_beta()`

`check_simulatable_factors()`

Determine if a simulation replication can be run with the given factors.

### Notes

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

#### Returns

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

#### Return type

`bool`

`check_stages()`

`replicate(rng_list)`

Simulate a single replication for the current model factors.

#### Parameters

**rng\_list** (*list of `mrg32k3a.mrg32k3a.MRG32k3a` objects*) – rngs for model to use when simulating a replication

#### Returns

- **responses** (*dict*) – performance measures of interest “level” = a list of contamination levels over time
- **gradients** (*dict of dicts*) – gradient estimates for each response

```
class simopt.models.contam.ContaminationTotalCostCont (name='CONTAM-2',
                                                    fixed_factors=None,
                                                    model_fixed_factors=None)
```

Bases: `Problem`

Base class to implement simulation-optimization problems.

#### name

name of problem

#### Type

`string`

#### dim

number of decision variables

#### Type

`int`

**n\_objectives**

number of objectives

**Type**

int

**n\_stochastic\_constraints**

number of stochastic constraints

**Type**

int

**minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type**

tuple of int (+/- 1)

**constraint\_type****description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**lower\_bounds**

lower bound for each decision variable

**Type**

tuple

**upper\_bounds**

upper bound for each decision variable

**Type**

tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**list of `mrg32k3a.mrg32k3a.MRG32k3a` objects**factors****changeable factors of the problem****initial\_solution**

[list] default initial solution from which solvers start

**budget**

[int &gt; 0] max number of replications (fn evals) for a solver to take

**prev\_cost**

[list] cost of prevention

**upper\_thres**

[float &gt; 0] upper limit of amount of contamination

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name for problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors

- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

See also:

`base.Problem`

**check\_budget** ()

Check if budget is strictly positive.

**Returns**

True if budget is strictly positive, otherwise False.

**Return type**

`bool`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

`bool`

**check\_error\_prob** ()

**check\_initial\_solution** ()

Check if initial solution is feasible and of correct dimension.

**Returns**

True if initial solution is feasible and of correct dimension, otherwise False.

**Return type**

`bool`

**check\_prev\_cost** ()

**check\_simulatable\_factors** ()

**check\_upper\_thres** ()

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**factor\_dict\_to\_vector\_gradients** (*factor\_dict*)

Convert a dictionary with factor keys to a gradient vector.

**Notes**

A subclass of `base.Problem` can have its own custom `factor_dict_to_vector_gradients` method if the objective is deterministic.

**Parameters**

**factor\_dict** (*dict*) – Dictionary with factor keys and associated values.

**Returns**

**vector** – Vector of partial derivatives associated with decision variables.

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_objectives\_gradients** (*response\_dict*)

Convert a dictionary with response keys to a vector of gradients.

### Notes

A subclass of `base.Problem` can have its own custom `response_dict_to_objectives_gradients` method if the objective is deterministic.

#### Parameters

**response\_dict** (*dict*) – Dictionary with response keys and associated values.

#### Returns

Vector of gradients.

#### Return type

`tuple`

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

#### Parameters

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

#### Returns

**stoch\_constraints** – vector of LHSs of stochastic constraint

#### Return type

`tuple`

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

#### Parameters

**vector** (*tuple*) – vector of values associated with decision variables

#### Returns

**factor\_dict** – dictionary with factor keys and associated values

#### Return type

dictionary

```
class simopt.models.contam.ContaminationTotalCostDisc (name='CONTAM-1',  
                                                    fixed_factors=None,  
                                                    model_fixed_factors=None)
```

Bases: `Problem`

Base class to implement simulation-optimization problems.

#### **name**

name of problem

#### **Type**

string

#### **dim**

number of decision variables

#### **Type**

`int`



**n\_objectives**  
 number of objectives  
**Type**  
 int

**n\_stochastic\_constraints**  
 number of stochastic constraints  
**Type**  
 int

**minmax**  
 indicator of maximization (+1) or minimization (-1) for each objective  
**Type**  
 tuple of int (+/- 1)

**constraint\_type**  
**description of constraints types:**  
 “unconstrained”, “box”, “deterministic”, “stochastic”  
**Type**  
 string

**variable\_type**  
**description of variable types:**  
 “discrete”, “continuous”, “mixed”  
**Type**  
 string

**lower\_bounds**  
 lower bound for each decision variable  
**Type**  
 tuple

**upper\_bounds**  
 upper bound for each decision variable  
**Type**  
 tuple

**gradient\_available**  
 indicates if gradient of objective function is available  
**Type**  
 bool

**optimal\_value**  
 optimal objective function value  
**Type**  
 float

**optimal\_solution**

optimal solution

**Type**

`tuple`

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

`dict`

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

`dict`

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

list of `mrg32k3a.mrg32k3a.MRG32k3a` objects

**factors****changeable factors of the problem****initial\_solution**

[list] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**prev\_cost**

[list] cost of prevention

**upper\_thres**

[float > 0] upper limit of amount of contamination

**Type**

`dict`

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**Parameters**

- **name** (`str`) – user-specified name for problem
- **fixed\_factors** (`dict`) – dictionary of user-specified problem factors

- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

See also:

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

`bool`

**check\_error\_prob** ()

**check\_prev\_cost** ()

**check\_upper\_thres** ()

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

`tuple`

**factor\_dict\_to\_vector\_gradients** (*factor\_dict*)

Convert a dictionary with factor keys to a gradient vector.

### Notes

A subclass of `base.Problem` can have its own custom `factor_dict_to_vector_gradients` method if the objective is deterministic.

#### Parameters

**factor\_dict** (*dict*) – Dictionary with factor keys and associated values.

#### Returns

**vector** – Vector of partial derivatives associated with decision variables.

#### Return type

`tuple`

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

#### Parameters

**rand\_sol\_rng** (*mrng32k3a.mrng32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

#### Returns

**x** – vector of decision variables

#### Return type

`tuple`

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

#### Parameters

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

#### Returns

**objectives** – vector of objectives

#### Return type

`tuple`

**response\_dict\_to\_objectives\_gradients** (*response\_dict*)

Convert a dictionary with response keys to a vector of gradients.

### Notes

A subclass of `base.Problem` can have its own custom `response_dict_to_objectives_gradients` method if the objective is deterministic.

#### Parameters

**response\_dict** (*dict*) – Dictionary with response keys and associated values.

#### Returns

Vector of gradients.

#### Return type

`tuple`

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

*dictionary*

## simopt.models.dualsourcing module

### Summary

Simulate multiple periods of ordering and sales for a dual sourcing inventory problem. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.dualsourcing.DualSourcing` (*fixed\_factors=None*)

Bases: *Model*

A model that simulates multiple periods of ordering and sales for a single-staged, dual sourcing inventory problem with stochastic demand. Returns average holding cost, average penalty cost, and average ordering cost per period.

**name**

name of model

**Type**

*str*

**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**

*int*

**n\_responses**

number of responses (performance measures)

**Type**

*int*

**factors**

changeable factors of the simulation model

**Type**

*dict*

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

*dict*

**check\_factor\_list**

switch case for checking factor simulatability

**Type**

*dict*

**Parameters**

**fixed\_factors** (*dict*) – fixed\_factors of the simulation model

**n\_days**

Number of days to simulate (*int*)

**initial\_inv**

Initial inventory (*int*)

**cost\_reg**

Regular ordering cost per unit (*flt*)

**cost\_exp**

Expedited ordering cost per unit (*flt*)

**lead\_reg**

Lead time for regular orders in days (*int*)

**lead\_exp**

Lead time for expedited orders in days (*int*)

**holding\_cost**

Holding cost per unit per period (*flt*)

**penalty\_cost**

Penalty cost per unit per period for backlogging(*flt*)

**st\_dev**

Standard deviation of demand distribution (*flt*)

**mu**

Mean of demand distribution (*flt*)

**order\_level\_reg**

Order-up-to level for regular orders (*int*)

**order\_level\_exp**

Order-up-to level for expedited orders (*int*)

**See also:**

`base.Model`

`check_cost_exp()`

`check_cost_reg()`

`check_holding_cost()`

`check_initial_inv()`

`check_lead_exp()``check_lead_reg()``check_mu()``check_n_days()``check_order_level_exp()``check_order_level_reg()``check_penalty_cost()``check_simulatable_factors()`

Determine if a simulation replication can be run with the given factors.

### Notes

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

#### Returns

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

#### Return type

`bool`

`check_st_dev()``replicate(rng_list)`

Simulate a single replication for the current model factors.

#### Parameters

**rng\_list** (*[list]* `[mrg32k3a.mrg32k3a.MRG32k3a]`) – rngs for model to use when simulating a replication

#### Returns

**responses** – performance measures of interest

**average\_holding\_cost**

The average holding cost over the time period

**average\_penalty\_cost**

The average penalty cost over the time period

**average\_ordering\_cost**

The average ordering cost over the time period

#### Return type

`dict`

```
class simopt.models.dualsourcing.DualSourcingMinCost (name='DUALSOURCING-I',
                                                    fixed_factors=None,
                                                    model_fixed_factors=None)
```

Bases: `Problem`

Class to make dual-sourcing inventory simulation-optimization problems.

**name**  
name of problem  
Type  
str

**dim**  
number of decision variables  
Type  
int

**n\_objectives**  
number of objectives  
Type  
int

**n\_stochastic\_constraints**  
number of stochastic constraints  
Type  
int

**minmax**  
indicator of maximization (+1) or minimization (-1) for each objective  
Type  
tuple of int (+/- 1)

**constraint\_type**  
**description of constraints types:**  
“unconstrained”, “box”, “deterministic”, “stochastic”  
Type  
str

**variable\_type**  
**description of variable types:**  
“discrete”, “continuous”, “mixed”  
Type  
str

**gradient\_available**  
indicates if gradient of objective function is available  
Type  
bool

**optimal\_value**  
optimal objective function value  
Type  
tuple



**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

base.Model

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

[list] [mrg32k3a.mrg32k3a.MRG32k3a]

**factors****changeable factors of the problem****initial\_solution**

[tuple] default initial solution from which solvers start

**budget**

[int &gt; 0] max number of replications (fn evals) for a solver to take

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (str) – user-specified name of problem
- **fixed\_factors** (dict) – dictionary of user-specified problem factors

- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

See also:

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

`bool`

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dict*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

`tuple`

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrng32k3a*, *mrng32k3a*, *MRG32k3a*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

`tuple`

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dict*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

`tuple`

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dict*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

`tuple`

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys.

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

`dict`

## **simopt.models.dynamnews module**

### **Summary**

Simulate a day's worth of sales for a newsvendor under dynamic consumer substitution. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.dynamnews.DynamNews` (*fixed\_factors=None*)

Bases: `Model`

A model that simulates a day's worth of sales for a newsvendor with dynamic consumer substitution. Returns the profit and the number of products that stock out.

**name**

name of model

**Type**

string

**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**

*int*

**n\_responses**

number of responses (performance measures)

**Type**

*int*

**factors**

changeable factors of the simulation model

**Type**

*dict*

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

*dict*

**check\_factor\_list**

switch case for checking factor simulatability

**Type**

*dict*

**Parameters**

**fixed\_factors** (*dict*) – fixed\_factors of the simulation model

**See also:**

`base.Model`

`check_c_utility()`

`check_cost()`

`check_init_level()`

`check_mu()`

`check_num_customer()`

`check_num_prod()`

`check_price()`

`check_simulatable_factors()`

Determine if a simulation replication can be run with the given factors.

## Notes

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

### Returns

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

### Return type

`bool`

**replicate** (*rng\_list*)

Simulate a single replication for the current model factors.

### Parameters

**rng\_list** (*list of mrg32k3a.mrg32k3a.MRG32k3a objects*) – rngs for model to use when simulating a replication

### Returns

**responses** – performance measures of interest “profit” = profit in this scenario  
“n\_prod\_stockout” = number of products which are out of stock

### Return type

`dict`

```
class simopt.models.dynamnews.DynamNewsMaxProfit (name='DYNAMNEWS-1',
                                                fixed_factors=None,
                                                model_fixed_factors=None)
```

Bases: *Problem*

Base class to implement simulation-optimization problems.

### name

name of problem

### Type

`string`

### dim

number of decision variables

### Type

`int`

### n\_objectives

number of objectives

### Type

`int`

### n\_stochastic\_constraints

number of stochastic constraints

### Type

`int`

### minmax

indicator of maximization (+1) or minimization (-1) for each objective

### Type

`tuple of int (+/- 1)`

**constraint\_type**

**description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type**

**description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**lower\_bounds**

lower bound for each decision variable

**Type**

tuple

**upper\_bounds**

upper bound for each decision variable

**Type**

tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

`dict`

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of `str`

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

list of `mrg32k3a.mrg32k3a.MRG32k3a` objects

**factors****changeable factors of the problem****initial\_solution**

[tuple] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**Type**

`dict`

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**Parameters**

- **name** (`str`) – user-specified name for problem
- **fixed\_factors** (`dict`) – dictionary of user-specified problem factors
- **factors** (`model_fixed`) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (`tuple`) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

`bool`

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*



**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

*dictionary*

## simopt.models.example module

### Summary

Simulate a synthetic problem with a deterministic objective function evaluated with noise.

**class** `simopt.models.example.ExampleModel` (*fixed\_factors=None*)

Bases: *Model*

A model that is a deterministic function evaluated with noise.

**name**

name of model

**Type**

*string*

**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**

*int*

**n\_responses**

number of responses (performance measures)

**Type**

*int*

**factors**

changeable factors of the simulation model

**Type**

*dict*

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**check\_factor\_list**

switch case for checking factor simulatability

**Type**

`dict`

**Parameters**

**fixed\_factors** (*dict*) – fixed\_factors of the simulation model

**See also:**

`base.Model`

**check\_simulatable\_factors()**

Determine if a simulation replication can be run with the given factors.

**Notes**

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

**Returns**

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

**Return type**

`bool`

**check\_x()****replicate** (*rng\_list*)

Evaluate a deterministic function  $f(x)$  with stochastic noise.

**Parameters**

**rng\_list** (*list of mrg32k3a.mrg32k3a.MRG32k3a objects*) – rngs for model to use when simulating a replication

**Returns**

**responses** – performance measures of interest “ $\text{est}_f(x)$ ” =  $f(x)$  evaluated with stochastic noise

**Return type**

`dict`

```
class simopt.models.example.ExampleProblem (name='EXAMPLE-1', fixed_factors=None,  
                                             model_fixed_factors=None)
```

Bases: `Problem`

Base class to implement simulation-optimization problems.

**name**

name of problem

**Type**

`string`

**dim**  
number of decision variables  
**Type**  
int

**n\_objectives**  
number of objectives  
**Type**  
int

**n\_stochastic\_constraints**  
number of stochastic constraints  
**Type**  
int

**minmax**  
indicator of maximization (+1) or minimization (-1) for each objective  
**Type**  
tuple of int (+/- 1)

**constraint\_type**  
**description of constraints types:**  
“unconstrained”, “box”, “deterministic”, “stochastic”  
**Type**  
string

**variable\_type**  
**description of variable types:**  
“discrete”, “continuous”, “mixed”  
**Type**  
string

**lower\_bounds**  
lower bound for each decision variable  
**Type**  
tuple

**upper\_bounds**  
upper bound for each decision variable  
**Type**  
tuple

**gradient\_available**  
indicates if gradient of objective function is available  
**Type**  
bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

list of mrg32k3a.mrg32k3a.MRG32k3a objects

**factors****changeable factors of the problem****initial\_solution**

[tuple] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name for problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

`bool`

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

`tuple`

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

dictionary

## simopt.models.facilitysizing module

### Summary

Simulate demand at facilities. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.facilitysizing.FacilitySize` (*fixed\_factors=None*)

Bases: *Model*

A model that simulates a facilitysize problem with a multi-variate normal distribution. Returns the probability of violating demand in each scenario.

**name**  
 name of model  
**Type**  
 string

**n\_rngs**  
 number of random-number generators used to run a simulation replication  
**Type**  
 int

**n\_responses**  
 number of responses (performance measures)  
**Type**  
 int

**factors**  
 changeable factors of the simulation model  
**Type**  
 dict

**specifications**  
 details of each factor (for GUI and data validation)  
**Type**  
 dict

**check\_factor\_list**  
 switch case for checking factor simulatability  
**Type**  
 dict

**Parameters**  
**fixed\_factors** (*nested dict*) – fixed factors of the simulation model

**See also:**  
 base.Model

**check\_capacity()**

**check\_cov()**

**check\_mean\_vec()**

**check\_n\_fac()**

**check\_simulatable\_factors()**  
 Determine if a simulation replication can be run with the given factors.

## Notes

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

### Returns

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

### Return type

`bool`

**replicate** (*rng\_list*)

Simulate a single replication for the current model factors.

### Parameters

**rng\_list** (*list of `mrg32k3a.mrg32k3a.MRG32k3a` objects*) – rngs for model to use when simulating a replication

### Returns

- **responses** (*dict*) – performance measures of interest “stockout\_flag” = a binary variable  
0 : all facilities satisfy the demand 1 : at least one of the facilities did not satisfy the demand  
“n\_fac\_stockout” = the number of facilities which cannot satisfy the demand “n\_cut” = the number of total demand which cannot be satisfied
- **gradients** (*dict of dicts*) – gradient estimates for each response

```
class simopt.models.facilitysizing.FacilitySizingMaxService (name='FACSIZE-2',  
                                                         fixed_factors=None,  
                                                         model_fixed_factors=None)
```

Bases: *Problem*

Base class to implement simulation-optimization problems.

### name

name of problem

### Type

`string`

### dim

number of decision variables

### Type

`int`

### n\_objectives

number of objectives

### Type

`int`

### n\_stochastic\_constraints

number of stochastic constraints

### Type

`int`



**minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type**

tuple of int (+/- 1)

**constraint\_type****description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**lower\_bounds**

lower bound for each decision variable

**Type**

tuple

**upper\_bounds**

upper bound for each decision variable

**Type**

tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

`dict`

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

`dict`

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

`set of str`

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

`list of mrg32k3a.mrg32k3a.MRG32k3a objects`

**factors****changeable factors of the problem****initial\_solution**

`[tuple]` default initial solution from which solvers start

**budget**

`[int > 0]` max number of replications (fn evals) for a solver to take

**Type**

`dict`

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**Parameters**

- **name** (`str`) – user-specified name for problem
- **fixed\_factors** (`dict`) – dictionary of user-specified problem factors
- **factors** (`model_fixed`) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_deterministic\_constraints** (`x`)

Check if a solution `x` satisfies the problem's deterministic constraints.

**Parameters**

**x** (`tuple`) – vector of decision variables

**Returns**

**satisfies** – indicates if solution  $x$  satisfies the deterministic constraints.

**Return type**

`bool`

**check\_installation\_budget** ()

**check\_installation\_costs** ()

**deterministic\_objectives\_and\_gradients** ( $x$ )

Compute deterministic components of objectives for a solution  $x$ .

**Parameters**

**$x$**  (`tuple`) – vector of decision variables

**Returns**

- **det\_objectives** (`tuple`) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (`tuple`) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** ( $x$ )

Compute deterministic components of stochastic constraints for a solution  $x$ .

**Parameters**

**$x$**  (`tuple`) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (`tuple`) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (`tuple`) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (`factor_dict`)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (`dictionary`) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

`tuple`

**get\_random\_solution** (`rand_sol_rng`)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (`mrg32k3a.mrg32k3a.MRG32k3a object`) – random-number generator used to sample a new random solution

**Returns**

**$x$**  – vector of decision variables

**Return type**

`tuple`

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

dictionary

```
class simopt.models.facilitysizing.FacilitySizingTotalCost (name='FACSIZE-1',  
                                                         fixed_factors=None,  
                                                         model_fixed_factors=None)
```

Bases: *Problem*

Base class to implement simulation-optimization problems.

**name**

name of problem

**Type**

string

**dim**

number of decision variables

**Type**

*int*

**n\_objectives**

number of objectives

**Type**

*int*

**n\_stochastic\_constraints**

number of stochastic constraints

**Type**

int

**minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type**

tuple of int (+/- 1)

**constraint\_type****description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**lower\_bounds**

lower bound for each decision variable

**Type**

tuple

**upper\_bounds**

upper bound for each decision variable

**Type**

tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

float

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

list of mrg32k3a.mrg32k3a.MRG32k3a objects

**factors****changeable factors of the problem****initial\_solution**

[tuple] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name for problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

**See also:**

base.Problem

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

*bool*

**check\_epsilon** ()**check\_installation\_costs** ()**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**factor\_dict\_to\_vector\_gradients** (*factor\_dict*)

Convert a dictionary with factor keys to a gradient vector.

## Notes

A subclass of `base.Problem` can have its own custom `factor_dict_to_vector_gradients` method if the objective is deterministic.

### Parameters

**factor\_dict** (*dict*) – Dictionary with factor keys and associated values.

### Returns

**vector** – Vector of partial derivatives associated with decision variables.

### Return type

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

### Parameters

**rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

### Returns

**x** – vector of decision variables

### Return type

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

### Parameters

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

### Returns

**objectives** – vector of objectives

### Return type

*tuple*

**response\_dict\_to\_objectives\_gradients** (*response\_dict*)

Convert a dictionary with response keys to a vector of gradients.

## Notes

A subclass of `base.Problem` can have its own custom `response_dict_to_objectives_gradients` method if the objective is deterministic.

### Parameters

**response\_dict** (*dict*) – Dictionary with response keys and associated values.

### Returns

Vector of gradients.

### Return type

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

### Parameters

**response\_dict** (*dictionary*) – dictionary with response keys and associated values



**Returns****stoch\_constraints** – vector of LHSs of stochastic constraint**Return type**`tuple`**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters****vector** (*tuple*) – vector of values associated with decision variables**Returns****factor\_dict** – dictionary with factor keys and associated values**Return type**

dictionary

**simopt.models.fixedsan module****Summary**

Simulate duration of a stochastic activity network (SAN). A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.fixedsan.FixedSAN` (*fixed\_factors=None*)

Bases: `Model`

A model that simulates a stochastic activity network problem with tasks that have exponentially distributed durations, and the selected means come with a cost.

**name**

name of model

**Type**`string`**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**`int`**n\_responses**

number of responses (performance measures)

**Type**`int`**factors**

changeable factors of the simulation model

**Type**`dict`**specifications**

details of each factor (for GUI and data validation)

**Type**`dict`

**check\_factor\_list**

switch case for checking factor simulatability

**Type**

`dict`

**Parameters**

**fixed\_factors** (*nested dict*) – fixed factors of the simulation model

**See also:**

`base.Model`

**check\_arc\_means** ()

**check\_num\_arcs** ()

**check\_num\_nodes** ()

**replicate** (*rng\_list*)

Simulate a single replication for the current model factors.

**Parameters**

**rng\_list** (*list of mrg32k3a.mrg32k3a.MRG32k3a objects*) – rngs for model to use when simulating a replication

**Returns**

- **responses** (*dict*) – performance measures of interest “longest\_path\_length” = length/duration of longest path
- **gradients** (*dict of dicts*) – gradient estimates for each response

**class** `simopt.models.fixedsan.FixedSANLongestPath` (*name='FIXEDSAN-1', fixed\_factors=None, model\_fixed\_factors=None*)

Bases: [\*Problem\*](#)

Base class to implement simulation-optimization problems.

**name**

name of problem

**Type**

`string`

**dim**

number of decision variables

**Type**

`int`

**n\_objectives**

number of objectives

**Type**

`int`

**n\_stochastic\_constraints**

number of stochastic constraints

**Type**

`int`

**minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type**

tuple of int (+/- 1)

**constraint\_type****description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**lower\_bounds**

lower bound for each decision variable

**Type**

tuple

**upper\_bounds**

upper bound for each decision variable

**Type**

tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

`dict`

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

`dict`

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

`set of str`

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

`list of mrg32k3a.mrg32k3a.MRG32k3a objects`

**factors****changeable factors of the problem****initial\_solution**

[list] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**Type**

`dict`

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**Parameters**

- **name** (`str`) – user-specified name for problem
- **fixed\_factors** (`dict`) – dictionary of user-specified problem factors
- **factors** (`model_fixed`) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

`check_arc_costs()`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

*bool*

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

dictionary

## simopt.models.hotel module

### Summary

Simulate expected revenue for a hotel. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.hotel.Hotel` (*fixed\_factors=None*)

Bases: *Model*

A model that simulates business of a hotel with Poisson arrival rate.

**name**

name of model

**Type**

string

**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**

*int*

**n\_responses**

number of responses (performance measures)

**Type**

int

**factors**

changeable factors of the simulation model

**Type**

dict

**specifications**

details of each factor (for GUI and data validation)

**Type**

dict

**check\_factor\_list**

switch case for checking factor simulatability

**Type**

dict

**Parameters**

**fixed\_factors** (*nested dict*) – fixed factors of the simulation model

**See also:**

`base.Model`

`check_booking_limits()`

`check_discount_rate()`

`check_lambda()`

`check_num_products()`

`check_num_rooms()`

`check_product_incidence()`

`check_rack_rate()`

`check_runlength()`

`check_time_before()`

`check_time_limit()`

**replicate** (*rng\_list*)

Simulate a single replication for the current model factors.

**Parameters**

**rng\_list** (*list of mrg32k3a.mrg32k3a.MRG32k3a objects*) – rngs for model to use when simulating a replication

**Returns**

- **responses** (*dict*) – performance measures of interest “revenue” = expected revenue

- **gradients** (*dict of dicts*) – gradient estimates for each response

```
class simopt.models.hotel.HotelRevenue (name='HOTEL-1', fixed_factors=None,  
                                         model_fixed_factors=None)
```

Bases: *Problem*

Base class to implement simulation-optimization problems.

**name**

name of problem

**Type**

string

**dim**

number of decision variables

**Type**

int

**n\_objectives**

number of objectives

**Type**

int

**n\_stochastic\_constraints**

number of stochastic constraints

**Type**

int

**minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type**

tuple of int (+/- 1)

**constraint\_type**

**description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type**

**description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**lower\_bounds**

lower bound for each decision variable

**Type**

tuple



**upper\_bounds**

upper bound for each decision variable

**Type**

tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

list of mrg32k3a.mrg32k3a.MRG32k3a objects

**factors**

**changeable factors of the problem**

**initial\_solution**

[list] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name for problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_budget()**

Check if budget is strictly positive.

**Returns**

True if budget is strictly positive, otherwise False.

**Return type**

bool

**check\_deterministic\_constraints(x)**

Check if a solution  $x$  satisfies the problem's deterministic constraints.

**Parameters**

$\mathbf{x}$  (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution  $x$  satisfies the deterministic constraints.

**Return type**

bool

**check\_initial\_solution()**

Check if initial solution is feasible and of correct dimension.

**Returns**

True if initial solution is feasible and of correct dimension, otherwise False.

**Return type**

bool

**check\_simulatable\_factors()****deterministic\_objectives\_and\_gradients(x)**

Compute deterministic components of objectives for a solution  $x$ .

**Parameters**

$\mathbf{x}$  (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns****stoch\_constraints** – vector of LHSs of stochastic constraint**Return type**`tuple`**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters****vector** (*tuple*) – vector of values associated with decision variables**Returns****factor\_dict** – dictionary with factor keys and associated values**Return type**

dictionary

**simopt.models.ironore module****Summary**

Simulate multiple periods of production and sales for an iron ore inventory problem. A detailed description of the model/problem can be found [here](#).

**Changed get\_random\_solution quantiles**

from 10 and 200 => mean=59.887, sd=53.338, p(X>100)=0.146 to 10 and 1000 => mean=199.384, sd=343.925, p(X>100)=0.5

**class** `simopt.models.ironore.IronOre` (*fixed\_factors=None*)Bases: `Model`

A model that simulates multiple periods of production and sales for an inventory problem with stochastic price determined by a mean-reverting random walk. Returns total profit, fraction of days producing iron, and mean stock.

**name**

name of model

**Type**`str`**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**`int`**n\_responses**

number of responses (performance measures)

**Type**`int`**factors**

changeable factors of the simulation model

**Type**`dict`

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**check\_factor\_list**

switch case for checking factor simulatability

**Type**

`dict`

**Parameters**

**fixed\_factors** (*dict*) – fixed\_factors of the simulation model

**See also:**

`base.Model`

`check_capacity()`

`check_holding_cost()`

`check_inven_stop()`

`check_max_price()`

`check_max_prod_perday()`

`check_mean_price()`

`check_min_price()`

`check_n_days()`

`check_price_prod()`

`check_price_sell()`

`check_price_stop()`

`check_prod_cost()`

`check_simulatable_factors()`

Determine if a simulation replication can be run with the given factors.

**Notes**

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

**Returns**

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

**Return type**

`bool`

`check_st_dev()`

**replicate** (*rng\_list*)

Simulate a single replication for the current model factors.

**Parameters****rng\_list** (*[list]* *[mrg32k3a.mrg32k3a.MRG32k3a]*) – rngs for model to use when simulating a replication**Returns****responses** – performance measures of interest “total\_profit” = The total profit over the time period “frac\_producing” = The fraction of days spent producing iron ore “mean\_stock” = The average stocks over the time period**Return type***dict***class** `simopt.models.ironore.IronOreMaxRev` (*name='IRONORE-1'*, *fixed\_factors=None*,  
*model\_fixed\_factors=None*)Bases: *Problem*

Class to make iron ore inventory simulation-optimization problems.

**name**

name of problem

**Type***str***dim**

number of decision variables

**Type***int***n\_objectives**

number of objectives

**Type***int***n\_stochastic\_constraints**

number of stochastic constraints

**Type***int***minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type***tuple of int (+/- 1)***constraint\_type****description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type***str*

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

str

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

float

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

base.Model

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

[list] [mrg32k3a.mrg32k3a.MRG32k3a]

**factors**

changeable factors of the problem

**initial\_solution**

[tuple] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name of problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

bool

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints



- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dict*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dict*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dict*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

*dict*

```
class simopt.models.ironore.IronOreMaxRevCnt (name='IRONORECONT-1', fixed_factors=None,  
                                              model_fixed_factors=None)
```

Bases: *Problem*

Class to make iron ore inventory simulation-optimization problems.

**name**

name of problem

**Type**

*str*

**dim**

number of decision variables

**Type**

*int*

**n\_objectives**

number of objectives

**Type**

*int*

**n\_stochastic\_constraints**

number of stochastic constraints

**Type**

*int*

**minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type**

*tuple of int (+/- 1)*

**constraint\_type**

**description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

*str*

**variable\_type**

**description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

*str*

**gradient\_available**

indicates if gradient of objective function is available

**Type**

*bool*

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

base.Model

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

[list] [mrg32k3a.mrg32k3a.MRG32k3a]

**factors****changeable factors of the problem****initial\_solution**

[tuple] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name of problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

*bool*

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dict*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrng32k3a*, *mrng32k3a*, *MRG32k3a*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dict*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dict*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

*dict*

## simopt.models.mm1queue module

### Summary

Simulate a M/M/1 queue. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.mm1queue.MM1MinMeanSojournTime` (*name='MM1-1'*, *fixed\_factors=None*, *model\_fixed\_factors=None*)

Bases: *Problem*

Base class to implement simulation-optimization problems.

**name**  
name of problem  
**Type**  
string

**dim**  
number of decision variables  
**Type**  
int

**n\_objectives**  
number of objectives  
**Type**  
int

**n\_stochastic\_constraints**  
number of stochastic constraints  
**Type**  
int

**minmax**  
indicator of maximization (+1) or minimization (-1) for each objective  
**Type**  
tuple of int (+/- 1)

**constraint\_type**  
**description of constraints types:**  
“unconstrained”, “box”, “deterministic”, “stochastic”  
**Type**  
string

**variable\_type**  
**description of variable types:**  
“discrete”, “continuous”, “mixed”  
**Type**  
string

**lower\_bounds**  
lower bound for each decision variable  
**Type**  
tuple

**upper\_bounds**  
upper bound for each decision variable  
**Type**  
tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

list of mrg32k3a.mrg32k3a.MRG32k3a objects

**factors**

changeable factors of the problem

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name for problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **model\_fixed\_factors** (*dict*) – subset of user-specified non-decision factors to pass through to the model

See also:

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

`bool`

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

`tuple`



**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

dictionary

**class** simopt.models.mm1queue.**MM1Queue** (*fixed\_factors=None*)

Bases: *Model*

A model that simulates an M/M/1 queue with an Exponential(lambda) interarrival time distribution and an Exponential(x) service time distribution. Returns

- the average sojourn time
- the average waiting time
- the fraction of customers who wait

for customers after a warmup period.

**name**  
name of model  
**Type**  
string

**n\_rngs**  
number of random-number generators used to run a simulation replication  
**Type**  
int

**n\_responses**  
number of responses (performance measures)  
**Type**  
int

**factors**  
changeable factors of the simulation model  
**Type**  
dict

**specifications**  
details of each factor (for GUI, data validation, and defaults)  
**Type**  
dict

**check\_factor\_list**  
switch case for checking factor simulatability  
**Type**  
dict

**Parameters**  
**fixed\_factors** (*nested dict*) – fixed factors of the simulation model

**See also:**

`base.Model`

`check_lambda()`

`check_mu()`

`check_people()`

`check_simulatable_factors()`

Determine if a simulation replication can be run with the given factors.

## Notes

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

### Returns

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

### Return type

`bool`

**check\_warmup** ()

**replicate** (*rng\_list*)

Simulate a single replication for the current model factors.

### Parameters

**rng\_list** (*list of mrg32k3a.mrg32k3a.MRG32k3a objects*) – rngs for model to use when simulating a replication

### Returns

- **responses** (*dict*) – performance measures of interest “avg\_sojourn\_time” = average sojourn time “avg\_waiting\_time” = average waiting time “frac\_cust\_wait” = fraction of customers who wait
- **gradients** (*dict of dicts*) – gradient estimates for each response

## simopt.models.network module

### Summary

Simulate messages being processed in a queueing network. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.network.Network` (*fixed\_factors=None*)

Bases: `Model`

Simulate messages being processed in a queueing network.

#### **name**

name of model

#### **Type**

`str`

#### **n\_rngs**

number of random-number generators used to run a simulation replication

#### **Type**

`int`

#### **n\_responses**

number of responses (performance measures)

#### **Type**

`int`

**factors**

changeable factors of the simulation model

**Type**

`dict`

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**check\_factor\_list**

switch case for checking factor simulatability

**Type**

`dict`

**Parameters**

**fixed\_factors** (`dict`) – fixed\_factors of the simulation model

**See also:**

`base.Model`

`check_arrival_rate()`

`check_cost_process()`

`check_cost_time()`

`check_lower_limits_transit_time()`

`check_mode_transit_time()`

`check_n_messages()`

`check_n_networks()`

`check_process_prob()`

`check_simulatable_factors()`

Determine if a simulation replication can be run with the given factors.

**Notes**

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

**Returns**

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

**Return type**

`bool`

`check_upper_limits_transit_time()`

**replicate** (*rng\_list*)

Simulate a single replication for the current model factors.

**Parameters**

**rng\_list** (*list of mrg32k3a.mrg32k3a.MRG32k3a objects*) – rngs for model to use when simulating a replication

**Returns**

- **responses** (*dict*) – performance measure of interest “total\_cost”: total cost spent to route all messages
- **gradients** (*dict of dicts*) – gradient estimates for each response

**class** simopt.models.network.**NetworkMinTotalCost** (*name='NETWORK-1', fixed\_factors=None, model\_fixed\_factors=None*)

Bases: *Problem*

Base class to implement simulation-optimization problems.

**name**

name of problem

**Type**

string

**dim**

number of decision variables

**Type**

int

**n\_objectives**

number of objectives

**Type**

int

**n\_stochastic\_constraints**

number of stochastic constraints

**Type**

int

**minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type**

tuple of int (+/- 1)

**constraint\_type**

**description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type**

**description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**lower\_bounds**

lower bound for each decision variable

**Type**

tuple

**upper\_bounds**

upper bound for each decision variable

**Type**

tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

list of `mrg32k3a.mrg32k3a.MRG32k3a` objects

**factors**

changeable factors of the problem

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name for problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **model\_fixed\_factors** (*dict*) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

bool

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values



**Return type**  
dictionary

## simopt.models.paramesti module

### Summary

Simulate MLE estimation for the parameters of a two-dimensional gamma distribution. A detailed description of the model/problem can be found [here](#).

```
class simopt.models.paramesti.ParamEstiMaxLogLik (name='PARAMESTI-1',
                                                fixed_factors=None,
                                                model_fixed_factors=None)
```

Bases: *Problem*

Base class to implement simulation-optimization problems.

**name**  
name of problem

**Type**  
string

**dim**  
number of decision variables

**Type**  
int

**n\_objectives**  
number of objectives

**Type**  
int

**n\_stochastic\_constraints**  
number of stochastic constraints

**Type**  
int

**minmax**  
indicator of maximization (+1) or minimization (-1) for each objective

**Type**  
tuple of int (+/- 1)

**constraint\_type**

**description of constraints types:**  
“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**  
string

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**lower\_bounds**

lower bound for each decision variable

**Type**

tuple

**upper\_bounds**

upper bound for each decision variable

**Type**

tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

list of `mrg32k3a.mrg32k3a.MRG32k3a` objects

**factors****changeable factors of the problem****initial\_solution**

[list] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**prev\_cost**

[list] cost of prevention

**upper\_thres**

[float > 0] upper limit of amount of contamination

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name for problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

bool

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrng32k3a.mrng32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

dictionary

**class** `simopt.models.paramesti.ParameterEstimation` (*fixed\_factors=None*)

Bases: *Model*

A model that simulates MLE estimation for the parameters of a two-dimensional gamma distribution.

**name**

name of model

**Type**

string

**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**

`int`

**n\_responses**

number of responses (performance measures)

**Type**

`int`

**factors**

changeable factors of the simulation model

**Type**

`dict`

**specifications**

details of each factor (for GUI and data validation)

**Type**

`dict`

**check\_factor\_list**

switch case for checking factor simulatability

**Type**

`dict`

**Parameters**

**fixed\_factors** (*nested dict*) – fixed factors of the simulation model

**See also:**

`base.model`

**check\_simulatable\_factors()**

Determine if a simulation replication can be run with the given factors.

**Notes**

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

**Returns**

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

**Return type**

`bool`

**check\_x()****check\_xstar()****replicate(rng\_list)**

Simulate a single replication for the current model factors.

**Parameters**

**rng\_list** (*list of `mrg32k3a.mrg32k3a.MRG32k3a` objects*) – rngs for model to use when simulating a replication

**Returns**

- **responses** (*dict*) – performance measures of interest “loglik” = the corresponding loglikelihood
- **gradients** (*dict of dicts*) – gradient estimates for each response

**simopt.models.rmitd module****Summary**

Simulate a multi-stage revenue management system with inter-temporal dependence. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.rmitd.RMITD` (*fixed\_factors=None*)

Bases: [Model](#)

A model that simulates a multi-stage revenue management system with inter-temporal dependence. Returns the total revenue.

**name**

name of model

**Type**

string

**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**

[int](#)

**n\_responses**

number of responses (performance measures)

**Type**

[int](#)

**factors**

changeable factors of the simulation model

**Type**

[dict](#)

**specifications**

details of each factor (for GUI and data validation)

**Type**

[dict](#)

**check\_factor\_list**

switch case for checking factor simulatability

**Type**

[dict](#)

**Parameters**

**fixed\_factors** (*nested dict*) – fixed factors of the simulation model

**See also:**`base.Model``check_cost()``check_demand_means()``check_gamma_scale()``check_gamma_shape()``check_initial_inventory()``check_prices()``check_reservation_qtys()``check_simulatable_factors()`

Determine if a simulation replication can be run with the given factors.

**Notes**

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

**Returns**

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

**Return type**

`bool`

`check_time_horizon()``replicate(rng_list)`

Simulate a single replication for the current model factors.

**Parameters**

**rng\_list** (*list of `mrg32k3a.mrg32k3a.MRG32k3a` objects*) – rngs for model to use when simulating a replication

**Returns**

- **responses** (*dict*) – performance measures of interest “revenue” = total revenue
- **gradients** (*dict of dicts*) – gradient estimates for each response

```
class simopt.models.rmitd.RMITDMaxRevenue (name='RMITD-1', fixed_factors=None,  
                                           model_fixed_factors=None)
```

Bases: `Problem`

Base class to implement simulation-optimization problems.

**name**

name of problem

**Type**

string

**dim**  
number of decision variables  
**Type**  
int

**n\_objectives**  
number of objectives  
**Type**  
int

**n\_stochastic\_constraints**  
number of stochastic constraints  
**Type**  
int

**minmax**  
indicator of maximization (+1) or minimization (-1) for each objective  
**Type**  
tuple of int (+/- 1)

**constraint\_type**  
**description of constraints types:**  
“unconstrained”, “box”, “deterministic”, “stochastic”  
**Type**  
string

**variable\_type**  
**description of variable types:**  
“discrete”, “continuous”, “mixed”  
**Type**  
string

**lower\_bounds**  
lower bound for each decision variable  
**Type**  
tuple

**upper\_bounds**  
upper bound for each decision variable  
**Type**  
tuple

**gradient\_available**  
indicates if gradient of objective function is available  
**Type**  
bool



**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

list of mrg32k3a.mrg32k3a.MRG32k3a objects

**factors****changeable factors of the problem****initial\_solution**

[tuple] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name for problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **model\_fixed\_factors** (*dict*) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

*bool*

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

dictionary

## simopt.models.san module

### Summary

Simulate duration of a stochastic activity network (SAN). A detailed description of the model/problem can be found [here](#).

**class** simopt.models.san.**SAN** (*fixed\_factors=None*)

Bases: *Model*

A model that simulates a stochastic activity network problem with tasks that have exponentially distributed durations, and the selected means come with a cost.

**name**

name of model

**Type**

string

**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**

int

**n\_responses**

number of responses (performance measures)

**Type**

int

**factors**

changeable factors of the simulation model

**Type**

dict

**specifications**

details of each factor (for GUI and data validation)

**Type**

dict

**check\_factor\_list**

switch case for checking factor simulatability

**Type**  
`dict`

**Parameters**

**fixed\_factors** (*nested dict*) – fixed factors of the simulation model

**See also:**

`base.Model`

**check\_arc\_means** ()

**check\_arcs** ()

**check\_num\_nodes** ()

**dfs** (*graph, start, visited=None*)

**replicate** (*rng\_list*)

Simulate a single replication for the current model factors.

**Parameters**

**rng\_list** (*list of mrg32k3a.mrg32k3a.MRG32k3a*) – rngs for model to use when simulating a replication

**Returns**

- **responses** (*dict*) – performance measures of interest “longest\_path\_length” = length/duration of longest path
- **gradients** (*dict of dicts*) – gradient estimates for each response

**class** `simopt.models.san.SANLongestPath` (*name='SAN-1', fixed\_factors=None, model\_fixed\_factors=None*)

Bases: `Problem`

Base class to implement simulation-optimization problems.

**name**

name of problem

**Type**  
`string`

**dim**

number of decision variables

**Type**  
`int`

**n\_objectives**

number of objectives

**Type**  
`int`

**n\_stochastic\_constraints**

number of stochastic constraints

**Type**  
`int`

**minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type**

tuple of int (+/- 1)

**constraint\_type****description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**lower\_bounds**

lower bound for each decision variable

**Type**

tuple

**upper\_bounds**

upper bound for each decision variable

**Type**

tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

Model object

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

`dict`

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

`dict`

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

`set of str`

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

`list of mrg32k3a.mrg32k3a.MRG32k3a objects`

**factors****changeable factors of the problem****initial\_solution**

[list] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**Type**

`dict`

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**Parameters**

- **name** (`str`) – user-specified name for problem
- **fixed\_factors** (`dict`) – dictionary of user-specified problem factors
- **factors** (`model_fixed`) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

`check_arc_costs()`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

*bool*

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dictionary*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a object*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dictionary*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

dictionary

## simopt.models.sscont module

### Summary

Simulate multiple periods worth of sales for a (s,S) inventory problem with continuous inventory. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.sscont.SSCont` (*fixed\_factors=None*)

Bases: *Model*

A model that simulates multiple periods' worth of sales for a (s,S) inventory problem with continuous inventory, exponentially distributed demand, and poisson distributed lead time. Returns the various types of average costs per period, order rate, stockout rate, fraction of demand met with inventory on hand, average amount backordered given a stockout occurred, and average amount ordered given an order occurred.

**name**

name of model

**Type**

*str*



**n\_rngs**

number of random-number generators used to run a simulation replication

Type

*int*

**n\_responses**

number of responses (performance measures)

Type

*int*

**factors**

changeable factors of the simulation model

Type

*dict*

**specifications**

details of each factor (for GUI, data validation, and defaults)

Type

*dict*

**check\_factor\_list**

switch case for checking factor simulatability

Type

*dict*

**Parameters**

**fixed\_factors** (*dict*) – fixed\_factors of the simulation model

**demand\_mean**

Mean of exponentially distributed demand in each period (*flt*)

**lead\_mean**

Mean of Poisson distributed order lead time (*flt*)

**backorder\_cost**

Cost per unit of demand not met with in-stock inventory (*flt*)

**holding\_cost**

Holding cost per unit per period (*flt*)

**fixed\_cost**

Order fixed cost (*flt*)

**variable\_cost**

Order variable cost per unit (*flt*)

**s**

Inventory position threshold for placing order (*flt*)

**S**

Max inventory position (*flt*)

**n\_days**

Number of periods to simulate (*int*)

**warmup**

Number of periods as warmup before collecting statistics (*int*)

See also:

`base.Model`

`check_S()`

`check_backorder_cost()`

`check_demand_mean()`

`check_fixed_cost()`

`check_holding_cost()`

`check_lead_mean()`

`check_n_days()`

`check_s()`

`check_simulatable_factors()`

Determine if a simulation replication can be run with the given factors.

## Notes

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

### Returns

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

### Return type

`bool`

`check_variable_cost()`

`check_warmup()`

`replicate(rng_list)`

Simulate a single replication for the current model factors.

### Parameters

**rng\_list** (*[list]* `[mrg32k3a.mrg32k3a.MRG32k3a]`) – rngs for model to use when simulating a replication

### Returns

**responses** – performance measures of interest

**avg\_backorder\_costs**  
average backorder costs per period

**avg\_order\_costs**  
average order costs per period

**avg\_holding\_costs**  
average holding costs per period

**on\_time\_rate**  
fraction of demand met with stock on hand in store

**order\_rate**  
fraction of periods an order was made

**stockout\_rate**  
fraction of periods a stockout occurred

**avg\_stockout**  
mean amount of product backordered given a stockout occurred

**avg\_order**  
mean amount of product ordered given an order occurred

**Return type**  
`dict`

**class** `simopt.models.sscont.SSContMinCost` (*name='SSCONT-1', fixed\_factors=None, model\_fixed\_factors=None*)

Bases: `Problem`

Class to make (s,S) inventory simulation-optimization problems.

**name**  
name of problem

**Type**  
`str`

**dim**  
number of decision variables

**Type**  
`int`

**n\_objectives**  
number of objectives

**Type**  
`int`

**n\_stochastic\_constraints**  
number of stochastic constraints

**Type**  
`int`

**minmax**  
indicator of maximization (+1) or minimization (-1) for each objective

**Type**  
`tuple of int (+/- 1)`

**constraint\_type**

**description of constraints types:**  
“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**  
`str`

**variable\_type**

**description of variable types:**  
“discrete”, “continuous”, “mixed”

**Type**

str

**lower\_bounds**

lower bound for each decision variable

**Type**

tuple

**upper\_bounds**

upper bound for each decision variable

**Type**

tuple

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

base.Model

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**`[list] [mrg32k3a.mrg32k3a.MRG32k3a]`**factors****changeable factors of the problem****initial\_solution**`[tuple]` default initial solution from which solvers start**budget**`[int > 0]` max number of replications (fn evals) for a solver to take**Type**`dict`**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**`dict`**Parameters**

- **name** (`str`) – user-specified name of problem
- **fixed\_factors** (`dict`) – dictionary of user-specified problem factors
- **factors** (`model_fixed`) – subset of user-specified non-decision factors to pass through to the model

**See also:**`base.Problem`**check\_deterministic\_constraints** (`x`)Check if a solution `x` satisfies the problem's deterministic constraints.**Parameters****x** (`tuple`) – vector of decision variables**Returns****satisfies** – indicates if solution `x` satisfies the deterministic constraints.**Return type**`bool`**deterministic\_objectives\_and\_gradients** (`x`)Compute deterministic components of objectives for a solution `x`.**Parameters****x** (`tuple`) – vector of decision variables**Returns**

- **det\_objectives** (`tuple`) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (`tuple`) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters**

**factor\_dict** (*dict*) – dictionary with factor keys and associated values

**Returns**

**vector** – vector of values associated with decision variables

**Return type**

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters**

**rand\_sol\_rng** (*mrng32k3a.mrng32k3a.MRG32k3a*) – random-number generator used to sample a new random solution

**Returns**

**x** – vector of decision variables

**Return type**

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters**

**response\_dict** (*dict*) – dictionary with response keys and associated values

**Returns**

**objectives** – vector of objectives

**Return type**

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$

**Parameters**

**response\_dict** (*dict*) – dictionary with response keys and associated values

**Returns**

**stoch\_constraints** – vector of LHSs of stochastic constraint

**Return type**

*tuple*

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters**

**vector** (*tuple*) – vector of values associated with decision variables

**Returns**

**factor\_dict** – dictionary with factor keys and associated values

**Return type**

*dict*

## simopt.models.tableallocation module

### Summary

Simulate multiple periods of arrival and seating at a restaurant. A detailed description of the model/problem can be found [here](#).

**class** `simopt.models.tableallocation.TableAllocation` (*fixed\_factors=None*)

Bases: *Model*

A model that simulates a table capacity allocation problem at a restaurant with a homogenous Poisson arrival process and exponential service times. Returns expected maximum revenue.

**name**

name of model

**Type**

*str*

**n\_rngs**

number of random-number generators used to run a simulation replication

**Type**

*int*

**n\_responses**

number of responses (performance measures)

**Type**

*int*

**factors**

changeable factors of the simulation model

**Type**

*dict*

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

*dict*

**check\_factor\_list**

switch case for checking factor simulatability

**Type**`dict`**Parameters****fixed\_factors** (*dict*) – fixed\_factors of the simulation model**n\_hours**Number of hours to simulate (*int*)**capacity**Maximum total capacity (*int*)**table\_cap**Capacity of each type of table (*int*)**lambda**Average number of arrivals per hour (*flt*)**service\_time\_means**Mean service time in minutes (*flt*)**table\_revenue**Per table revenue earned (*flt*)**num\_tables**Number of tables of each capacity (*int*)**See also:**`base.Model``check_capacity()``check_lambda()``check_n_hours()``check_num_tables()``check_service_time_means()``check_simulatable_factors()`

Determine if a simulation replication can be run with the given factors.

**Notes**Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.**Returns****is\_simulatable** – True if model specified by factors is simulatable, otherwise False.**Return type**`bool``check_table_cap()``check_table_revenue()`



**replicate** (*rng\_list*)

Simulate a single replication for the current model factors.

**Parameters**

**rng\_list** (*[list]* [*mrg32k3a.mrg32k3a.MRG32k3a*]) – rngs for model to use when simulating a replication

**Returns**

**responses** – performance measures of interest

**total\_revenue**

Total revenue earned over the simulation period.

**service\_rate**

Fraction of customer arrivals that are seated.

**Return type**

*dict*

**class** `simopt.models.tableallocation.TableAllocationMaxRev` (*name='TABLEALLOCATION-1', fixed\_factors=None, model\_fixed\_factors=None*)

Bases: *Problem*

Class to make table allocation simulation-optimization problems.

**name**

name of problem

**Type**

*str*

**dim**

number of decision variables

**Type**

*int*

**n\_objectives**

number of objectives

**Type**

*int*

**n\_stochastic\_constraints**

number of stochastic constraints

**Type**

*int*

**minmax**

indicator of maximization (+1) or minimization (-1) for each objective

**Type**

*tuple of int (+/- 1)*

**constraint\_type**

**description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

str

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

str

**gradient\_available**

indicates if gradient of objective function is available

**Type**

bool

**optimal\_value**

optimal objective function value

**Type**

tuple

**optimal\_solution**

optimal solution

**Type**

tuple

**model**

associated simulation model that generates replications

**Type**

base.Model

**model\_default\_factors**

default values for overriding model-level default factors

**Type**

dict

**model\_fixed\_factors**

combination of overridden model-level factors and defaults

**Type**

dict

**model\_decision\_factors**

set of keys for factors that are decision variables

**Type**

set of str

**rng\_list**

list of RNGs used to generate a random initial solution or a random problem instance

**Type**

[list] [mrg32k3a.mrg32k3a.MRG32k3a]

**factors****changeable factors of the problem****initial\_solution**

[tuple] default initial solution from which solvers start

**budget**

[int > 0] max number of replications (fn evals) for a solver to take

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**Parameters**

- **name** (*str*) – user-specified name of problem
- **fixed\_factors** (*dict*) – dictionary of user-specified problem factors
- **factors** (*model\_fixed*) – subset of user-specified non-decision factors to pass through to the model

**See also:**

`base.Problem`

**check\_deterministic\_constraints** (*x*)

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

**satisfies** – indicates if solution *x* satisfies the deterministic constraints.

**Return type**

bool

**deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_objectives** (*tuple*) – vector of deterministic components of objectives
- **det\_objectives\_gradients** (*tuple*) – vector of gradients of deterministic components of objectives

**deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

**Parameters**

**x** (*tuple*) – vector of decision variables

**Returns**

- **det\_stoch\_constraints** (*tuple*) – vector of deterministic components of stochastic constraints
- **det\_stoch\_constraints\_gradients** (*tuple*) – vector of gradients of deterministic components of stochastic constraints

**factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

**Parameters****factor\_dict** (*dict*) – dictionary with factor keys and associated values**Returns****vector** – vector of values associated with decision variables**Return type***tuple***get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

**Parameters****rand\_sol\_rng** (*mrg32k3a.mrg32k3a.MRG32k3a*) – random-number generator used to sample a new random solution**Returns****x** – vector of decision variables**Return type***tuple***response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

**Parameters****response\_dict** (*dict*) – dictionary with response keys and associated values**Returns****objectives** – vector of objectives**Return type***tuple***response\_dict\_to\_stoch\_constraints** (*response\_dict*)Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$ **Parameters****response\_dict** (*dict*) – dictionary with response keys and associated values**Returns****stoch\_constraints** – vector of LHSs of stochastic constraint**Return type***tuple***vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys

**Parameters****vector** (*tuple*) – vector of values associated with decision variables

**Returns****factor\_dict** – dictionary with factor keys and associated values**Return type**

dict

**Module contents****simopt.solvers package****Submodules****simopt.solvers.adam module****Summary**

ADAM An algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. A detailed description of the solver can be found [here](#).

**class** `simopt.solvers.adam.ADAM` (*name='ADAM', fixed\_factors=None*)

Bases: *Solver*

An algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments.

**name**

name of solver

**Type**

string

**objective\_type****description of objective types:**

“single” or “multi”

**Type**

string

**constraint\_type****description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**gradient\_needed**

indicates if gradient of objective function is needed

**Type**

`bool`

**factors**

changeable factors (i.e., parameters) of the solver

**Type**

`dict`

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**rng\_list**

list of RNGs used for the solver's internal purposes

**Type**

`list` of `mrg32k3a.mrg32k3a.MRG32k3a` objects

**Parameters**

- **name** (`str`) – user-specified name for solver
- **fixed\_factors** (`dict`) – fixed\_factors of the solver

**See also:**

`base.Solver`

`check_alpha()`

`check_beta_1()`

`check_beta_2()`

`check_epsilon()`

`check_r()`

`check_sensitivity()`

`finite_diff(new_solution, BdsCheck, problem)`

`solve(problem)`

Run a single macroreplication of a solver on a problem.

**Parameters**

- **problem** (*Problem object*) – simulation-optimization problem to solve
- **crn\_across\_solns** (`bool`) – indicates if CRN are used when simulating different solutions

**Returns**

- **recommended\_solns** (*list of Solution objects*) – list of solutions recommended throughout the budget

- **intermediate\_budgets** (*list of ints*) – list of intermediate budgets when recommended solutions changes

## simopt.solvers.aloe module

### Summary

ALOE The solver is a stochastic line search algorithm with the gradient estimate recomputed in each iteration, whether or not a step is accepted. The algorithm includes the relaxation of the Armijo condition by an additive constant. A detailed description of the solver can be found [here](#).

**class** `simopt.solvers.aloe.ALOE` (*name='ALOE', fixed\_factors=None*)

Bases: `Solver`

Adaptive Line-search with Oracle Estimations

#### **name**

name of solver

#### **Type**

string

#### **objective\_type**

##### **description of objective types:**

“single” or “multi”

#### **Type**

string

#### **constraint\_type**

##### **description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

#### **Type**

string

#### **variable\_type**

##### **description of variable types:**

“discrete”, “continuous”, “mixed”

#### **Type**

string

#### **gradient\_needed**

indicates if gradient of objective function is needed

#### **Type**

bool

#### **factors**

changeable factors (i.e., parameters) of the solver

#### **Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**rng\_list**

list of RNGs used for the solver's internal purposes

**Type**

`list` of `mrg32k3a.mrg32k3a.MRG32k3a` objects

**Parameters**

- **name** (`str`) – user-specified name for solver
- **fixed\_factors** (`dict`) – fixed\_factors of the solver

**See also:**

`base.Solver`

`check_alpha_0()`

`check_alpha_max()`

`check_epsilon_f()`

`check_gamma()`

`check_lambda()`

`check_r()`

`check_sensitivity()`

`check_theta()`

`finite_diff(new_solution, BdsCheck, problem, stepsize, r)`

`solve(problem)`

Run a single macroreplication of a solver on a problem.

**Parameters**

- **problem** (*Problem object*) – simulation-optimization problem to solve
- **crn\_across\_solns** (`bool`) – indicates if CRN are used when simulating different solutions

**Returns**

- **recommended\_solns** (*list of Solution objects*) – list of solutions recommended throughout the budget
- **intermediate\_budgets** (*list of ints*) – list of intermediate budgets when recommended solutions changes



## simopt.solvers.astrodf module

### Summary

The ASTRO-DF solver progressively builds local models (quadratic with diagonal Hessian) using interpolation on a set of points on the coordinate bases of the best (incumbent) solution. Solving the local models within a trust region (closed ball around the incumbent solution) at each iteration suggests a candidate solution for the next iteration. If the candidate solution is worse than the best interpolation point, it is replaced with the latter (a.k.a. direct search). The solver then decides whether to accept the candidate solution and expand the trust-region or reject it and shrink the trust-region based on a success ratio test. The sample size at each visited point is determined adaptively and based on closeness to optimality. A detailed description of the solver can be found [here](#). This version does not require a `delta_max`, instead it estimates the maximum step size using `get_random_solution()`. Parameter tuning on `delta_max` is therefore not needed and removed from this version as well.

```
class simopt.solvers.astrodf.ASTRODF (name='ASTRODF', fixed_factors=None)
```

Bases: *Solver*

The ASTRO-DF solver.

**name**

name of solver

**Type**

string

**objective\_type**

**description of objective types:**

“single” or “multi”

**Type**

string

**constraint\_type**

**description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type**

**description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**gradient\_needed**

indicates if gradient of objective function is needed

**Type**

bool

**factors**

changeable factors (i.e., parameters) of the solver

**Type**

*dict*

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

*dict*

**rng\_list**

list of RNGs used for the solver's internal purposes

**Type**

*list* of `mrg32k3a.mrg32k3a.MRG32k3a` objects

**Parameters**

- **name** (*str*) – user-specified name for solver
- **fixed\_factors** (*dict*) – fixed\_factors of the solver

See also:

`base.Solver`

`check_eta_1()`

`check_eta_2()`

`check_gamma_1()`

`check_gamma_2()`

`check_lambda_min()`

`construct_model(x_k, delta, k, problem, expended_budget, kappa, new_solution, visited_pts_list)`

`evaluate_model(x_k, q)`

`get_coordinate_basis_interpolation_points(x_k, delta, problem)`

`get_coordinate_vector(size, v_no)`

`get_model_coefficients(Y, fval, problem)`

`get_rotated_basis(first_basis, rotate_index)`

`get_rotated_basis_interpolation_points(x_k, delta, problem, rotate_matrix, reused_x)`

`get_stopping_time(k, sig2, delta, kappa, dim)`

`iterate(k, delta_k, delta_max, problem, visited_pts_list, new_x, expended_budget, budget_limit, recommended_solns, intermediate_budgets, kappa, new_solution)`

**solve** (*problem*)

Run a single macroreplication of a solver on a problem. :param problem: simulation-optimization problem to solve :type problem: Problem object :param crn\_across\_solns: indicates if CRN are used when simulating different solutions :type crn\_across\_solns: bool

**Returns**

- **recommended\_solns** (*list of Solution objects*) – list of solutions recommended throughout the budget
- **intermediate\_budgets** (*list of ints*) – list of intermediate budgets when recommended solutions changes

**simopt.solvers.neldmd module****Summary**

Nelder-Mead: An algorithm that maintains a simplex of points that moves around the feasible region according to certain geometric operations: reflection, expansion, contraction, and shrinking. A detailed description of the solver can be found [here](#).

**class** `simopt.solvers.neldmd.NelderMead` (*name='NELDMD', fixed\_factors=None*)

Bases: `Solver`

The Nelder-Mead algorithm, which maintains a simplex of points that moves around the feasible region according to certain geometric operations: reflection, expansion, contraction, and shrinking.

**name**

name of solver

**Type**

string

**objective\_type****description of objective types:**

“single” or “multi”

**Type**

string

**constraint\_type****description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**gradient\_needed**

indicates if gradient of objective function is needed

**Type**

`bool`

**factors**

changeable factors (i.e., parameters) of the solver

**Type**

`dict`

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**rng\_list**

list of RNGs used for the solver's internal purposes

**Type**

`list` of `mrg32k3a.mrg32k3a.MRG32k3a` objects

**Parameters**

- **name** (`str`) – user-specified name for solver
- **fixed\_factors** (`dict`) – fixed\_factors of the solver

**See also:**

`base.Solver`

`check_alpha()`

`check_betap()`

`check_const(pt, pt2)`

`check_delta()`

`check_gammap()`

`check_initial_spread()`

`check_r()`

`check_sensitivity()`

`solve(problem)`

Run a single macroreplication of a solver on a problem.

**Parameters**

**problem** (*Problem object*) – simulation-optimization problem to solve

**Returns**

- **recommended\_solns** (*list of Solution objects*) – list of solutions recommended throughout the budget
- **intermediate\_budgets** (*list of ints*) – list of intermediate budgets when recommended solutions changes

`sort_and_end_update` (*problem, sol*)

## simopt.solvers.randomsearch module

### Summary

Randomly sample solutions from the feasible region. Can handle stochastic constraints. A detailed description of the solver can be found [here](#).

**class** `simopt.solvers.randomsearch.RandomSearch` (*name='RNDSEARCH', fixed\_factors=None*)

Bases: `Solver`

A solver that randomly samples solutions from the feasible region. Take a fixed number of replications at each solution.

#### **name**

name of solver

#### **Type**

string

#### **objective\_type**

##### **description of objective types:**

“single” or “multi”

#### **Type**

string

#### **constraint\_type**

##### **description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

#### **Type**

string

#### **variable\_type**

##### **description of variable types:**

“discrete”, “continuous”, “mixed”

#### **Type**

string

#### **gradient\_needed**

indicates if gradient of objective function is needed

#### **Type**

bool

#### **factors**

changeable factors (i.e., parameters) of the solver

#### **Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**rng\_list**

list of RNGs used for the solver's internal purposes

**Type**

list of `mrg32k3a.mrg32k3a.MRG32k3a` objects

**Parameters**

- **name** (*str*) – user-specified name for solver
- **fixed\_factors** (*dict*) – fixed\_factors of the solver

**See also:**

`base.Solver`

`check_sample_size()`

`solve(problem)`

Run a single macroreplication of a solver on a problem.

**Parameters**

- **problem** (*Problem object*) – simulation-optimization problem to solve
- **crn\_across\_solns** (*bool*) – indicates if CRN are used when simulating different solutions

**Returns**

- **recommended\_solns** (*list of Solution objects*) – list of solutions recommended throughout the budget
- **intermediate\_budgets** (*list of ints*) – list of intermediate budgets when recommended solutions changes

**simopt.solvers.spsa module****Summary**

Simultaneous perturbation stochastic approximation (SPSA) is an algorithm for optimizing systems with multiple unknown parameters.

**class** `simopt.solvers.spsa.SPSA` (*name='SPSA', fixed\_factors=None*)

Bases: `Solver`

Simultaneous perturbation stochastic approximation (SPSA) is an algorithm for optimizing systems with multiple unknown parameters.

**name**

name of solver

**Type**

string

**objective\_type****description of objective types:**

“single” or “multi”

**Type**

string

**constraint\_type****description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

**Type**

string

**variable\_type****description of variable types:**

“discrete”, “continuous”, “mixed”

**Type**

string

**gradient\_needed**

indicates if gradient of objective function is needed

**Type**

bool

**factors**

changeable factors (i.e., parameters) of the solver

**Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

dict

**rng\_list**

list of RNGs used for the solver’s internal purposes

**Type**

list of `mrg32k3a.mrg32k3a.MRG32k3a` objects

**Parameters**

- **name** (*str*) – user-specified name for solver
- **fixed\_factors** (*dict*) – fixed\_factors of the solver

**See also:**

`base.Solver`

`check_alpha()`

`check_eval_pct()`

`check_gamma()`

`check_gavg()`

`check_iter_pct()`

`check_n_loss()`

`check_n_reps()`

`check_problem_factors()`

`check_step()`

`gen_simul_pert_vec(dim)`

Generate a new simultaneous perturbation vector with a 50/50 probability discrete distribution, with values of -1 and 1. The vector size is the problem's dimension. The vector components are independent from each other.

**Parameters**

**dim** (*int*) – Length of the vector.

**Returns**

Vector of -1's and 1's.

**Return type**

*list*

`solve(problem)`

Run a single macroreplication of a solver on a problem.

**Parameters**

- **problem** (*Problem object*) – simulation-optimization problem to solve
- **crn\_across\_solns** (*bool*) – indicates if CRN are used when simulating different solutions

**Returns**

- **recommended\_solns** (*list of Solution objects*) – list of solutions recommended throughout the budget
- **intermediate\_budgets** (*list of ints*) – list of intermediate budgets when recommended solutions changes

`simopt.solvers.spsa.check_cons(candidate_x, new_x, lower_bound, upper_bound)`

Evaluates the distance from the new vector (candidate\_x) compared to the current vector (new\_x) respecting the vector's boundaries of feasibility. Returns the evaluated vector (modified\_x) and the weight (t2 - how much of a full step took) of the new vector. The weight (t2) is used to calculate the weighted average in the ftheta calculation.



## simopt.solvers.strong module

### Summary

**STRONG**: A trust-region-based algorithm that fits first- or second-order models through function evaluations taken within a neighborhood of the incumbent solution. A detailed description of the solver can be found [here](#).

**class** `simopt.solvers.strong.STRONG` (*name*='STRONG', *fixed\_factors*=None)

Bases: `Solver`

A trust-region-based algorithm that fits first- or second-order models through function evaluations taken within a neighborhood of the incumbent solution.

#### **name**

name of solver

#### **Type**

string

#### **objective\_type**

##### **description of objective types:**

“single” or “multi”

#### **Type**

string

#### **constraint\_type**

##### **description of constraints types:**

“unconstrained”, “box”, “deterministic”, “stochastic”

#### **Type**

string

#### **variable\_type**

##### **description of variable types:**

“discrete”, “continuous”, “mixed”

#### **Type**

string

#### **gradient\_needed**

indicates if gradient of objective function is needed

#### **Type**

bool

#### **factors**

changeable factors (i.e., parameters) of the solver

#### **Type**

dict

**specifications**

details of each factor (for GUI, data validation, and defaults)

**Type**

`dict`

**rng\_list**

list of RNGs used for the solver's internal purposes

**Type**

list of `mrg32k3a.mrg32k3a.MRG32k3a` objects

**Parameters**

- **name** (`str`) – user-specified name for solver
- **fixed\_factors** (`dict`) – fixed\_factors of the solver

**See also:**

`base.Solver`

**cauchy\_point** (*grad, Hessian, new\_x, problem*)

**check\_cons** (*candidate\_x, new\_x, lower\_bound, upper\_bound*)

**check\_delta\_T** ()

**check\_delta\_threshold** ()

**check\_eta\_0** ()

**check\_eta\_1** ()

**check\_gamma\_1** ()

**check\_gamma\_2** ()

**check\_lambda** ()

**check\_n\_r** ()

**check\_sensitivity** ()

**finite\_diff** (*new\_solution, BdsCheck, stage, problem, n\_r*)

**solve** (*problem*)

Run a single macroreplication of a solver on a problem.

**Parameters**

- **problem** (*Problem object*) – simulation-optimization problem to solve
- **crn\_across\_solns** (`bool`) – indicates if CRN are used when simulating different solutions

**Returns**

- **recommended\_solns** (*list of Solution objects*) – list of solutions recommended throughout the budget
- **intermediate\_budgets** (*list of ints*) – list of intermediate budgets when recommended solutions changes

## Module contents

### 2.1.1.2 Submodules

#### 2.1.1.3 simopt.GUI module

**class** `simopt.GUI.Cross_Design_Window` (*master, main\_widow, forced\_creation=False*)

Bases: `object`

**confirm\_cross\_design\_function** ()

**get\_crossdesign\_MetaExperiment** ()

**test\_function** (\*args)

**class** `simopt.GUI.Experiment_Window` (*master*)

Bases: `Tk`

Main window of the GUI

**self.frame**

**Type**

Tkinter frame that contains the GUI widgets

**self.experiment\_master\_list**

**Type**

2D array list that contains queue of experiment object arguments

**self.widget\_list**

- this functionality is currently not enabled, possible constraint of the GUI framework

**Type**

Current method to clear, view/edit, and run individual experiments

**self.experiment\_object\_list**

**Type**

List that contains matching experiment objects to every sublist from `self.experiment_master_list`

**self.problem\_var**

**Type**

Variable that contains selected problem (use `.get()` method to obtain value for)

**self.solver\_var**

**Type**

Variable that contains selected solver (use `.get()` method to obtain value for)

**self.maco\_var**

**Type**

Variable that contains inputted number of macroreplications (use `.get()` method to obtain value for)

**Functions**

-----

**show\_problem\_factors** (*self*, \\*args)

connected to : self.problem\_menu &lt;- ttk.OptionMenu

**Type**

displays additional information on problem and oracle factors

**show\_solver\_factors** (*self*, \\*args)

connected to : self.solver\_menu &lt;- ttk.OptionMenu

**Type**

displays additional information on solver factors

**run\_single\_function** (*self*, \\*args)

connected to : self.run\_button &lt;- ttk.Button

**Type**

completes single-object experiment and invokes Post\_Processing\_Window class

**crossdesign\_function** (*self*)

connected to : self.crossdesign\_button &lt;- ttk.Button

**Type**

invokes Cross\_Design\_Window class

**clearRow\_function** (*self*)

connected to : self.clear\_button\_added &lt;- ttk.Button, within self.add\_experiment

**Type**

~not functional~ meant to clear a single row of the experiment queue

**clear\_queue** (*self*)

connected to : self.clear\_queue\_button &lt;- ttk.Button

**Type**

clears entire experiment queue and resets all lists containing experiment data

**add\_experiment** (*self*)

connected to : self.add\_button &lt;- ttk.Button

**Type**

adds experiment to experiment queue

**confirm\_problem\_factors** (*self*)

return : problem\_factors\_return | type = list | contains = [problem factor dictionary, None or problem rename]

**Type**

used within run\_single\_function, stores all problem factors in a dictionary

**confirm\_oracle\_factors** (*self*)

return : oracle\_factors\_return | type = list | contains = [oracle factor dictionary]

**Type**

used within run\_single\_function, stores all oracle factors in a dictionary

**confirm\_solver\_factors** (*self*)

return : solver\_factors\_return | type = list | contains = [solver factor dictionary, None or solver rename]

**Type**

used within `run_single_function`, stores all solver factors in a dictionary

**onFrameConfigure\_queue** (*self*, *event*)

**Type**

creates scrollbar for the queue notebook

**onFrameConfigure\_factor\_problem** (*self*, *event*)

**Type**

creates scrollbar for the problem factors notebook

**onFrameConfigure\_factor\_solver** (*self*, *event*)

**Type**

creates scrollbar for the solver factors notebook

**onFrameConfigure\_factor\_oracle** (*self*, *event*)

**Type**

creates scrollbar for the oracle factor notebook

**test\_function** (*self*, *\\*args*)

**Type**

placeholder function to make sure buttons, OptionMenus, etc are connected properly

**add\_experiment** (*\\*args*)

**add\_meta\_exp\_to\_frame** (*n\_macroreps=None*, *input\_meta\_experiment=None*)

**checkbox\_function2** (*exp*, *rowNum*)

**clearRow\_function** (*integer*)

**clear\_meta\_function** (*integer*)

**clear\_queue** ()

**confirm\_oracle\_factors** ()

**confirm\_problem\_factors** ()

**confirm\_solver\_factors** ()

**crossdesign\_function** ()

**exit\_meta\_view** (*row\_num*)

**load\_pickle\_file\_function** ()

**make\_meta\_experiment\_func** ()

**meta\_experiment\_problem\_solver\_list** (*metaExperiment*)

**onFrameConfigure\_factor\_oracle** (*event*)

**onFrameConfigure\_factor\_problem** (*event*)

**onFrameConfigure\_factor\_solver** (*event*)

```
onFrameConfigure_queue (event)

plot_meta_function (integer)

post_norm_return_func ()

post_norm_setup ()

post_normal_all_function ()

post_process_disable_button (meta=False)

post_rep_function (integer)

post_rep_meta_function (integer)

progress_bar_test ()

run_meta_function (integer)

run_row_function (integer)

save_edit_function (integer)

select_pickle_file_fuction (*args)

show_problem_factors (*args)

show_problem_factors2 (row_index, *args)

show_solver_factors (*args)

show_solver_factors2 (row_index, *args)

update_problem_list_compatibility ()

viewEdit_function (integer)

view_meta_function (row_num)

class simopt.GUI.Plot_Window (master, main_window, experiment_list=None, metaList=None)
    Bases: object
    Plot Window Page of the GUI

    Parameters
        • master (tk.Tk) – Tkinter window created from Experiment_Window.run_single_function
        • myexperiment (object (Experiment)) – Experiment object created in Experiment_Window.run_single_function
        • experiment_list (list) – List of experiment object arguments

    add_plot ()

    changeOnHover (button, colorOnHover, colorOnLeave)

    clear_row (place)

    get_parameters_and_settings (a, plot_choice)
```

```

plot_button()

solver_select_function(a)

view_one_pot(path_name)

class simopt.GUI.Post_Normal_Window(master, experiment_list, main_window, meta=False)
    Bases: object
    Post-Normalization Page of the GUI

        Parameters
            • master (tk.Tk) – Tkinter window created from Experiment_Window.run_single_function
            • myexperiment (object(Experiment)) – Experiment object created in Experiment_Window.run_single_function
            • experiment_list (list) – List of experiment object arguments

    post_norm_run_function()

    test_function2(*args)

class simopt.GUI.Post_Processing_Window(master, myexperiment, experiment_list, main_window,
                                         meta=False)
    Bases: object
    Postprocessing Page of the GUI

        Parameters
            • master (tk.Tk) – Tkinter window created from Experiment_Window.run_single_function
            • myexperiment (object(Experiment)) – Experiment object created in Experiment_Window.run_single_function
            • experiment_list (list) – List of experiment object arguments

    post_processing_run_function()

    test_function2(*args)

simopt.GUI.main()

simopt.GUI.problem_solver_abbreviated_name_to_unabbreviated(problem_or_solver,
                                                             abbreviated_dictionary,
                                                             unabbreviated_dictionary)

simopt.GUI.problem_solver_unabbreviated_to_object(problem_or_solver,
                                                  unabbreviated_dictionary)

```

#### 2.1.1.4 simopt.base module

##### Summary

Provide base classes for solvers, problems, and models.

**class** `simopt.base.Model` (*fixed\_factors*)

Bases: `object`

Base class to implement simulation models (models) featured in simulation-optimization problems.

**name**

Name of model.

**Type**

`str`

**n\_rngs**

Number of random-number generators used to run a simulation replication.

**Type**

`int`

**n\_responses**

Number of responses (performance measures).

**Type**

`int`

**factors**

Changeable factors of the simulation model.

**Type**

`dict`

**specifications**

Details of each factor (for GUI, data validation, and defaults).

**Type**

`dict`

**check\_factor\_list**

Switch case for checking factor simulatability.

**Type**

`dict`

**Parameters**

**fixed\_factors** (*dict*) – Dictionary of user-specified model factors.

**check\_factor\_datatype** (*factor\_name*)

Determine if a factor's data type matches its specification.

**Returns**

**is\_right\_type** – True if factor is of specified data type, otherwise False.

**Return type**

`bool`



**check\_simulatable\_factor** (*factor\_name*)

Determine if a simulation replication can be run with the given factor.

**Parameters**

**factor\_name** (*str*) – Name of factor for dictionary lookup (i.e., key).

**Returns**

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

**Return type**

*bool*

**check\_simulatable\_factors** ()

Determine if a simulation replication can be run with the given factors.

## Notes

Each subclass of `base.Model` has its own custom `check_simulatable_factors` method.

**Returns**

**is\_simulatable** – True if model specified by factors is simulatable, otherwise False.

**Return type**

*bool*

**replicate** (*rng\_list*)

Simulate a single replication for the current model factors.

**Parameters**

**rng\_list** (list [*mrg32k3a.mrg32k3a.MRG32k3a*]) – RNGs for model to use when simulating a replication.

**Returns**

- **responses** (*dict*) – Performance measures of interest.
- **gradients** (*dict [dict]*) – Gradient estimate for each response.

**class** `simopt.base.Problem` (*fixed\_factors*, *model\_fixed\_factors*)

Bases: `object`

Base class to implement simulation-optimization problems.

**name**

Name of problem.

**Type**

*str*

**dim**

Number of decision variables.

**Type**

*int*

**n\_objectives**

Number of objectives.

**Type**

*int*

**n\_stochastic\_constraints**

Number of stochastic constraints.

**Type**

int

**minmax**

Indicators of maximization (+1) or minimization (-1) for each objective.

**Type**

tuple [int]

**constraint\_type**

Description of constraints types: “unconstrained”, “box”, “deterministic”, “stochastic”.

**Type**

str

**variable\_type**

Description of variable types: “discrete”, “continuous”, “mixed”.

**Type**

str

**lower\_bounds**

Lower bound for each decision variable.

**Type**

tuple

**upper\_bounds**

Upper bound for each decision variable.

**Type**

tuple

**gradient\_available**

True if direct gradient of objective function is available, otherwise False.

**Type**

bool

**optimal\_value**

Optimal objective function value.

**Type**

float

**optimal\_solution**

Optimal solution.

**Type**

tuple

**model**

Associated simulation model that generates replications.

**Type**

base.Model

**model\_default\_factors**

Default values for overriding model-level default factors.

Type  
`dict`

**model\_fixed\_factors**

Combination of overridden model-level factors and defaults.

Type  
`dict`

**model\_decision\_factors**

Set of keys for factors that are decision variables.

Type  
`set [str]`

**rng\_list**

List of RNGs used to generate a random initial solution or a random problem instance.

Type  
`list [mrg32k3a.mrg32k3a.MRG32k3a]`

**factors**

**Changeable factors of the problem:**

**initial\_solution**  
`[tuple]` Default initial solution from which solvers start.

**budget**  
`[int]` Max number of replications (fn evals) for a solver to take.

Type  
`dict`

**specifications**

Details of each factor (for GUI, data validation, and defaults).

Type  
`dict`

**Parameters**

- **fixed\_factors** (`dict`) – Dictionary of user-specified problem factors.
- **model\_fixed\_factors** (`dict`) – Subset of user-specified non-decision factors to pass through to the model.

**attach\_rngs** (`rng_list`)

Attach a list of random-number generators to the problem.

**Parameters**

**rng\_list** (`list [mrg32k3a.mrg32k3a.MRG32k3a]`) – List of random-number generators used to generate a random initial solution or a random problem instance.

**check\_budget ()**

Check if budget is strictly positive.

**Returns**

True if budget is strictly positive, otherwise False.

**Return type**

bool

**check\_deterministic\_constraints (*x*)**

Check if a solution *x* satisfies the problem's deterministic constraints.

**Parameters**

**x** (*tuple*) – Vector of decision variables.

**Returns**

**satisfies** – True if solution *x* satisfies the deterministic constraints, otherwise False.

**Return type**

bool

**check\_factor\_datatype (*factor\_name*)**

Determine if a factor's data type matches its specification.

**Parameters**

**factor\_name** (*str*) – String corresponding to name of factor to check.

**Returns**

**is\_right\_type** – True if factor is of specified data type, otherwise False.

**Return type**

bool

**check\_initial\_solution ()**

Check if initial solution is feasible and of correct dimension.

**Returns**

True if initial solution is feasible and of correct dimension, otherwise False.

**Return type**

bool

**check\_problem\_factor (*factor\_name*)**

Determine if the setting of a problem factor is permissible.

**Parameters**

**factor\_name** (*str*) – Name of factor for dictionary lookup (i.e., key).

**Returns**

**is\_permissible** – True if problem factor is permissible, otherwise False.

**Return type**

bool

**check\_problem\_factors ()**

Determine if the joint settings of problem factors are permissible.

## Notes

Each subclass of `base.Problem` has its own custom `check_problem_factors` method.

### Returns

**is\_simulatable** – True if problem factors are permissible, otherwise False.

### Return type

`bool`

### **deterministic\_objectives\_and\_gradients** (*x*)

Compute deterministic components of objectives for a solution *x*.

### Parameters

**x** (*tuple*) – Vector of decision variables.

### Returns

- **det\_objectives** (*tuple*) – Vector of deterministic components of objectives.
- **det\_objectives\_gradients** (*tuple*) – Vector of gradients of deterministic components of objectives.

### **deterministic\_stochastic\_constraints\_and\_gradients** (*x*)

Compute deterministic components of stochastic constraints for a solution *x*.

### Parameters

**x** (*tuple*) – Vector of decision variables.

### Returns

- **det\_stoch\_constraints** (*tuple*) – Vector of deterministic components of stochastic constraints.
- **det\_stoch\_constraints\_gradients** (*tuple*) – Vector of gradients of deterministic components of stochastic constraints.

### **factor\_dict\_to\_vector** (*factor\_dict*)

Convert a dictionary with factor keys to a vector of variables.

## Notes

Each subclass of `base.Problem` has its own custom `factor_dict_to_vector` method.

### Parameters

**factor\_dict** (*dict*) – Dictionary with factor keys and associated values.

### Returns

**vector** – Vector of values associated with decision variables.

### Return type

`tuple`

### **factor\_dict\_to\_vector\_gradients** (*factor\_dict*)

Convert a dictionary with factor keys to a gradient vector.

## Notes

A subclass of `base.Problem` can have its own custom `factor_dict_to_vector_gradients` method if the objective is deterministic.

### Parameters

**factor\_dict** (*dict*) – Dictionary with factor keys and associated values.

### Returns

**vector** – Vector of partial derivatives associated with decision variables.

### Return type

*tuple*

**get\_random\_solution** (*rand\_sol\_rng*)

Generate a random solution for starting or restarting solvers.

### Parameters

**rand\_sol\_rng** (`mrg32k3a.mrg32k3a.MRG32k3a`) – Random-number generator used to sample a new random solution.

### Returns

**x** – vector of decision variables

### Return type

*tuple*

**response\_dict\_to\_objectives** (*response\_dict*)

Convert a dictionary with response keys to a vector of objectives.

## Notes

Each subclass of `base.Problem` has its own custom `response_dict_to_objectives` method.

### Parameters

**response\_dict** (*dict*) – Dictionary with response keys and associated values.

### Returns

**objectives** – Vector of objectives.

### Return type

*tuple*

**response\_dict\_to\_objectives\_gradients** (*response\_dict*)

Convert a dictionary with response keys to a vector of gradients.

## Notes

A subclass of `base.Problem` can have its own custom `response_dict_to_objectives_gradients` method if the objective is deterministic.

### Parameters

**response\_dict** (*dict*) – Dictionary with response keys and associated values.

### Returns

**vector** – Vector of gradients.

### Return type

*tuple*

**response\_dict\_to\_stoch\_constraints** (*response\_dict*)

Convert a dictionary with response keys to a vector of left-hand sides of stochastic constraints:  $E[Y] \leq 0$ .

### Notes

Each subclass of `base.Problem` has its own custom `response_dict_to_stoch_constraints` method.

#### Parameters

**response\_dict** (*dict*) – Dictionary with response keys and associated values.

#### Returns

**stoch\_constraints** – Vector of LHSs of stochastic constraints.

#### Return type

*tuple*

**simulate** (*solution, m=1*)

Simulate  $m$  i.i.d. replications at solution  $x$ .

### Notes

Gradients of objective function and stochastic constraint LHSs are temporarily commented out. Under development.

#### Parameters

- **solution** (*base.Solution*) – Solution to evaluate.
- **m** (*int*) – Number of replications to simulate at  $x$ .

**simulate\_up\_to** (*solutions, n\_reps*)

Simulate a set of solutions up to a given number of replications.

#### Parameters

- **solutions** (*set [base.Solution]*) – A set of `base.Solution` objects.
- **n\_reps** (*int*) – Common number of replications to simulate each solution up to.

**vector\_to\_factor\_dict** (*vector*)

Convert a vector of variables to a dictionary with factor keys.

### Notes

Each subclass of `base.Problem` has its own custom `vector_to_factor_dict` method.

#### Parameters

**vector** (*tuple*) – Vector of values associated with decision variables.

#### Returns

**factor\_dict** – Dictionary with factor keys and associated values.

#### Return type

*dict*

**class** `simopt.base.Solution` (*x*, *problem*)

Bases: `object`

Base class for solutions represented as vectors of decision variables and dictionaries of decision factors.

**x**

Vector of decision variables.

**Type**

`tuple`

**dim**

Number of decision variables describing *x*.

**Type**

`int`

**decision\_factors**

Decision factor names and values.

**Type**

`dict`

**rng\_list**

RNGs for model to use when running replications at the solution.

**Type**

`list [mrg32k3a.mrg32k3a.MRG32k3a]`

**n\_reps**

Number of replications run at the solution.

**Type**

`int`

**det\_objectives**

Deterministic components added to objectives.

**Type**

`tuple`

**det\_objectives\_gradients**

Gradients of deterministic components added to objectives; # objectives x dimension.

**Type**

`tuple [tuple]`

**det\_stoch\_constraints**

Deterministic components added to LHS of stochastic constraints.

**Type**

`tuple`

**det\_stoch\_constraints\_gradients**

Gradients of deterministic components added to LHS stochastic constraints; # stochastic constraints x dimension.

**Type**

`tuple [tuple]`



**storage\_size**

Max number of replications that can be recorded in current storage.

**Type**

`int`

**objectives**

Objective(s) estimates from each replication; # replications x # objectives.

**Type**

numpy array

**objectives\_gradients**

Gradient estimates of objective(s) from each replication; # replications x # objectives x dimension.

**Type**

numpy array

**stochastic\_constraints**

Stochastic constraint estimates from each replication; # replications x # stochastic constraints.

**Type**

numpy array

**stochastic\_constraints\_gradients**

Gradient estimates of stochastic constraints from each replication; # replications x # stochastic constraints x dimension.

**Type**

numpy array

**Parameters**

- **x** (`tuple`) – Vector of decision variables.
- **problem** (`base.Problem`) – Problem to which  $x$  is a solution.

**attach\_rngs** (`rng_list`, `copy=True`)

Attach a list of random-number generators to the solution.

**Parameters**

- **rng\_list** (list [`mrg32k3a.mrg32k3a.MRG32k3a`]) – List of random-number generators used to run simulation replications.
- **copy** (`bool`, `default=True`) – True if we want to copy the `mrg32k3a.MRG32k3a` objects, otherwise False.

**pad\_storage** (`m`)

Append zeros to numpy arrays for summary statistics.

**Parameters**

- **m** (`int`) – Number of replications to simulate.

**recompute\_summary\_statistics** ()

Recompute summary statistics of the solution.

## Notes

Statistics for gradients of objectives and stochastic constraint LHSs are temporarily commented out. Under development.

**class** `simopt.base.Solver` (*fixed\_factors*)

Bases: `object`

Base class to implement simulation-optimization solvers.

**name**

Name of solver.

**Type**

`str`

**objective\_type**

Description of objective types: “single” or “multi”.

**Type**

`str`

**constraint\_type**

Description of constraints types: “unconstrained”, “box”, “deterministic”, “stochastic”.

**Type**

`str`

**variable\_type**

Description of variable types: “discrete”, “continuous”, “mixed”.

**Type**

`str`

**gradient\_needed**

True if gradient of objective function is needed, otherwise False.

**Type**

`bool`

**factors**

Changeable factors (i.e., parameters) of the solver.

**Type**

`dict`

**specifications**

Details of each factor (for GUI, data validation, and defaults).

**Type**

`dict`

**rng\_list**

List of RNGs used for the solver’s internal purposes.

**Type**

`list [mrg32k3a.mrg32k3a.MRG32k3a]`

**solution\_progenitor\_rngs**

List of RNGs used as a baseline for simulating solutions.

**Type**

`list [mrg32k3a.mrg32k3a.MRG32k3a]`

**Parameters**

**fixed\_factors** (*dict*) – Dictionary of user-specified solver factors.

**attach\_rngs** (*rng\_list*)

Attach a list of random-number generators to the solver.

**Parameters**

**rng\_list** (`list [mrg32k3a.mrg32k3a.MRG32k3a]`) – List of random-number generators used for the solver's internal purposes.

**check\_crn\_across\_solns** ()

Check solver factor `crn_across_solns`.

**Notes**

Currently implemented to always return True. This factor must be a bool.

**check\_factor\_datatype** (*factor\_name*)

Determine if a factor's data type matches its specification.

**Parameters**

**factor\_name** (*str*) – String corresponding to name of factor to check.

**Returns**

**is\_right\_type** – True if factor is of specified data type, otherwise False.

**Return type**

`bool`

**check\_solver\_factor** (*factor\_name*)

Determine if the setting of a solver factor is permissible.

**Parameters**

**factor\_name** (*str*) – Name of factor for dictionary lookup (i.e., key).

**Returns**

**is\_permissible** – True if the solver factor is permissible, otherwise False.

**Return type**

`bool`

**check\_solver\_factors** ()

Determine if the joint settings of solver factors are permissible.

**Notes**

Each subclass of `base.Solver` has its own custom `check_solver_factors` method.

**Returns**

**is\_simulatable** – True if the solver factors are permissible, otherwise False.

**Return type**

`bool`

**create\_new\_solution** (*x*, *problem*)

Create a new solution object with attached RNGs primed to simulate replications.

**Parameters**

- **x** (*tuple*) – Vector of decision variables.
- **problem** (*base.Problem*) – Problem being solved by the solvers.

**Returns**

**new\_solution** – New solution.

**Return type**

*base.Solution*

**rebase** (*n\_reps*)

Rebase the progenitor rngs to start at a later substream index.

**Parameters**

**n\_reps** (*int*) – Substream index to skip to.

**solve** (*problem*)

Run a single macroreplication of a solver on a problem.

**Notes**

Each subclass of *base.Solver* has its own custom *solve* method.

**Parameters**

**problem** (*base.Problem*) – Simulation-optimization problem to solve.

**Returns**

- **recommended\_solns** (*list [Solution]*) – List of solutions recommended throughout the budget.
- **intermediate\_budgets** (*list [int]*) – List of intermediate budgets when recommended solutions changes.

### 2.1.1.5 simopt.data\_farming\_base module

**class** *simopt.data\_farming\_base.DataFarmingExperiment* (*model\_name*, *factor\_settings\_filename*,  
*factor\_headers*,  
*design\_filename=None*,  
*model\_fixed\_factors={}*)

Bases: *object*

Base class for data-farming experiments consisting of an model and design of associated factors.

**model**

Model on which the experiment is run.

**Type**

*base.Model*

**design**

List of design points forming the design.

**Type**

*list [data\_farming\_base.DesignPoint]*

**n\_design\_pts**

Number of design points in the design.

**Type**

int

**Parameters**

- **model\_name** (*str*) – Name of model on which the experiment is run.
- **factor\_settings\_filename** (*str*) – Name of .txt file containing factor ranges and # of digits.
- **factor\_headers** (*list* [*str*]) – Ordered list of factor names appearing in factor settings/design file.
- **design\_filename** (*str*) – Name of .txt file containing design matrix.
- **model\_fixed\_factors** (*dict*) – Non-default values of model factors that will not be varied.

**print\_to\_csv** (*csv\_filename*='raw\_results')

Extract observed responses from simulated design points and publish to .csv output file.

**Parameters**

**csv\_filename** (*str*, *default*="raw\_results") – Name of .csv file to print output to.

**run** (*n\_reps*=10, *crn\_across\_design\_pts*=True)

Run a fixed number of macroreplications at each design point.

**Parameters**

- **n\_reps** (*int*, *default*=10) – Number of replications run at each design point.
- **crn\_across\_design\_pts** (*bool*, *default*=True) – True if CRN are to be used across design points, otherwise False.

```
class simopt.data_farming_base.DataFarmingMetaExperiment (solver_name, problem_name,
                                                         solver_factor_headers,
                                                         solver_factor_settings_filename=None,
                                                         design_filename=None,
                                                         solver_fixed_factors=None,
                                                         problem_fixed_factors=None,
                                                         model_fixed_factors=None)
```

Bases: `object`

Base class for data-farming meta experiments consisting of problem-solver pairs and a design of associated factors.

**design**

List of design points forming the design.

**Type**

list [`experiment_base.ProblemSolver`]

**n\_design\_pts**

Number of design points in the design.

**Type**

int

**Parameters**

- **solver\_name** (*str*) – Name of solver.
- **problem\_name** (*str*) – Name of problem.
- **solver\_factor\_headers** (*list [str]*) – Ordered list of solver factor names appearing in factor settings/design file.
- **solver\_factor\_settings\_filename** (*str*, *default=None*) – Name of .txt file containing solver factor ranges and # of digits.
- **design\_filename** (*str*, *default=None*) – Name of .txt file containing design matrix.
- **solver\_fixed\_factors** (*dict*, *default=None*) – Dictionary of user-specified solver factors that will not be varied.
- **problem\_fixed\_factors** (*dict*, *default=None*) – Dictionary of user-specified problem factors that will not be varied.
- **model\_fixed\_factors** (*dict*, *default=None*) – Dictionary of user-specified model factors that will not be varied.

**post\_normalize** (*n\_postreps\_init\_opt*, *crn\_across\_init\_opt=True*)

Post-normalize problem-solver pairs.

**Parameters**

- **n\_postreps\_init\_opt** (*int*) – Number of postreplications to take at initial  $x_0$  and optimal  $x^*$ .
- **crn\_across\_init\_opt** (*bool*, *default=True*) – True if CRN are to be used for post-replications at solutions  $x_0$  and  $x^*$ , otherwise False.

**post\_replicate** (*n\_postreps*, *crn\_across\_budget=True*, *crn\_across\_macroreps=False*)

For each design point, run postreplications at solutions recommended by the solver on each macroreplication.

**Parameters**

- **n\_postreps** (*int*) – Number of postreplications to take at each recommended solution.
- **crn\_across\_budget** (*bool*, *default=True*) – True if CRN are to be used for post-replications at solutions recommended at different times, otherwise False.
- **crn\_across\_macroreps** (*bool*, *default=False*) – True if CRN are to be used for post-replications at solutions recommended on different macroreplications, otherwise False.

**report\_statistics** (*solve\_tols=[0.05, 0.1, 0.2, 0.5]*, *csv\_filename='df\_solver\_results'*)

For each design point, calculate statistics from each macroreplication and print to csv.

**Parameters**

- **solve\_tols** (*list [float]*, *default = [0.05, 0.10, 0.20, 0.50]*) – Relative optimality gap(s) defining when a problem is solved; in (0,1].
- **csv\_filename** (*str*, *default="df\_solver\_results"*) – Name of .csv file to print output to.

**run** (*n\_macroreps=10*)

Run *n\_macroreps* of each problem-solver design point.

**Parameters**

**n\_macroreps** (*int*) – Number of macroreplications for each design point.

**class** `simopt.data_farming_base.DesignPoint` (*model*)

Bases: `object`

Base class for design points represented as dictionaries of factors.

**model**

Model to simulate.

**Type**

`base.Model`

**model\_factors**

Model factor names and values.

**Type**

`dict`

**rng\_list**

Rngs for model to use when running replications at the solution.

**Type**

`list [mrg32k3a.mrg32k3a.MRG32k3a]`

**n\_reps**

Number of replications run at a design point.

**Type**

`int`

**responses**

Responses observed from replications.

**Type**

`dict`

**gradients**

Gradients of responses (w.r.t. model factors) observed from replications.

**Type**

`dict [dict]`

**Parameters**

**model** (`base.Model`) – Model with factors `model_factors`.

**attach\_rngs** (*rng\_list*, *copy=True*)

Attach a list of random-number generators to the design point.

**Parameters**

**rng\_list** (`list [mrg32k3a.mrg32k3a.MRG32k3a]`) – List of random-number generators used to run simulation replications.

**simulate** (*m=1*)

Simulate *m* replications for the current model factors and append results to the responses and gradients dictionaries.

**Parameters**

**m** (*int*, *default=1*) – Number of macroreplications to run at the design point; > 0.

### 2.1.1.6 simopt.directory module

#### Summary

Provide dictionary directories listing solvers, problems, and models.

### 2.1.1.7 simopt.experiment\_base module

#### Summary

Provide base classes for problem-solver pairs and helper functions for reading/writing data and plotting.

**class** `simopt.experiment_base.Curve` (*x\_vals*, *y\_vals*)

Bases: `object`

Base class for all curves.

**x\_vals**

Values of horizontal components.

**Type**

`list [float]`

**y\_vals**

Values of vertical components.

**Type**

`list [float]`

**n\_points**

Number of values in x- and y- vectors.

**Type**

`int`

**Parameters**

- **x\_vals** (`list [float]`) – Values of horizontal components.
- **y\_vals** (`list [float]`) – Values of vertical components.

**compute\_area\_under\_curve** ()

Compute the area under a curve.

**Returns**

**area** – Area under the curve.

**Return type**

`float`

**compute\_crossing\_time** (*threshold*)

Compute the first time at which a curve drops below a given threshold.

**Parameters**

**threshold** (`float`) – Value for which to find first crossing time.

**Returns**

**crossing\_time** – First time at which a curve drops below threshold.



**Return type**`float`**curve\_to\_full\_curve()**

Create a curve with duplicate x- and y-values to indicate steps.

**Returns**

**full\_curve** – Curve with duplicate x- and y-values.

**Return type**`experiment_base.Curve`**curve\_to\_mesh(mesh)**

Create a curve defined at equally spaced x values.

**Parameters**

**mesh** (*list of floats*) – List of uniformly spaced x-values.

**Returns**

**mesh\_curve** – Curve with equally spaced x-values.

**Return type**`experiment_base.Curve`**lookup(x)**

Lookup the y-value of the curve at an intermediate x-value.

**Parameters**

**x** (*float*) – X-value at which to lookup the y-value.

**Returns**

**y** – Y-value corresponding to x.

**Return type**`float`**plot(color\_str='C0', curve\_type='regular')**

Plot a curve.

**Parameters**

- **color\_str** (*str*, *default="C0"*) – String indicating line color, e.g., “C0”, “C1”, etc.
- **curve\_type** (*str*, *default="regular"*) – String indicating type of line: “regular” or “conf\_bound”.

**Returns**

**handle** – Curve handle, to use when creating legends.

**Return type**`list [matplotlib.lines.Line2D]`

```
class simopt.experiment_base.ProblemSolver (solver_name=None, problem_name=None,
                                             solver_rename=None, problem_rename=None,
                                             solver=None, problem=None,
                                             solver_fixed_factors=None,
                                             problem_fixed_factors=None,
                                             model_fixed_factors=None, file_name_path=None)
```

Bases: `object`

Base class for running one solver on one problem.

**solver**

Simulation-optimization solver.

**Type**

`base.Solver`

**problem**

Simulation-optimization problem.

**Type**

`base.Problem`

**n\_macroreps**

Number of macroreplications run.

**Type**

`int`

**file\_name\_path**

Path of .pickle file for saving `experiment_base.ProblemSolver` object.

**Type**

`str`

**all\_recommended\_xs**

Sequences of recommended solutions from each macroreplication.

**Type**

`list [list [tuple]]`

**all\_intermediate\_budgets**

Sequences of intermediate budgets from each macroreplication.

**Type**

`list [list]`

**timings**

Runtimes (in seconds) for each macroreplication.

**Type**

`list [float]`

**n\_postreps**

Number of postreplications to take at each recommended solution.

**Type**

`int`

**crn\_across\_budget**

True if CRN used for post-replications at solutions recommended at different times, otherwise False.

**Type**

`bool`

**crn\_across\_macroreps**

True if CRN used for post-replications at solutions recommended on different macroreplications, otherwise False.

**Type**

`bool`

**all\_post\_replicates**

All post-replicates from all solutions from all macroreplications.

**Type**

list [list [list]]

**all\_est\_objectives**

Estimated objective values of all solutions from all macroreplications.

**Type**

numpy array [numpy array]

**n\_postreps\_init\_opt**

Number of postreplications to take at initial solution ( $x_0$ ) and optimal solution ( $x^*$ ).

**Type**

int

**crn\_across\_init\_opt**

True if CRN used for post-replications at solutions  $x_0$  and  $x^*$ , otherwise False.

**Type**

bool

**x0**

Initial solution ( $x_0$ ).

**Type**

tuple

**x0\_postreps**

Post-replicates at  $x_0$ .

**Type**

list

**xstar**

Proxy for optimal solution ( $x^*$ ).

**Type**

tuple

**xstar\_postreps**

Post-replicates at  $x^*$ .

**Type**

list

**objective\_curves**

Curves of estimated objective function values, one for each macroreplication.

**Type**

list [experiment\_base.Curve]

**progress\_curves**

Progress curves, one for each macroreplication.

**Type**

list [experiment\_base.Curve]

**Parameters**

- **solver\_name**(*str*, *optional*) – Name of solver.
- **problem\_name**(*str*, *optional*) – Name of problem.
- **solver\_rename**(*str*, *optional*) – User-specified name for solver.
- **problem\_rename**(*str*, *optional*) – User-specified name for problem.
- **solver**(`base.Solver`, *optional*) – Simulation-optimization solver.
- **problem**(`base.Problem`, *optional*) – Simulation-optimization problem.
- **solver\_fixed\_factors**(*dict*, *optional*) – Dictionary of user-specified solver factors.
- **problem\_fixed\_factors**(*dict*, *optional*) – Dictionary of user-specified problem factors.
- **model\_fixed\_factors**(*dict*, *optional*) – Dictionary of user-specified model factors.
- **file\_name\_path**(*str*, *optional*) – Path of .pickle file for saving `experiment_base.ProblemSolver` objects.

**bootstrap\_sample**(*bootstrap\_rng*, *normalize=True*)

Generate a bootstrap sample of estimated objective curves or estimated progress curves.

**Parameters**

- **bootstrap\_rng**(`mrg32k3a.mrg32k3a.MRG32k3a`) – Random number generator to use for bootstrapping.
- **normalize**(*bool*, *default=True*) – True if progress curves are to be normalized w.r.t. optimality gaps, otherwise False.

**Returns**

**bootstrap\_curves** – Bootstrapped estimated objective curves or estimated progress curves of all solutions from all bootstrapped macroreplications.

**Return type**

`list[experiment_base.Curve]`

**check\_compatibility**()

Check whether the experiment's solver and problem are compatible.

**Returns**

**error\_str** – Error message in the event problem and solver are incompatible.

**Return type**

`str`

**check\_postnormalize**()

Check if the experiment has been postnormalized.

**Returns**

**postnormalized** – True if the experiment has been postnormalized, otherwise False.

**Return type**

`bool`

**check\_postreplicate**()

Check if the experiment has been postreplicated.

**Returns**

**postreplicated** – True if the experiment has been postreplicated, otherwise False.

**Return type**

bool

**check\_run()**

Check if the experiment has been run.

**Returns**

**ran** – True if the experiment been run, otherwise False.

**Return type**

bool

**clear\_postnorm()**

Delete results from `post_normalize()` associated with experiment.

**clear\_postreplicate()**

Delete results from `post_replicate()` method and any downstream results.

**clear\_run()**

Delete results from `run()` method and any downstream results.

**log\_experiment\_results** (*print\_solutions=True*)

Create readable .txt file from a problem-solver pair's .pickle file.

**post\_replicate** (*n\_postreps, crn\_across\_budget=True, crn\_across\_macroreps=False*)

Run postreplications at solutions recommended by the solver.

**Parameters**

- **n\_postreps** (*int*) – Number of postreplications to take at each recommended solution.
- **crn\_across\_budget** (*bool, default=True*) – True if CRN used for postreplications at solutions recommended at different times, otherwise False.
- **crn\_across\_macroreps** (*bool, default=False*) – True if CRN used for postreplications at solutions recommended on different macroreplications, otherwise False.

**record\_experiment\_results()**

Save `experiment_base.ProblemSolver` object to .pickle file.

**run** (*n\_macroreps*)

Run `n_macroreps` of the solver on the problem.

**Notes**

RNGs dedicated for random problem instances and temporarily unused. Under development.

**Parameters**

**n\_macroreps** (*int*) – Number of macroreplications of the solver to run on the problem.

```
class simopt.experiment_base.ProblemsSolvers (solver_names=None, problem_names=None,  
                                              solver_renames=None, problem_renames=None,  
                                              fixed_factors_filename=None, solvers=None,  
                                              problems=None, experiments=None,  
                                              file_name_path=None)
```

Bases: `object`

Base class for running one or more solver on one or more problem.

**`solver_names`**

List of solver names.

**Type**

`list [str]`

**`n_solvers`**

Number of solvers.

**Type**

`int`

**`problem_names`**

List of problem names.

**Type**

`list [str]`

**`n_problems`**

Number of problems.

**Type**

`int`

**`solvers`**

List of solvers.

**Type**

`list [base.Solver]`

**`problems`**

List of problems.

**Type**

`list [base.Problem]`

**`all_solver_fixed_factors`**

**Fixed solver factors for each solver:**

outer key is solver name; inner key is factor name.

**Type**

`dict [dict]`

**`all_problem_fixed_factors`**

**Fixed problem factors for each problem:**

outer key is problem name; inner key is factor name.

**Type**

`dict [dict]`

**all\_model\_fixed\_factors****Fixed model factors for each problem:**

outer key is problem name; inner key is factor name.

**Type**

dict of dict

**experiments**

All problem-solver pairs.

**Type**

list [list [experiment\_base.ProblemSolver]]

**file\_name\_path**

Path of .pickle file for saving experiment\_base.ProblemsSolvers object.

**Type**

str

**Parameters**

- **solver\_names** (list [str], optional) – List of solver names.
- **problem\_names** (list [str], optional) – List of problem names.
- **solver\_renames** (list [str], optional) – User-specified names for solvers.
- **problem\_renames** (list [str], optional) – User-specified names for problems.
- **fixed\_factors\_filename** (str, optional) – Name of .py file containing dictionaries of fixed factors for solvers/problems/models.
- **solvers** (list [base.Solver], optional) – List of solvers.
- **problems** (list [base.Problem], optional) – List of problems.
- **experiments** (list [list [experiment\_base.ProblemSolver]], optional) – All problem-solver pairs.
- **file\_name\_path** (str) – Path of .pickle file for saving experiment\_base.ProblemsSolvers object.

**check\_compatibility()**

Check whether all experiments' solvers and problems are compatible.

**Returns**

**error\_str** – Error message in the event any problem and solver are incompatible.

**Return type**

str

**log\_group\_experiment\_results()**

Create readable .txt file describing the solvers and problems that make up the ProblemSolvers object.

**post\_normalize** (n\_postreps\_init\_opt, crn\_across\_init\_opt=True)

Construct objective curves and (normalized) progress curves for all collections of experiments on all given problem.

**Parameters**

- **experiments** (list [experiment\_base.ProblemSolver]) – Problem-solver pairs of different solvers on a common problem.
- **n\_postreps\_init\_opt** (*int*) – Number of postreplications to take at initial  $x_0$  and optimal  $x^*$ .
- **crn\_across\_init\_opt** (*bool*, *default=True*) – True if CRN used for postreplications at solutions  $x_0$  and  $x^*$ , otherwise False.

**post\_replicate** (*n\_postreps*, *crn\_across\_budget=True*, *crn\_across\_macroreps=False*)

For each problem-solver pair, run postreplications at solutions recommended by the solver on each macroreplication.

#### Parameters

- **n\_postreps** (*int*) – Number of postreplications to take at each recommended solution.
- **crn\_across\_budget** (*bool*, *default=True*) – True if CRN used for postreplications at solutions recommended at different times, otherwise False.
- **crn\_across\_macroreps** (*bool*, *default=False*) – True if CRN used for postreplications at solutions recommended on different macroreplications, otherwise False.

**record\_group\_experiment\_results** ()

Save `experiment_base.ProblemsSolvers` object to .pickle file.

**run** (*n\_macroreps*)

Run *n\_macroreps* of each solver on each problem.

#### Parameters

- **n\_macroreps** (*int*) – Number of macroreplications of the solver to run on the problem.

`simopt.experiment_base.bootstrap_procedure` (*experiments*, *n\_bootstraps*, *conf\_level*, *plot\_type*, *beta=None*, *solve\_tol=None*, *estimator=None*, *normalize=True*)

Obtain bootstrap sample and compute confidence intervals.

#### Parameters

- **experiments** (list [list [experiment\_base.ProblemSolver]]) – Problem-solver pairs of different solvers and/or problems.
- **n\_bootstraps** (*int*) – Number of times to generate a bootstrap sample of estimated progress curves.
- **conf\_level** (*float*) – Confidence level for confidence intervals, i.e.,  $1-\gamma$ ; in (0, 1).
- **plot\_type** (*str*) –

String indicating which type of plot to produce:

- "mean" : estimated mean progress curve;
- "quantile" : estimated beta quantile progress curve;
- "area\_mean" : mean of area under progress curve;
- "area\_std\_dev" : standard deviation of area under progress curve;
- "solve\_time\_quantile" : beta quantile of solve time;
- "solve\_time\_cdf" : cdf of solve time;
- "cdf\_solvability" : cdf solvability profile;
- "quantile\_solvability" : quantile solvability profile;



"diff\_cdf\_solvability" : difference of cdf solvability profiles;

"diff\_quantile\_solvability" : difference of quantile solvability profiles.

- **beta** (*float*, *optional*) – Quantile to plot, e.g., beta quantile; in (0, 1).
- **solve\_tol** (*float*, *optional*) – Relative optimality gap defining when a problem is solved; in (0, 1].
- **estimator** (float or `experiment_base.Curve`, *optional*) – Main estimator, e.g., mean convergence curve from an experiment.
- **normalize** (*bool*, *default=True*) – True if progress curves are to be normalized w.r.t. optimality gaps, otherwise False.

#### Returns

Lower and upper bound(s) of bootstrap CI(s), as floats or curves.

#### Return type

`bs_CI_lower_bounds`, `bs_CI_upper_bounds` = float or `experiment_base.Curve`

`simopt.experiment_base.bootstrap_sample_all` (*experiments*, *bootstrap\_rng*, *normalize=True*)

Generate bootstrap samples of estimated progress curves (normalized and unnormalized) from a set of experiments.

#### Parameters

- **experiments** (list [list [`experiment_base.ProblemSolver`]]) – Problem-solver pairs of different solvers and/or problems.
- **bootstrap\_rng** (`mrng32k3a.mrng32k3a.MRG32k3a`) – Random number generator to use for bootstrapping.
- **normalize** (*bool*, *default=True*) – True if progress curves are to be normalized w.r.t. optimality gaps, otherwise False.

#### Returns

**bootstrap\_curves** – Bootstrapped estimated objective curves or estimated progress curves of all solutions from all macroreplications.

#### Return type

list [list [list [`experiment_base.Curve`]]]

`simopt.experiment_base.cdf_of_curves_crossing_times` (*curves*, *threshold*)

Compute the cdf of crossing times of curves.

#### Parameters

- **curves** (list [`experiment_base.Curve`]) – Collection of curves to aggregate.
- **threshold** (*float*) – Value for which to find first crossing time.

#### Returns

**cdf\_curve** – CDF of crossing times.

#### Return type

`experiment_base.Curve`

`simopt.experiment_base.check_common_problem_and_reference` (*experiments*)

Check if a collection of experiments have the same problem,  $x_0$ , and  $x^*$ .

#### Parameters

- **experiments** (list [`experiment_base.ProblemSolver`]) – Problem-solver pairs of different solvers on a common problem.

```
simopt.experiment_base.compute_bootstrap_CI (observations, conf_level, bias_correction=True,  
                                             overall_estimator=None)
```

Construct a bootstrap confidence interval for an estimator.

**Parameters**

- **observations** (*list*) – Estimators from all bootstrap instances.
- **conf\_level** (*float*) – Confidence level for confidence intervals, i.e.,  $1-\gamma$ ; in (0, 1).
- **bias\_correction** (*bool*, *default=True*) – True if bias-corrected bootstrap CIs (via percentile method) are to be used, otherwise False.
- **overall\_estimator** (*float*, *optional*) – Estimator to compute bootstrap confidence interval of; required for bias corrected CI.

**Returns**

- **bs\_CI\_lower\_bound** (*float*) – Lower bound of bootstrap CI.
- **bs\_CI\_upper\_bound** (*float*) – Upper bound of bootstrap CI.

```
simopt.experiment_base.difference_of_curves (curve1, curve2)
```

Compute the difference of two curves (Curve 1 - Curve 2).

**Parameters**

- **curve1** (*experiment\_base.Curve*) – Curves to take the difference of.
- **curve2** (*experiment\_base.Curve*) – Curves to take the difference of.

**Returns**

**difference\_curve** – Difference of curves.

**Return type**

*experiment\_base.Curve*

```
simopt.experiment_base.find_missing_experiments (experiments)
```

Identify problem-solver pairs that are not part of a list of experiments.

**Parameters**

**experiments** (*list* [*experiment\_base.ProblemSolver*]) – Problem-solver pairs of different solvers on different problems.

**Returns**

- **unique\_solvers** (*list* [*base.Solver*]) – List of solvers present in the list of experiments
- **unique\_problems** (*list* [*base.Problem*]) – List of problems present in the list of experiments.
- **missing** (*list* [*tuple* [*base.Solver*, *base.Problem*]]) – List of names of missing problem-solver pairs.

```
simopt.experiment_base.find_unique_solvers_problems (experiments)
```

Identify the unique problems and solvers in a collection of experiments.

**Parameters**

**experiments** (*list* [*experiment\_base.ProblemSolver*]) – ProblemSolver pairs of different solvers on different problems.

**Returns**

- **unique\_solvers** (*list* [*base.Solver*]) – Unique solvers.

- **unique\_problems** (list [base.Problem]) – Unique problems.

`simopt.experiment_base.functional_of_curves` (*bootstrap\_curves*, *plot\_type*, *beta*=0.5, *solve\_tol*=0.1)

Compute a functional of the bootstrapped objective/progress curves.

#### Parameters

- **bootstrap\_curves** (list [list [list [experiment\_base.Curve]]]) – Bootstrapped estimated objective curves or estimated progress curves of all solutions from all macroreplications.
- **plot\_type** (*str*) –

String indicating which type of plot to produce:

- "mean" : estimated mean progress curve;
- "quantile" : estimated beta quantile progress curve;
- "area\_mean" : mean of area under progress curve;
- "area\_std\_dev" : standard deviation of area under progress curve;
- "solve\_time\_quantile" : beta quantile of solve time;
- "solve\_time\_cdf" : cdf of solve time;
- "cdf\_solvability" : cdf solvability profile;
- "quantile\_solvability" : quantile solvability profile;
- "diff\_cdf\_solvability" : difference of cdf solvability profiles;
- "diff\_quantile\_solvability" : difference of quantile solvability profiles;

- **beta** (*float*, *default*=0.5) – Quantile to plot, e.g., beta quantile; in (0, 1).
- **solve\_tol** (*float*, *default*=0.1) – Relative optimality gap defining when a problem is solved; in (0, 1].

#### Returns

**functional** – Functional of bootstrapped curves, e.g., mean progress curves, mean area under progress curve, quantile of crossing time, etc.

#### Return type

list

`simopt.experiment_base.make_full_metaexperiment` (*existing\_experiments*, *unique\_solvers*, *unique\_problems*, *missing\_experiments*)

Create experiment objects for missing problem-solver pairs and run them.

#### Parameters

- **existing\_experiments** (list [experiment\_base.ProblemSolver]) – Problem-solver pairs of different solvers on different problems.
- **unique\_solvers** (list [base.Solver objects]) – List of solvers present in the list of experiments.
- **unique\_problems** (list [base.Problem]) – List of problems present in the list of experiments.
- **missing\_experiments** (list [tuple [base.Solver, base.Problem]]) – List of missing problem-solver pairs.

**Returns**

**metaexperiment** – New ProblemsSolvers object.

**Return type**

`experiment_base.ProblemsSolvers`

`simopt.experiment_base.max_difference_of_curves` (*curve1*, *curve2*)

Compute the maximum difference of two curves (Curve 1 - Curve 2).

**Parameters**

- **curve1** (`experiment_base.Curve`) – Curves to take the difference of.
- **curve2** (`experiment_base.Curve`) – Curves to take the difference of.

**Returns**

**max\_diff** – Maximum difference of curves.

**Return type**

`float`

`simopt.experiment_base.mean_of_curves` (*curves*)

Compute pointwise (w.r.t. x-values) mean of curves. Starting and ending x-values must coincide for all curves.

**Parameters**

**curves** (`list [experiment_base.Curve]`) – Collection of curves to aggregate.

**Returns**

**mean\_curve** – Mean curve.

**Return type**

`experiment_base.Curve` object

`simopt.experiment_base.plot_area_scatterplots` (*experiments*, *all\_in\_one=True*,  
*n\_bootstraps=100*, *conf\_level=0.95*,  
*plot\_CIs=True*, *print\_max\_hw=True*)

Plot a scatter plot of mean and standard deviation of area under progress curves. Either one plot for each solver or one plot for all solvers.

**Notes**

TO DO: Add the capability to compute and print the max halfwidth of the bootstrapped CI intervals.

**Parameters**

- **experiments** (`list [list [experiment_base.ProblemSolver]]`) – Problem-solver pairs used to produce plots.
- **all\_in\_one** (`bool`, *default=True*) – True if curves are to be plotted together, otherwise False.
- **n\_bootstraps** (`int`, *default=100*) – Number of bootstrap samples.
- **conf\_level** (`float`) – Confidence level for confidence intervals, i.e., 1-gamma; in (0, 1).
- **plot\_CIs** (`bool`, *default=True*) – True if bootstrapping confidence intervals are to be plotted, otherwise False.
- **print\_max\_hw** (`bool`, *default=True*) – True if caption with max half-width is to be printed, otherwise False.

**Returns**

**file\_list** – List compiling path names for plots produced.

**Return type**

`list [str]`

```
simopt.experiment_base.plot_bootstrap_CIs (bs_CI_lower_bounds, bs_CI_upper_bounds,
                                           color_str='C0')
```

Plot bootstrap confidence intervals.

**Parameters**

- **bs\_CI\_lower\_bounds** (`experiment_base.Curve`) – Lower and upper bounds of bootstrap CIs, as curves.
- **bs\_CI\_upper\_bounds** (`experiment_base.Curve`) – Lower and upper bounds of bootstrap CIs, as curves.
- **color\_str** (`str`, `default="C0"`) – String indicating line color, e.g., “C0”, “C1”, etc.

```
simopt.experiment_base.plot_progress_curves (experiments, plot_type, beta=0.5, normalize=True,
                                             all_in_one=True, n_bootstraps=100,
                                             conf_level=0.95, plot_CIs=True,
                                             print_max_hw=True)
```

Plot individual or aggregate progress curves for one or more solvers on a single problem.

**Parameters**

- **experiments** (`list [experiment_base.ProblemSolver]`) – Problem-solver pairs of different solvers on a common problem.
- **plot\_type** (`str`) –  
**String indicating which type of plot to produce:**  
“all” : all estimated progress curves;  
“mean” : estimated mean progress curve;  
“quantile” : estimated beta quantile progress curve.

- **beta** (`float`, `default=0.50`) – Quantile to plot, e.g., beta quantile; in (0, 1).
- **normalize** (`bool`, `default=True`) – True if progress curves are to be normalized w.r.t. optimality gaps, otherwise False.
- **all\_in\_one** (`bool`, `default=True`) – True if curves are to be plotted together, otherwise False.
- **n\_bootstraps** (`int`, `default=100`) – Number of bootstrap samples.
- **conf\_level** (`float`) – Confidence level for confidence intervals, i.e., 1-gamma; in (0, 1).
- **plot\_CIs** (`bool`, `default=True`) – True if bootstrapping confidence intervals are to be plotted, otherwise False.
- **print\_max\_hw** (`bool`, `default=True`) – True if caption with max half-width is to be printed, otherwise False.

**Returns**

**file\_list** – List compiling path names for plots produced.

**Return type**

`list [str]`

```
simopt.experiment_base.plot_solvability_cdfs (experiments, solve_tol=0.1, all_in_one=True,  
                                              n_bootstraps=100, conf_level=0.95,  
                                              plot_CIs=True, print_max_hw=True)
```

Plot the solvability cdf for one or more solvers on a single problem.

#### Parameters

- **experiments** (list [experiment\_base.ProblemSolver]) – Problem-solver pairs of different solvers on a common problem.
- **solve\_tol** (*float*, *default*=0.1) – Relative optimality gap defining when a problem is solved; in (0, 1].
- **all\_in\_one** (*bool*, *default*=True) – True if curves are to be plotted together, otherwise False.
- **n\_bootstraps** (*int*, *default*=100) – Number of bootstrap samples.
- **conf\_level** (*float*) – Confidence level for confidence intervals, i.e., 1-gamma; in (0, 1).
- **plot\_CIs** (*bool*, *default*=True) – True if bootstrapping confidence intervals are to be plotted, otherwise False.
- **print\_max\_hw** (*bool*, *default*=True) – True if caption with max half-width is to be printed, otherwise False.

#### Returns

**file\_list** – List compiling path names for plots produced.

#### Return type

list [str]

```
simopt.experiment_base.plot_solvability_profiles (experiments, plot_type, all_in_one=True,  
                                                  n_bootstraps=100, conf_level=0.95,  
                                                  plot_CIs=True, print_max_hw=True,  
                                                  solve_tol=0.1, beta=0.5, ref_solver=None)
```

Plot the (difference of) solvability profiles for each solver on a set of problems.

#### Parameters

- **experiments** (list [list [experiment\_base.ProblemSolver]]) – Problem-solver pairs used to produce plots.
- **plot\_type** (*str*) –  
**String indicating which type of plot to produce:**
  - ”cdf\_solvability” : cdf-solvability profile;
  - ”quantile\_solvability” : quantile-solvability profile;
  - ”diff\_cdf\_solvability” : difference of cdf-solvability profiles;
  - ”diff\_quantile\_solvability” : difference of quantile-solvability profiles.
- **all\_in\_one** (*bool*, *default*=True) – True if curves are to be plotted together, otherwise False.
- **n\_bootstraps** (*int*, *default*=100) – Number of bootstrap samples.
- **conf\_level** (*float*) – Confidence level for confidence intervals, i.e., 1-gamma; in (0, 1).
- **plot\_CIs** (*bool*, *default*=True) – True if bootstrapping confidence intervals are to be plotted, otherwise False.

- **print\_max\_hw** (*bool*, *default=True*) – True if caption with max half-width is to be printed, otherwise False.
- **solve\_tol** (*float*, *default=0.1*) – Relative optimality gap defining when a problem is solved; in (0, 1].
- **beta** (*float*, *default=0.5*) – Quantile to compute, e.g., beta quantile; in (0, 1).
- **ref\_solver** (*str*, *optional*) – Name of solver used as benchmark for difference profiles.

**Returns**

**file\_list** – List compiling path names for plots produced.

**Return type**

*list* [*str*]

```
simopt.experiment_base.plot_terminal_progress(experiments, plot_type='violin', normalize=True,
                                              all_in_one=True)
```

Plot individual or aggregate terminal progress for one or more solvers on a single problem.

**Parameters**

- **experiments** (*list* [*experiment\_base.ProblemSolver*]) – ProblemSolver pairs of different solvers on a common problem.
- **plot\_type** (*str*, *default="violin"*) – String indicating which type of plot to produce:
  - ”box” : comparative box plots;
  - ”violin” : comparative violin plots.
- **normalize** (*bool*, *default=True*) – True if progress curves are to be normalized w.r.t. optimality gaps, otherwise False.
- **all\_in\_one** (*bool*, *default=True*) – True if curves are to be plotted together, otherwise False.

**Returns**

**file\_list** – List compiling path names for plots produced.

**Return type**

*list* [*str*]

```
simopt.experiment_base.plot_terminal_scatterplots(experiments, all_in_one=True)
```

Plot a scatter plot of mean and standard deviation of terminal progress. Either one plot for each solver or one plot for all solvers.

**Parameters**

- **experiments** (*list* [*list* [*experiment\_base.Experiment*]]) – ProblemSolver pairs used to produce plots.
- **all\_in\_one** (*bool*, *default=True*) – True if curves are to be plotted together, otherwise False.

**Returns**

**file\_list** – List compiling path names for plots produced.

**Return type**

*list* [*str*]

```
simopt.experiment_base.post_normalize(experiments, n_postreps_init_opt, crn_across_init_opt=True,  
                                     proxy_init_val=None, proxy_opt_val=None,  
                                     proxy_opt_x=None)
```

Construct objective curves and (normalized) progress curves for a collection of experiments on a given problem.

#### Parameters

- **experiments** (list [experiment\_base.ProblemSolver]) – Problem-solver pairs of different solvers on a common problem.
- **n\_postreps\_init\_opt** (*int*) – Number of postreplications to take at initial  $x_0$  and optimal  $x^*$ .
- **crn\_across\_init\_opt** (*bool*, *default=True*) – True if CRN used for post-replications at solutions  $x_0$  and  $x^*$ , otherwise False.
- **proxy\_init\_val** (*float*, *optional*) – Known objective function value of initial solution.
- **proxy\_opt\_val** (*float*, *optional*) – Proxy for or bound on optimal objective function value.
- **proxy\_opt\_x** (*tuple*, *optional*) – Proxy for optimal solution.

```
simopt.experiment_base.quantile_cross_jump(curves, threshold, beta)
```

Compute a simple curve with a jump at the quantile of the crossing times.

#### Parameters

- **curves** (list [experiment\_base.Curve]) – Collection of curves to aggregate.
- **threshold** (*float*) – Value for which to find first crossing time.
- **beta** (*float*) – Quantile level.

#### Returns

**jump\_curve** – Piecewise-constant curve with a jump at the quantile crossing time (if finite).

#### Return type

experiment\_base.Curve

```
simopt.experiment_base.quantile_of_curves(curves, beta)
```

Compute pointwise (w.r.t.  $x$  values) quantile of curves. Starting and ending  $x$  values must coincide for all curves.

#### Parameters

- **curves** (list [experiment\_base.Curve]) – Collection of curves to aggregate.
- **beta** (*float*) – Quantile level.

#### Returns

**quantile\_curve** – Quantile curve.

#### Return type

experiment\_base.Curve

```
simopt.experiment_base.read_experiment_results(file_name_path)
```

Read in experiment\_base.ProblemSolver object from .pickle file.

#### Parameters

**file\_name\_path** (*str*) – Path of .pickle file for reading experiment\_base.ProblemSolver object.



**Returns**

**experiment** – Problem-solver pair that has been run or has been post-processed.

**Return type**

`experiment_base.ProblemSolver`

`simopt.experiment_base.read_group_experiment_results (file_name_path)`

Read in `experiment_base.ProblemsSolvers` object from .pickle file.

**Parameters**

**file\_name\_path** (*str*) – Path of .pickle file for reading `experiment_base.ProblemsSolvers` object.

**Returns**

**groupexperiment** – Problem-solver group that has been run or has been post-processed.

**Return type**

`experiment_base.ProblemsSolvers`

`simopt.experiment_base.report_max_halfwidth (curve_pairs, normalize, conf_level, difference=False)`

Compute and print caption for max halfwidth of one or more bootstrap CI curves.

**Parameters**

- **curve\_pairs** (list [list [`experiment_base.Curve`]]) – List of paired bootstrap CI curves.
- **normalize** (*bool*) – True if progress curves are to be normalized w.r.t. optimality gaps, otherwise False.
- **conf\_level** (*float*) – Confidence level for confidence intervals, i.e.,  $1-\gamma$ ; in (0, 1).
- **difference** (*bool*) – True if the plot is for difference profiles, otherwise False.

`simopt.experiment_base.save_plot (solver_name, problem_name, plot_type, normalize, extra=None)`

Create new figure. Add labels to plot and reformat axes.

**Parameters**

- **solver\_name** (*str*) – Name of solver.
- **problem\_name** (*str*) – Name of problem.
- **plot\_type** (*str*) –

**String indicating which type of plot to produce:**

"all" : all estimated progress curves;  
 "mean" : estimated mean progress curve;  
 "quantile" : estimated beta quantile progress curve;  
 "solve\_time\_cdf" : cdf of solve time;  
 "cdf\_solvability" : cdf solvability profile;  
 "quantile\_solvability" : quantile solvability profile;  
 "diff\_cdf\_solvability" : difference of cdf solvability profiles;  
 "diff\_quantile\_solvability" : difference of quantile solvability profiles;  
 "area" : area scatterplot;  
 "terminal\_scatter" : scatterplot of mean and std dev of terminal progress.

- **normalize** (*bool*) – True if progress curves are to be normalized w.r.t. optimality gaps, otherwise False.
- **extra** (*float or list [float]*, *optional*) – Extra number(s) specifying quantile (e.g., beta) and/or solve tolerance.

**Returns**

**path\_name** – Path name pointing to location where plot will be saved.

**Return type**

*str*

`simopt.experiment_base.setup_plot` (*plot\_type*, *solver\_name*='SOLVER SET', *problem\_name*='PROBLEM SET', *normalize*=True, *budget*=None, *beta*=None, *solve\_tol*=None)

Create new figure. Add labels to plot and reformat axes.

**Parameters**

- **plot\_type** (*str*) –  
**String indicating which type of plot to produce:**
  - "all" : all estimated progress curves;
  - "mean" : estimated mean progress curve;
  - "quantile" : estimated beta quantile progress curve;
  - "solve\_time\_cdf" : cdf of solve time;
  - "cdf\_solvability" : cdf solvability profile;
  - "quantile\_solvability" : quantile solvability profile;
  - "diff\_cdf\_solvability" : difference of cdf solvability profiles;
  - "diff\_quantile\_solvability" : difference of quantile solvability profiles;
  - "area" : area scatterplot;
  - "box" : box plot of terminal progress;
  - "violin" : violin plot of terminal progress;
  - "terminal\_scatter" : scatterplot of mean and std dev of terminal progress.
- **solver\_name** (*str*, *default*="SOLVER\_SET") – Name of solver.
- **problem\_name** (*str*, *default*="PROBLEM\_SET") – Name of problem.
- **normalize** (*bool*, *default*=True) – True if progress curves are to be normalized w.r.t. optimality gaps, otherwise False.
- **budget** (*int*, *optional*) – Budget of problem, measured in function evaluations.
- **beta** (*float*, *optional*) – Quantile to compute, e.g., beta quantile; in (0, 1).
- **solve\_tol** (*float*, *optional*) – Relative optimality gap defining when a problem is solved; in (0, 1].

`simopt.experiment_base.trim_solver_results` (*problem*, *recommended\_solns*, *intermediate\_budgets*)

Trim solutions recommended by solver after problem's max budget.

**Parameters**

- **problem** (`base.Problem`) – Problem object on which the solver was run.

- **recommended\_solutions** (`list [base.Solution]`) – Solutions recommended by the solver.
- **intermediate\_budgets** (`list [int]`) – Intermediate budgets at which solver recommended different solutions.

#### 2.1.1.8 Module contents



## ACKNOWLEDGMENTS

An earlier website for [SimOpt](#) was developed through work supported by the National Science Foundation under grant nos. DMI-0400287, CMMI-0800688 and CMMI-1200315. Recent work on the development of SimOpt has been supported by the National Science Foundation under grant nos. DGE-1650441, CMMI-1537394, CMMI-1254298, CMMI-1536895, IIS-1247696, and TRIPODS+X DMS-1839346, by the Air Force Office of Scientific Research under grant nos. FA9550-12-1-0200, FA9550-15-1-0038, and FA9550-16-1-0046, and by the Army Research Office under grant no. W911NF-17-1-0094. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF).



## PYTHON MODULE INDEX

### S

- `simopt`, 175
- `simopt.base`, 140
- `simopt.data_farming_base`, 152
- `simopt.directory`, 156
- `simopt.experiment_base`, 156
- `simopt.GUI`, 135
- `simopt.models`, 121
  - `simopt.models.amusementpark`, 5
  - `simopt.models.chessmm`, 11
  - `simopt.models.cntnv`, 16
  - `simopt.models.contam`, 22
  - `simopt.models.dualsourcing`, 33
  - `simopt.models.dynamnews`, 39
  - `simopt.models.example`, 45
  - `simopt.models.facilitysizing`, 50
  - `simopt.models.fixedsan`, 61
  - `simopt.models.hotel`, 66
  - `simopt.models.ironore`, 72
  - `simopt.models.mm1queue`, 81
  - `simopt.models.network`, 87
  - `simopt.models.paramesti`, 93
  - `simopt.models.rmitd`, 98
  - `simopt.models.san`, 103
  - `simopt.models.sscont`, 108
  - `simopt.models.tableallocation`, 115
- `simopt.solvers`, 135
  - `simopt.solvers.adam`, 121
  - `simopt.solvers.aloe`, 123
  - `simopt.solvers.astrodf`, 125
  - `simopt.solvers.neldmd`, 127
  - `simopt.solvers.randomsearch`, 129
  - `simopt.solvers.spsa`, 130
  - `simopt.solvers.strong`, 133





## A

ADAM (class in *simopt.solvers.adam*), 121  
 add\_experiment (*simopt.GUI.Experiment\_Window* attribute), 136  
 add\_experiment () (*simopt.GUI.Experiment\_Window* method), 137  
 add\_meta\_exp\_to\_frame () (*simopt.GUI.Experiment\_Window* method), 137  
 add\_plot () (*simopt.GUI.Plot\_Window* method), 138  
 all\_est\_objectives (*simopt.experiment\_base.ProblemSolver* attribute), 159  
 all\_intermediate\_budgets (*simopt.experiment\_base.ProblemSolver* attribute), 158  
 all\_model\_fixed\_factors (*simopt.experiment\_base.ProblemsSolvers* attribute), 162  
 all\_post\_replicates (*simopt.experiment\_base.ProblemSolver* attribute), 158  
 all\_problem\_fixed\_factors (*simopt.experiment\_base.ProblemsSolvers* attribute), 162  
 all\_recommended\_xs (*simopt.experiment\_base.ProblemSolver* attribute), 158  
 all\_solver\_fixed\_factors (*simopt.experiment\_base.ProblemsSolvers* attribute), 162  
 ALOE (class in *simopt.solvers.aloe*), 123  
 AmusementPark (class in *simopt.models.amusementpark*), 5  
 AmusementParkMinDepart (class in *simopt.models.amusementpark*), 7  
 ASTRODF (class in *simopt.solvers.astrodf*), 125  
 attach\_rngs () (*simopt.base.Problem* method), 143  
 attach\_rngs () (*simopt.base.Solution* method), 149  
 attach\_rngs () (*simopt.base.Solver* method), 151  
 attach\_rngs () (*simopt.data\_farming\_base.DesignPoint* method), 155

## B

bootstrap\_procedure () (in module *simopt.experiment\_base*), 164  
 bootstrap\_sample () (*simopt.experiment\_base.ProblemSolver* method), 160  
 bootstrap\_sample\_all () (in module *simopt.experiment\_base*), 165

## C

cauchy\_point () (*simopt.solvers.strong.STRONG* method), 134  
 cdf\_of\_curves\_crossing\_times () (in module *simopt.experiment\_base*), 165  
 changeOnHover () (*simopt.GUI.Plot\_Window* method), 138  
 check\_allowable\_diff () (*simopt.models.chessmm.ChessMatchmaking* method), 16  
 check\_alpha () (*simopt.solvers.adam.ADAM* method), 122  
 check\_alpha () (*simopt.solvers.neldmd.NelderMead* method), 128  
 check\_alpha () (*simopt.solvers.spsa.SPSA* method), 131  
 check\_alpha\_0 () (*simopt.solvers.aloe.ALOE* method), 124  
 check\_alpha\_max () (*simopt.solvers.aloe.ALOE* method), 124  
 check\_arc\_costs () (*simopt.models.fixedsan.FixedSANLongestPath* method), 64  
 check\_arc\_costs () (in *simopt.models.san.SANLongestPath* method), 106  
 check\_arc\_means () (*simopt.models.fixedsan.FixedSAN* method), 62  
 check\_arc\_means () (*simopt.models.san.SAN* method), 104  
 check\_arcs () (*simopt.models.san.SAN* method), 104  
 check\_arrival\_gammas ()

(*simopt.models.amusementpark.AmusementPark*  
*method*), 6  
 check\_arrival\_rate() (*simopt.models.network.Network*  
*method*), 88  
 check\_backorder\_cost() (*simopt.models.sscont.SSCont* *method*), 110  
 check\_beta\_1() (*simopt.solvers.adam.ADAM*  
*method*), 122  
 check\_beta\_2() (*simopt.solvers.adam.ADAM*  
*method*), 122  
 check\_betap() (*simopt.solvers.neldmd.NelderMead*  
*method*), 128  
 check\_booking\_limits() (*simopt.models.hotel.Hotel* *method*), 67  
 check\_budget() (*simopt.base.Problem* *method*), 143  
 check\_budget() (*simopt.models.contam.ContaminationTotalCostCont*  
*method*), 26  
 check\_budget() (*simopt.models.hotel.HotelRevenue*  
*method*), 70  
 check\_Burr\_c() (*simopt.models.cntnv.CntNV*  
*method*), 17  
 check\_Burr\_k() (*simopt.models.cntnv.CntNV*  
*method*), 17  
 check\_c\_utility() (*simopt.models.dynamnews.DynamNews*  
*method*), 40  
 check\_capacity() (*simopt.models.facilitysizing.FacilitySize*  
*method*), 51  
 check\_capacity() (*simopt.models.ironore.IronOre*  
*method*), 73  
 check\_capacity() (*simopt.models.tableallocation.TableAllocation*  
*method*), 116  
 check\_common\_problem\_and\_reference() (*in*  
*module simopt.experiment\_base*), 165  
 check\_compatibility() (*simopt.experiment\_base.ProblemSolver* *method*),  
 160  
 check\_compatibility() (*simopt.experiment\_base.ProblemsSolvers*  
*method*), 163  
 check\_cons() (*in module simopt.solvers.spsa*), 132  
 check\_cons() (*simopt.solvers.strong.STRONG*  
*method*), 134  
 check\_const() (*simopt.solvers.neldmd.NelderMead*  
*method*), 128  
 check\_contam\_rate\_alpha() (*simopt.models.contam.Contamination* *method*),  
 22  
 check\_contam\_rate\_beta() (*simopt.models.contam.Contamination* *method*),  
 22  
 check\_cost() (*simopt.models.dynamnews.DynamNews*  
*method*), 40  
 check\_cost() (*simopt.models.rmitd.RMITD* *method*),  
 99  
 check\_cost\_exp() (*simopt.models.dualsourcing.DualSourcing*  
*method*), 34  
 check\_cost\_process() (*simopt.models.network.Network* *method*),  
 88  
 check\_cost\_reg() (*simopt.models.dualsourcing.DualSourcing*  
*method*), 34  
 check\_cost\_time() (*simopt.models.network.Network* *method*),  
 88  
 check\_cov() (*simopt.models.facilitysizing.FacilitySize*  
*method*), 51  
 check\_crn\_across\_solns() (*simopt.base.Solver*  
*method*), 151  
 check\_delta() (*simopt.solvers.neldmd.NelderMead*  
*method*), 128  
 check\_delta\_T() (*simopt.solvers.strong.STRONG*  
*method*), 134  
 check\_delta\_threshold() (*simopt.solvers.strong.STRONG* *method*), 134  
 check\_demand\_mean() (*simopt.models.sscont.SSCont*  
*method*), 110  
 check\_demand\_means() (*simopt.models.rmitd.RMITD* *method*), 99  
 check\_depart\_probabilities() (*simopt.models.amusementpark.AmusementPark*  
*method*), 6  
 check\_deterministic\_constraints() (*simopt.base.Problem* *method*), 144  
 check\_deterministic\_constraints() (*simopt.models.amusementpark.AmusementParkMinDepart*  
*method*), 9  
 check\_deterministic\_constraints() (*simopt.models.chessmm.ChessAvgDifference*  
*method*), 13  
 check\_deterministic\_constraints() (*simopt.models.cntnv.CntNVMaxProfit* *method*),  
 20  
 check\_deterministic\_constraints() (*simopt.models.contam.ContaminationTotalCostCont*  
*method*), 26  
 check\_deterministic\_constraints() (*simopt.models.contam.ContaminationTotalCostDisc*  
*method*), 31  
 check\_deterministic\_constraints() (*simopt.models.dualsourcing.DualSourcingMinCost*  
*method*), 38  
 check\_deterministic\_constraints() (*simopt.models.dynamnews.DynamNewsMaxProfit*  
*method*), 43  
 check\_deterministic\_constraints() (*simopt.models.example.ExampleProblem*

<i>method</i> ), 49	<i>method</i> ), 6
check_deterministic_constraints() ( <i>simopt.models.facilitysizing.FacilitySizingMaxService</i> <i>method</i> ), 54	check_erlang_shape() ( <i>simopt.models.amusementpark.AmusementPark</i> <i>method</i> ), 6
check_deterministic_constraints() ( <i>simopt.models.facilitysizing.FacilitySizingTotalCost</i> <i>method</i> ), 58	check_error_prob() ( <i>simopt.models.contam.ContaminationTotalCostCont</i> <i>method</i> ), 26
check_deterministic_constraints() ( <i>simopt.models.fixedsan.FixedSANLongestPath</i> <i>method</i> ), 64	check_error_prob() ( <i>simopt.models.contam.ContaminationTotalCostDisc</i> <i>method</i> ), 31
check_deterministic_constraints() ( <i>simopt.models.hotel.HotelRevenue</i> <i>method</i> ), 70	check_eta_0() ( <i>simopt.solvers.strong.STRONG</i> <i>method</i> ), 134
check_deterministic_constraints() ( <i>simopt.models.ironore.IronOreMaxRev</i> <i>method</i> ), 76	check_eta_1() ( <i>simopt.solvers.astrodf.ASTRODF</i> <i>method</i> ), 126
check_deterministic_constraints() ( <i>simopt.models.ironore.IronOreMaxRevCnt</i> <i>method</i> ), 80	check_eta_1() ( <i>simopt.solvers.strong.STRONG</i> <i>method</i> ), 134
check_deterministic_constraints() ( <i>simopt.models.ironore.IronOreMaxRevCnt</i> <i>method</i> ), 80	check_eta_2() ( <i>simopt.solvers.astrodf.ASTRODF</i> <i>method</i> ), 126
check_deterministic_constraints() ( <i>simopt.models.mm1queue.MM1MinMeanSojournTime</i> <i>method</i> ), 84	check_eval_pct() ( <i>simopt.solvers.spsa.SPSA</i> <i>method</i> ), 132
check_deterministic_constraints() ( <i>simopt.models.network.NetworkMinTotalCost</i> <i>method</i> ), 91	check_factor_datatype() ( <i>simopt.base.Model</i> <i>method</i> ), 140
check_deterministic_constraints() ( <i>simopt.models.paramesti.ParamEstiMaxLogLik</i> <i>method</i> ), 95	check_factor_datatype() ( <i>simopt.base.Problem</i> <i>method</i> ), 144
check_deterministic_constraints() ( <i>simopt.models.rmtd.RMTDMaxRevenue</i> <i>method</i> ), 102	check_factor_datatype() ( <i>simopt.base.Solver</i> <i>method</i> ), 151
check_deterministic_constraints() ( <i>simopt.models.san.SANLongestPath</i> <i>method</i> ), 106	check_factor_list ( <i>simopt.base.Model</i> <i>attribute</i> ), 140
check_deterministic_constraints() ( <i>simopt.models.sscont.SSContMinCost</i> <i>method</i> ), 113	check_factor_list ( <i>simopt.models.amusementpark.AmusementPark</i> <i>attribute</i> ), 6
check_deterministic_constraints() ( <i>simopt.models.tableallocation.TableAllocationMaxRev</i> <i>method</i> ), 119	check_factor_list ( <i>simopt.models.chessmm.ChessMatchmaking</i> <i>attribute</i> ), 15
check_discount_rate() ( <i>simopt.models.hotel.Hotel</i> <i>method</i> ), 67	check_factor_list ( <i>simopt.models.cntnv.CntNV</i> <i>at-</i> <i>tribute</i> ), 17
check_elo_mean() ( <i>simopt.models.chessmm.ChessMatchmaking</i> <i>method</i> ), 16	check_factor_list ( <i>simopt.models.contam.Contamination</i> <i>attribute</i> ), 22
check_elo_sd() ( <i>simopt.models.chessmm.ChessMatchmaking</i> <i>method</i> ), 16	check_factor_list ( <i>simopt.models.dualsourcing.DualSourcing</i> <i>attribute</i> ), 34
check_epsilon() ( <i>simopt.models.facilitysizing.FacilitySizingTotalCost</i> <i>method</i> ), 59	check_factor_list ( <i>simopt.models.dynamnews.DynamNews</i> <i>at-</i> <i>tribute</i> ), 40
check_epsilon() ( <i>simopt.solvers.adam.ADAM</i> <i>method</i> ), 122	check_factor_list ( <i>simopt.models.example.ExampleModel</i> <i>at-</i> <i>tribute</i> ), 46
check_epsilon_f() ( <i>simopt.solvers.aloe.ALOE</i> <i>method</i> ), 124	check_factor_list ( <i>simopt.models.facilitysizing.FacilitySize</i> <i>at-</i> <i>tribute</i> ), 51
check_erlang_scale() ( <i>simopt.models.amusementpark.AmusementPark</i> <i>method</i> ), 6	check_factor_list ( <i>simopt.models.fixedsan.FixedSAN</i> <i>attribute</i> ), 61
	check_factor_list ( <i>simopt.models.hotel.Hotel</i> <i>at-</i> <i>tribute</i> ), 6

tribute), 67

check\_factor\_list (simopt.models.ironore.IronOre attribute), 73

check\_factor\_list (simopt.models.mm1queue.MM1Queue attribute), 86

check\_factor\_list (simopt.models.network.Network attribute), 88

check\_factor\_list (simopt.models.paramesti.ParameterEstimation attribute), 97

check\_factor\_list (simopt.models.rmitd.RMITD attribute), 98

check\_factor\_list (simopt.models.san.SAN attribute), 103

check\_factor\_list (simopt.models.sscont.SSCont attribute), 109

check\_factor\_list (simopt.models.tableallocation.TableAllocation attribute), 115

check\_fixed\_cost() (simopt.models.sscont.SSCont method), 110

check\_gamma() (simopt.solvers.aloe.ALOE method), 124

check\_gamma() (simopt.solvers.spsa.SPSA method), 132

check\_gamma\_1() (simopt.solvers.astrodf.ASTRODF method), 126

check\_gamma\_1() (simopt.solvers.strong.STRONG method), 134

check\_gamma\_2() (simopt.solvers.astrodf.ASTRODF method), 126

check\_gamma\_2() (simopt.solvers.strong.STRONG method), 134

check\_gamma\_scale() (simopt.models.rmitd.RMITD method), 99

check\_gamma\_shape() (simopt.models.rmitd.RMITD method), 99

check\_gammap() (simopt.solvers.neldmd.NelderMead method), 128

check\_gavg() (simopt.solvers.spsa.SPSA method), 132

check\_holding\_cost() (simopt.models.dualsourcing.DualSourcing method), 34

check\_holding\_cost() (simopt.models.ironore.IronOre method), 73

check\_holding\_cost() (simopt.models.sscont.SSCont method), 110

check\_init\_level() (simopt.models.dynamnews.DynamNews method), 40

check\_initial\_inv() (simopt.models.dualsourcing.DualSourcing method), 34

check\_initial\_inventory() (simopt.models.rmitd.RMITD method), 99

check\_initial\_rate\_alpha() (simopt.models.contam.Contamination method), 22

check\_initial\_rate\_beta() (simopt.models.contam.Contamination method), 23

check\_initial\_solution() (simopt.base.Problem method), 144

check\_initial\_solution() (simopt.models.contam.ContaminationTotalCostCont method), 26

check\_initial\_solution() (simopt.models.hotel.HotelRevenue method), 70

check\_initial\_spread() (simopt.solvers.neldmd.NelderMead method), 128

check\_installation\_budget() (simopt.models.facilitysizing.FacilitySizingMaxService method), 55

check\_installation\_costs() (simopt.models.facilitysizing.FacilitySizingMaxService method), 55

check\_installation\_costs() (simopt.models.facilitysizing.FacilitySizingTotalCost method), 59

check\_inven\_stop() (simopt.models.ironore.IronOre method), 73

check\_iter\_pct() (simopt.solvers.spsa.SPSA method), 132

check\_lambda() (simopt.models.hotel.Hotel method), 67

check\_lambda() (simopt.models.mm1queue.MM1Queue method), 86

check\_lambda() (simopt.models.tableallocation.TableAllocation method), 116

check\_lambda() (simopt.solvers.aloe.ALOE method), 124

check\_lambda() (simopt.solvers.strong.STRONG method), 134

check\_lambda\_min() (simopt.solvers.astrodf.ASTRODF method), 126

check\_lead\_exp() (simopt.models.dualsourcing.DualSourcing method), 34

check\_lead\_mean() (simopt.models.sscont.SSCont method), 110

check\_lead\_reg() (simopt.models.dualsourcing.DualSourcing method), 35

check\_lower\_limits\_transit\_time() (simopt.models.network.Network method),

88

`check_max_price()` (*simopt.models.ironore.IronOre* method), 73

`check_max_prod_perday()` (*simopt.models.ironore.IronOre* method), 73

`check_mean_price()` (*simopt.models.ironore.IronOre* method), 73

`check_mean_vec()` (*simopt.models.facilitysizing.FacilitySize* method), 51

`check_min_price()` (*simopt.models.ironore.IronOre* method), 73

`check_mode_transit_time()` (*simopt.models.network.Network* method), 88

`check_mu()` (*simopt.models.dualsourcing.DualSourcing* method), 35

`check_mu()` (*simopt.models.dynamnews.DynamNews* method), 40

`check_mu()` (*simopt.models.mm1queue.MM1Queue* method), 86

`check_n_days()` (*simopt.models.dualsourcing.DualSourcing* method), 35

`check_n_days()` (*simopt.models.ironore.IronOre* method), 73

`check_n_days()` (*simopt.models.sscont.SSCont* method), 110

`check_n_fac()` (*simopt.models.facilitysizing.FacilitySize* method), 51

`check_n_hours()` (*simopt.models.tableallocation.TableAllocation* method), 116

`check_n_loss()` (*simopt.solvers.spsa.SPSA* method), 132

`check_n_messages()` (*simopt.models.network.Network* method), 88

`check_n_networks()` (*simopt.models.network.Network* method), 88

`check_n_r()` (*simopt.solvers.strong.STRONG* method), 134

`check_n_reps()` (*simopt.solvers.spsa.SPSA* method), 132

`check_num_arcs()` (*simopt.models.fixedsan.FixedSAN* method), 62

`check_num_customer()` (*simopt.models.dynamnews.DynamNews* method), 40

`check_num_nodes()` (*simopt.models.fixedsan.FixedSAN* method), 62

`check_num_nodes()` (*simopt.models.san.SAN* method), 104

`check_num_players()` (*simopt.models.chessmm.ChessMatchmaking* method), 16

`check_num_prod()` (*simopt.models.dynamnews.DynamNews* method), 40

`check_num_products()` (*simopt.models.hotel.Hotel* method), 67

`check_num_rooms()` (*simopt.models.hotel.Hotel* method), 67

`check_num_tables()` (*simopt.models.tableallocation.TableAllocation* method), 116

`check_number_attractions()` (*simopt.models.amusementpark.AmusementPark* method), 6

`check_order_level_exp()` (*simopt.models.dualsourcing.DualSourcing* method), 35

`check_order_level_reg()` (*simopt.models.dualsourcing.DualSourcing* method), 35

`check_order_quantity()` (*simopt.models.cntnv.CntNV* method), 17

`check_park_capacity()` (*simopt.models.amusementpark.AmusementPark* method), 6

`check_penalty_cost()` (*simopt.models.dualsourcing.DualSourcing* method), 35

`check_people()` (*simopt.models.mm1queue.MM1Queue* method), 86

`check_poisson_rate()` (*simopt.models.chessmm.ChessMatchmaking* method), 16

`check_postnormalize()` (*simopt.experiment\_base.ProblemSolver* method), 160

`check_postreplicate()` (*simopt.experiment\_base.ProblemSolver* method), 160

`check_prev_cost()` (*simopt.models.contam.Contamination* method), 23

`check_prev_cost()` (*simopt.models.contam.ContaminationTotalCostCont* method), 26

`check_prev_cost()` (*simopt.models.contam.ContaminationTotalCostDisc* method), 31

`check_prev_decision()` (*simopt.models.contam.Contamination* method), 23

`check_price()` (*simopt.models.dynamnews.DynamNews* method), 40

`check_price_prod()` (*simopt.models.ironore.IronOre* method), 73



`check_price_sell()` (*simopt.models.ironore.IronOre method*), 73  
`check_price_stop()` (*simopt.models.ironore.IronOre method*), 73  
`check_prices()` (*simopt.models.rmitd.RMITD method*), 99  
`check_problem_factor()` (*simopt.base.Problem method*), 144  
`check_problem_factors()` (*simopt.base.Problem method*), 144  
`check_problem_factors()` (*simopt.solvers.spsa.SPSA method*), 132  
`check_process_prob()` (*simopt.models.network.Network method*), 88  
`check_prod_cost()` (*simopt.models.ironore.IronOre method*), 73  
`check_product_incidence()` (*simopt.models.hotel.Hotel method*), 67  
`check_purchase_price()` (*simopt.models.cntnv.CntNV method*), 17  
`check_queue_capacities()` (*simopt.models.amusementpark.AmusementPark method*), 6  
`check_r()` (*simopt.solvers.adam.ADAM method*), 122  
`check_r()` (*simopt.solvers.aloe.ALOE method*), 124  
`check_r()` (*simopt.solvers.neldmd.NelderMead method*), 128  
`check_rack_rate()` (*simopt.models.hotel.Hotel method*), 67  
`check_reservation_qtys()` (*simopt.models.rmitd.RMITD method*), 99  
`check_restore_rate_alpha()` (*simopt.models.contam.Contamination method*), 23  
`check_restore_rate_beta()` (*simopt.models.contam.Contamination method*), 23  
`check_run()` (*simopt.experiment\_base.ProblemSolver method*), 161  
`check_runlength()` (*simopt.models.hotel.Hotel method*), 67  
`check_S()` (*simopt.models.sscont.SSCont method*), 110  
`check_s()` (*simopt.models.sscont.SSCont method*), 110  
`check_sales_price()` (*simopt.models.cntnv.CntNV method*), 17  
`check_salvage_price()` (*simopt.models.cntnv.CntNV method*), 17  
`check_sample_size()` (*simopt.solvers.randomsearch.RandomSearch method*), 130  
`check_sensitivity()` (*simopt.solvers.adam.ADAM method*), 122  
`check_sensitivity()` (*simopt.solvers.aloe.ALOE method*), 124  
`check_sensitivity()` (*simopt.solvers.neldmd.NelderMead method*), 128  
`check_sensitivity()` (*simopt.solvers.strong.STRONG method*), 134  
`check_service_time_means()` (*simopt.models.tableallocation.TableAllocation method*), 116  
`check_simulatable_factor()` (*simopt.base.Model method*), 140  
`check_simulatable_factors()` (*simopt.base.Model method*), 141  
`check_simulatable_factors()` (*simopt.models.amusementpark.AmusementPark method*), 6  
`check_simulatable_factors()` (*simopt.models.cntnv.CntNV method*), 17  
`check_simulatable_factors()` (*simopt.models.contam.Contamination method*), 23  
`check_simulatable_factors()` (*simopt.models.contam.ContaminationTotalCostCont method*), 26  
`check_simulatable_factors()` (*simopt.models.dualsourcing.DualSourcing method*), 35  
`check_simulatable_factors()` (*simopt.models.dynamnews.DynamNews method*), 40  
`check_simulatable_factors()` (*simopt.models.example.ExampleModel method*), 46  
`check_simulatable_factors()` (*simopt.models.facilitysizing.FacilitySize method*), 51  
`check_simulatable_factors()` (*simopt.models.hotel.HotelRevenue method*), 70  
`check_simulatable_factors()` (*simopt.models.ironore.IronOre method*), 73  
`check_simulatable_factors()` (*simopt.models.mm1queue.MMIQueue method*), 86  
`check_simulatable_factors()` (*simopt.models.network.Network method*), 88  
`check_simulatable_factors()` (*simopt.models.paramesti.ParameterEstimation method*), 97  
`check_simulatable_factors()` (*simopt.models.rmitd.RMITD method*), 99  
`check_simulatable_factors()` (*simopt.models.sscont.SSCont method*), 110

`check_simulatable_factors()` (*simopt.models.tableallocation.TableAllocation method*), 116  
`check_solver_factor()` (*simopt.base.Solver method*), 151  
`check_solver_factors()` (*simopt.base.Solver method*), 151  
`check_st_dev()` (*simopt.models.dualsourcing.DualSourcing method*), 35  
`check_st_dev()` (*simopt.models.ironore.IronOre method*), 73  
`check_stages()` (*simopt.models.contam.Contamination method*), 23  
`check_step()` (*simopt.solvers.spsa.SPSA method*), 132  
`check_table_cap()` (*simopt.models.tableallocation.TableAllocation method*), 116  
`check_table_revenue()` (*simopt.models.tableallocation.TableAllocation method*), 116  
`check_theta()` (*simopt.solvers.aloe.ALOE method*), 124  
`check_time_before()` (*simopt.models.hotel.Hotel method*), 67  
`check_time_horizon()` (*simopt.models.rmtd.RMTD method*), 99  
`check_time_limit()` (*simopt.models.hotel.Hotel method*), 67  
`check_time_open()` (*simopt.models.amusementpark.AmusementPark method*), 6  
`check_transition_probabilities()` (*simopt.models.amusementpark.AmusementPark method*), 6  
`check_upper_limits_transit_time()` (*simopt.models.network.Network method*), 88  
`check_upper_thres()` (*simopt.models.contam.ContaminationTotalCostCont method*), 26  
`check_upper_thres()` (*simopt.models.contam.ContaminationTotalCostDisc method*), 31  
`check_upper_time()` (*simopt.models.chessmm.ChessAvgDifference method*), 13  
`check_variable_cost()` (*simopt.models.sscont.SSCont method*), 110  
`check_warmup()` (*simopt.models.mm1queue.MM1Queue method*), 87  
`check_warmup()` (*simopt.models.sscont.SSCont method*), 110  
`check_x()` (*simopt.models.example.ExampleModel method*), 46  
`check_x()` (*simopt.models.paramesti.ParameterEstimation method*), 97  
`check_xstar()` (*simopt.models.paramesti.ParameterEstimation method*), 97  
`checkbox_function2()` (*simopt.GUI.Experiment\_Window method*), 137  
`ChessAvgDifference` (class in *simopt.models.chessmm*), 11  
`ChessMatchmaking` (class in *simopt.models.chessmm*), 15  
`clear_meta_function()` (*simopt.GUI.Experiment\_Window method*), 137  
`clear_postnorm()` (*simopt.experiment\_base.ProblemSolver method*), 161  
`clear_postreplicate()` (*simopt.experiment\_base.ProblemSolver method*), 161  
`clear_queue` (*simopt.GUI.Experiment\_Window attribute*), 136  
`clear_queue()` (*simopt.GUI.Experiment\_Window method*), 137  
`clear_row()` (*simopt.GUI.Plot\_Window method*), 138  
`clear_run()` (*simopt.experiment\_base.ProblemSolver method*), 161  
`clearRow_function` (*simopt.GUI.Experiment\_Window attribute*), 136  
`clearRow_function()` (*simopt.GUI.Experiment\_Window method*), 137  
`CntNV` (class in *simopt.models.cntnv*), 16  
`CntNVMaxProfit` (class in *simopt.models.cntnv*), 18  
`compute_area_under_curve()` (*simopt.experiment\_base.Curve method*), 156  
`compute_bootstrap_CI()` (in module *simopt.experiment\_base*), 165  
`compute_crossing_time()` (*simopt.experiment\_base.Curve method*), 156  
`confirm_cross_design_function()` (*simopt.GUI.Cross\_Design\_Window method*), 135  
`confirm_oracle_factors` (*simopt.GUI.Experiment\_Window attribute*), 136  
`confirm_oracle_factors()` (*simopt.GUI.Experiment\_Window method*), 137  
`confirm_problem_factors` (*simopt.GUI.Experiment\_Window attribute*), 136  
`confirm_problem_factors()` (*simopt.GUI.Experiment\_Window method*), 136

[137](#)  
 confirm\_solver\_factors (simopt.GUI.Experiment\_Window attribute), [136](#)  
 confirm\_solver\_factors() (simopt.GUI.Experiment\_Window method), [137](#)  
 constraint\_type (simopt.base.Problem attribute), [142](#)  
 constraint\_type (simopt.base.Solver attribute), [150](#)  
 constraint\_type (simopt.models.amusementpark.AmusementParkMinDepart attribute), [7](#)  
 constraint\_type (simopt.models.chessmm.ChessAvgDifference attribute), [11](#)  
 constraint\_type (simopt.models.cntnv.CntNVMaxProfit attribute), [18](#)  
 constraint\_type (simopt.models.contam.ContaminationTotalCostCont attribute), [24](#)  
 constraint\_type (simopt.models.contam.ContaminationTotalCostDisc attribute), [29](#)  
 constraint\_type (simopt.models.dualsourcing.DualSourcingMinCost attribute), [36](#)  
 constraint\_type (simopt.models.dynamnews.DynamNewsMaxProfit attribute), [41](#)  
 constraint\_type (simopt.models.example.ExampleProblem attribute), [47](#)  
 constraint\_type (simopt.models.facilitysizing.FacilitySizingMaxSetback attribute), [53](#)  
 constraint\_type (simopt.models.facilitysizing.FacilitySizingTotalCost attribute), [57](#)  
 constraint\_type (simopt.models.fixedsan.FixedSANLongestPath attribute), [63](#)  
 constraint\_type (simopt.models.hotel.HotelRevenue attribute), [68](#)  
 constraint\_type (simopt.models.ironore.IronOreMaxRev attribute), [74](#)  
 constraint\_type (simopt.models.ironore.IronOreMaxRevCnt attribute), [78](#)  
 constraint\_type (simopt.models.mm1queue.MM1MinMeanSojournTime attribute), [82](#)  
 constraint\_type (simopt.models.network.NetworkMinTotalCost attribute), [89](#)  
 constraint\_type (simopt.models.paramesti.ParamEstiMaxLogLik attribute), [93](#)  
 constraint\_type (simopt.models.rmitd.RMITDMaxRevenue attribute), [100](#)  
 constraint\_type (simopt.models.san.SANLongestPath attribute), [105](#)  
 constraint\_type (simopt.models.sscont.SSContMinCost attribute), [111](#)  
 constraint\_type (simopt.models.tableallocation.TableAllocationMaxRev attribute), [117](#)  
 constraint\_type (simopt.solvers.adam.ADAM attribute), [121](#)  
 constraint\_type (simopt.solvers.aloe.ALOE attribute), [123](#)  
 constraint\_type (simopt.solvers.astrodf.ASTRODF attribute), [125](#)  
 constraint\_type (simopt.solvers.neldmd.NelderMead attribute), [127](#)  
 constraint\_type (simopt.solvers.randomsearch.RandomSearch attribute), [129](#)  
 constraint\_type (simopt.solvers.spsa.SPSA attribute), [131](#)  
 constraint\_type (simopt.solvers.strong.STRONG attribute), [133](#)  
 construct\_model() (simopt.solvers.astrodf.ASTRODF method), [126](#)  
 Contamination (class in simopt.models.contam), [22](#)  
 TotalCostCont (class in simopt.models.contam), [23](#)  
 TotalCostDisc (class in simopt.models.contam), [28](#)  
 new\_solution() (simopt.base.Solver method), [151](#)  
 crn\_max\_profit (simopt.experiment\_base.ProblemSolver attribute), [158](#)  
 crn\_across\_init\_opt (simopt.experiment\_base.ProblemSolver attribute), [159](#)  
 crossdesign\_macroreps (simopt.experiment\_base.ProblemSolver attribute), [158](#)  
 Cross\_Design\_Window (class in simopt.GUI), [135](#)  
 crossdesign\_function (simopt.GUI.Experiment\_Window attribute), [136](#)  
 crossdesign\_function() (simopt.GUI.Experiment\_Window method), [137](#)  
 curve\_to\_full\_curve() (simopt.experiment\_base.Curve method), [157](#)  
 curve\_to\_mesh() (simopt.experiment\_base.Curve method), [157](#)  
 DataFarmingExperiment (class in simopt.data\_farming\_base), [152](#)  
 DataFarmingMetaExperiment (class in simopt.data\_farming\_base), [153](#)  
 decision\_factors (simopt.base.Solution attribute), [148](#)  
 design (simopt.data\_farming\_base.DataFarmingExperiment attribute), [152](#)



design (*simopt.data\_farming\_base.DataFarmingMetaExperiment attribute*), 153  
 DesignPoint (*class in simopt.data\_farming\_base*), 155  
 det\_objectives (*simopt.base.Solution attribute*), 148  
 det\_objectives\_gradients (*simopt.base.Solution attribute*), 148  
 det\_stoch\_constraints (*simopt.base.Solution attribute*), 148  
 det\_stoch\_constraints\_gradients (*simopt.base.Solution attribute*), 148  
 deterministic\_objectives\_and\_gradients () (*simopt.base.Problem method*), 145  
 deterministic\_objectives\_and\_gradients () (*simopt.models.amusementpark.AmusementParkMinDepart method*), 9  
 deterministic\_objectives\_and\_gradients () (*simopt.models.chessmm.ChessAvgDifference method*), 13  
 deterministic\_objectives\_and\_gradients () (*simopt.models.cntnv.CntNVMaxProfit method*), 20  
 deterministic\_objectives\_and\_gradients () (*simopt.models.contam.ContaminationTotalCostCont method*), 26  
 deterministic\_objectives\_and\_gradients () (*simopt.models.contam.ContaminationTotalCostDisc method*), 31  
 deterministic\_objectives\_and\_gradients () (*simopt.models.dualsourcing.DualSourcingMinCost method*), 38  
 deterministic\_objectives\_and\_gradients () (*simopt.models.dynamnews.DynamNewsMaxProfit method*), 43  
 deterministic\_objectives\_and\_gradients () (*simopt.models.example.ExampleProblem method*), 49  
 deterministic\_objectives\_and\_gradients () (*simopt.models.facilitysizing.FacilitySizingMaxService method*), 55  
 deterministic\_objectives\_and\_gradients () (*simopt.models.facilitysizing.FacilitySizingTotalCost method*), 59  
 deterministic\_objectives\_and\_gradients () (*simopt.models.fixedsan.FixedSANLongestPath method*), 65  
 deterministic\_objectives\_and\_gradients () (*simopt.models.hotel.HotelRevenue method*), 70  
 deterministic\_objectives\_and\_gradients () (*simopt.models.ironore.IronOreMaxRev method*), 76  
 deterministic\_objectives\_and\_gradients () (*simopt.models.ironore.IronOreMaxRevCnt method*), 80  
 deterministic\_objectives\_and\_gradients () (*simopt.models.mm1queue.MMI MinMeanSojournTime method*), 84  
 deterministic\_objectives\_and\_gradients () (*simopt.models.network.NetworkMinTotalCost method*), 91  
 deterministic\_objectives\_and\_gradients () (*simopt.models.paramesti.ParamEstiMaxLogLik method*), 95  
 deterministic\_objectives\_and\_gradients () (*simopt.models.san.SANLongestPath method*), 107  
 deterministic\_objectives\_and\_gradients () (*simopt.models.sscont.SSContMinCost method*), 113  
 deterministic\_objectives\_and\_gradients () (*simopt.models.tableallocation.TableAllocationMaxRev method*), 119  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.base.Problem method*), 145  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.models.amusementpark.AmusementParkMinDepart method*), 9  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.models.chessmm.ChessAvgDifference method*), 14  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.models.cntnv.CntNVMaxProfit method*), 20  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.models.contam.ContaminationTotalCostCont method*), 26  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.models.contam.ContaminationTotalCostDisc method*), 31  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.models.dualsourcing.DualSourcingMinCost method*), 38  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.models.dynamnews.DynamNewsMaxProfit method*), 44  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.models.example.ExampleProblem method*), 49  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.models.facilitysizing.FacilitySizingMaxService method*), 55  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.models.facilitysizing.FacilitySizingTotalCost method*), 59  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.models.fixedsan.FixedSANLongestPath method*), 65  
 deterministic\_stochastic\_constraints\_and\_gradients (*simopt.models.hotel.HotelRevenue method*), 71

- p>deterministic\_stochastic\_constraints\_and\_gradients() (simopt.models.ironore.IronOreMaxRev method), 76
- 
- p>deterministic\_stochastic\_constraints\_and\_gradients() (simopt.models.ironore.IronOreMaxRevCnt method), 80
- 
- p>deterministic\_stochastic\_constraints\_and\_gradients() (simopt.models.mm1queue.MM1MinMeanSojournTime method), 84
- 
- p>deterministic\_stochastic\_constraints\_and\_gradients() (simopt.models.network.NetworkMinTotalCost method), 91
- 
- p>deterministic\_stochastic\_constraints\_and\_gradients() (simopt.models.san.SANLongestPath method), 107
- 
- p>deterministic\_stochastic\_constraints\_and\_gradients() (simopt.models.sscont.SSContMinCost method), 113
- 
- p>deterministic\_stochastic\_constraints\_and\_gradients() (simopt.models.tableallocation.TableAllocationMaxRev method), 119
- 
- p>dfs() (simopt.models.san.SAN method), 104
- 
- p>difference\_of\_curves() (in module simopt.experiment\_base), 166
- 
- p>dim(simopt.base.Problem attribute), 141
- 
- p>dim(simopt.base.Solution attribute), 148
- 
- p>dim(simopt.models.amusementpark.AmusementParkMinDepart attribute), 7
- 
- p>dim(simopt.models.chessmm.ChessAvgDifference attribute), 11
- 
- p>dim(simopt.models.cntnv.CntNVMaxProfit attribute), 18
- 
- p>dim(simopt.models.contam.ContaminationTotalCostCont attribute), 23
- 
- p>dim(simopt.models.contam.ContaminationTotalCostDisc attribute), 28
- 
- p>dim(simopt.models.dualsourcing.DualSourcingMinCost attribute), 36
- 
- p>dim(simopt.models.dynamnews.DynamNewsMaxProfit attribute), 41
- 
- p>dim(simopt.models.example.ExampleProblem attribute), 46
- 
- p>dim(simopt.models.facilitysizing.FacilitySizingMaxService attribute), 52
- 
- p>dim(simopt.models.facilitysizing.FacilitySizingTotalCost attribute), 56
- 
- p>dim(simopt.models.fixedsan.FixedSANLongestPath attribute), 62
- 
- p>dim(simopt.models.hotel.HotelRevenue attribute), 68
- 
- p>dim(simopt.models.ironore.IronOreMaxRev attribute), 74
- 
- p>dim(simopt.models.ironore.IronOreMaxRevCnt attribute), 78
- 
- p>dim(simopt.models.mm1queue.MM1MinMeanSojournTime attribute), 82
- 
- p>dim(simopt.models.network.NetworkMinTotalCost attribute), 89
- 
- p>dim(simopt.models.paramesti.ParamEstiMaxLogLik attribute), 93
- 
- p>dim(simopt.models.rmitd.RMITDMaxRevenue attribute), 99
- 
- p>dim(simopt.models.san.SANLongestPath attribute), 104
- 
- p>dim(simopt.models.sscont.SSContMinCost attribute), 111
- 
- p>dim(simopt.models.tableallocation.TableAllocationMaxRev attribute), 117
- 
- p>DualSourcing (class in simopt.models.dualsourcing), 33
- 
- p>DualSourcingMinCost (class in simopt.models.dualsourcing), 35
- 
- p>DynamNews (class in simopt.models.dynamnews), 39
- 
- p>DynamNewsMaxProfit (class in simopt.models.dynamnews), 41
- 
- ## E
- p>evaluate\_model() (simopt.solvers.astrodf.ASTRODF method), 126
- 
- p>ExampleModel (class in simopt.models.example), 45
- 
- p>ExampleProblem (class in simopt.models.example), 46
- 
- p>exit\_meta\_view() (simopt.GUI.Experiment\_Window method), 137
- 
- p>experiment\_master\_list (simopt.GUI.Experiment\_Window.self attribute), 135
- 
- p>experiment\_object\_list (simopt.GUI.Experiment\_Window.self attribute), 135
- 
- p>Experiment\_Window (class in simopt.GUI), 135
- 
- p>experiments (simopt.experiment\_base.ProblemsSolvers attribute), 163
- 
- ## F
- p>FacilitySize (class in simopt.models.facilitysizing), 50
- 
- p>FacilitySizingMaxService (class in simopt.models.facilitysizing), 52
- 
- p>FacilitySizingTotalCost (class in simopt.models.facilitysizing), 56
- 
- p>factor\_dict\_to\_vector() (simopt.base.Problem method), 145
- 
- p>factor\_dict\_to\_vector() (simopt.models.amusementpark.AmusementParkMinDepart method), 10
- 
- p>factor\_dict\_to\_vector() (simopt.models.chessmm.ChessAvgDifference method), 14
- 
- p>factor\_dict\_to\_vector() (simopt.models.cntnv.CntNVMaxProfit method), 20
- 
- p>factor\_dict\_to\_vector() (simopt.models.contam.ContaminationTotalCostCont method), 27

`factor_dict_to_vector()`  
     (*simopt.models.contam.ContaminationTotalCostDisc*  
     *method*), 31  
`factor_dict_to_vector()`  
     (*simopt.models.dualsourcing.DualSourcingMinCost*  
     *method*), 38  
`factor_dict_to_vector()`  
     (*simopt.models.dynamnews.DynamNewsMaxProfit*  
     *method*), 44  
`factor_dict_to_vector()`  
     (*simopt.models.example.ExampleProblem*  
     *method*), 49  
`factor_dict_to_vector()`  
     (*simopt.models.facilitysizing.FacilitySizingMaxService*  
     *method*), 55  
`factor_dict_to_vector()`  
     (*simopt.models.facilitysizing.FacilitySizingTotalCost*  
     *method*), 59  
`factor_dict_to_vector()`  
     (*simopt.models.fixedsan.FixedSANLongestPath*  
     *method*), 65  
`factor_dict_to_vector()`  
     (*simopt.models.hotel.HotelRevenue* *method*),  
     71  
`factor_dict_to_vector()`  
     (*simopt.models.ironore.IronOreMaxRev* *method*),  
     77  
`factor_dict_to_vector()`  
     (*simopt.models.ironore.IronOreMaxRevCnt*  
     *method*), 80  
`factor_dict_to_vector()`  
     (*simopt.models.mm1queue.MM1MinMeanSojournTime*  
     *method*), 84  
`factor_dict_to_vector()`  
     (*simopt.models.network.NetworkMinTotalCost*  
     *method*), 92  
`factor_dict_to_vector()`  
     (*simopt.models.paramesti.ParamEstiMaxLogLik*  
     *method*), 95  
`factor_dict_to_vector()`  
     (*simopt.models.rmitd.RMITDMaxRevenue*  
     *method*), 102  
`factor_dict_to_vector()`  
     (*simopt.models.san.SANLongestPath* *method*),  
     107  
`factor_dict_to_vector()`  
     (*simopt.models.sscont.SSContMinCost* *method*),  
     114  
`factor_dict_to_vector()`  
     (*simopt.models.tableallocation.TableAllocationMaxRev*  
     *method*), 120  
`factor_dict_to_vector_gradients()`  
     (*simopt.base.Problem* *method*), 145  
`factor_dict_to_vector_gradients()`  
     (*simopt.models.contam.ContaminationTotalCostCont*  
     *method*), 27  
`factor_dict_to_vector_gradients()`  
     (*simopt.models.contam.ContaminationTotalCostDisc*  
     *method*), 31  
`factor_dict_to_vector_gradients()`  
     (*simopt.models.facilitysizing.FacilitySizingTotalCost*  
     *method*), 59  
`factors` (*simopt.base.Model* *attribute*), 140  
`factors` (*simopt.base.Problem* *attribute*), 143  
`factors` (*simopt.base.Solver* *attribute*), 150  
`factors` (*simopt.models.amusementpark.AmusementPark*  
     *attribute*), 5  
`factors` (*simopt.models.amusementpark.AmusementParkMinDepart*  
     *attribute*), 8  
`factors` (*simopt.models.chessmm.ChessAvgDifference* *at*  
     *tribute*), 13  
`factors` (*simopt.models.chessmm.ChessMatchmaking* *at*  
     *tribute*), 15  
`factors` (*simopt.models.cntnv.CntNV* *attribute*), 16  
`factors` (*simopt.models.cntnv.CntNVMaxProfit* *at*  
     *tribute*), 19  
`factors` (*simopt.models.contam.Contamination* *at*  
     *tribute*), 22  
`factors` (*simopt.models.contam.ContaminationTotalCostCont*  
     *attribute*), 25  
`factors` (*simopt.models.contam.ContaminationTotalCostDisc*  
     *attribute*), 30  
`factors` (*simopt.models.dualsourcing.DualSourcing* *at*  
     *tribute*), 33  
`factors` (*simopt.models.dualsourcing.DualSourcingMinCost*  
     *attribute*), 37  
`factors` (*simopt.models.dynamnews.DynamNews*  
     *attribute*), 40  
`factors` (*simopt.models.dynamnews.DynamNewsMaxProfit*  
     *attribute*), 43  
`factors` (*simopt.models.example.ExampleModel* *at*  
     *tribute*), 45  
`factors` (*simopt.models.example.ExampleProblem*  
     *attribute*), 48  
`factors` (*simopt.models.facilitysizing.FacilitySize* *at*  
     *tribute*), 51  
`factors` (*simopt.models.facilitysizing.FacilitySizingMaxService*  
     *attribute*), 54  
`factors` (*simopt.models.facilitysizing.FacilitySizingTotalCost*  
     *attribute*), 58  
`factors` (*simopt.models.fixedsan.FixedSAN* *attribute*), 61  
`factors` (*simopt.models.fixedsan.FixedSANLongestPath*  
     *attribute*), 64  
`factors` (*simopt.models.hotel.Hotel* *attribute*), 67  
`factors` (*simopt.models.hotel.HotelRevenue* *attribute*), 69  
`factors` (*simopt.models.ironore.IronOre* *attribute*), 72  
`factors` (*simopt.models.ironore.IronOreMaxRev* *at*  
     *tribute*), 75

- factors (*simopt.models.ironore.IronOreMaxRevCnt* attribute), 79
- factors (*simopt.models.mm1queue.MM1MinMeanSojournTime* attribute), 83
- factors (*simopt.models.mm1queue.MM1Queue* attribute), 86
- factors (*simopt.models.network.Network* attribute), 87
- factors (*simopt.models.network.NetworkMinTotalCost* attribute), 91
- factors (*simopt.models.paramesti.ParamEstiMaxLogLik* attribute), 95
- factors (*simopt.models.paramesti.ParameterEstimation* attribute), 97
- factors (*simopt.models.rmitd.RMITD* attribute), 98
- factors (*simopt.models.rmitd.RMITDMaxRevenue* attribute), 101
- factors (*simopt.models.san.SAN* attribute), 103
- factors (*simopt.models.san.SANLongestPath* attribute), 106
- factors (*simopt.models.sscont.SSCont* attribute), 109
- factors (*simopt.models.sscont.SSContMinCost* attribute), 113
- factors (*simopt.models.tableallocation.TableAllocation* attribute), 115
- factors (*simopt.models.tableallocation.TableAllocationMaxRev* attribute), 118
- factors (*simopt.solvers.adam.ADAM* attribute), 122
- factors (*simopt.solvers.aloe.ALOE* attribute), 123
- factors (*simopt.solvers.astrodf.ASTRODF* attribute), 125
- factors (*simopt.solvers.neldmd.NelderMead* attribute), 128
- factors (*simopt.solvers.randomsearch.RandomSearch* attribute), 129
- factors (*simopt.solvers.spsa.SPSA* attribute), 131
- factors (*simopt.solvers.strong.STRONG* attribute), 133
- file\_name\_path (*simopt.experiment\_base.ProblemSolver* attribute), 158
- file\_name\_path (*simopt.experiment\_base.ProblemsSolvers* attribute), 163
- find\_missing\_experiments() (in module *simopt.experiment\_base*), 166
- find\_unique\_solvers\_problems() (in module *simopt.experiment\_base*), 166
- finite\_diff() (*simopt.solvers.adam.ADAM* method), 122
- finite\_diff() (*simopt.solvers.aloe.ALOE* method), 124
- finite\_diff() (*simopt.solvers.strong.STRONG* method), 134
- FixedSAN (class in *simopt.models.fixedsan*), 61
- FixedSANLongestPath (class in *simopt.models.fixedsan*), 62
- frame (*simopt.GUI.Experiment\_Window.self* attribute), 135
- functional\_of\_curves() (in module *simopt.experiment\_base*), 167
- main\_actions (*simopt.GUI.Experiment\_Window* attribute), 135
- ## G
- gen\_simul\_pert\_vec() (*simopt.solvers.spsa.SPSA* method), 132
- get\_coordinate\_basis\_interpolation\_points() (*simopt.solvers.astrodf.ASTRODF* method), 126
- get\_coordinate\_vector() (*simopt.solvers.astrodf.ASTRODF* method), 126
- get\_crossdesign\_MetaExperiment() (*simopt.GUI.Cross\_Design\_Window* method), 135
- get\_model\_coefficients() (*simopt.solvers.astrodf.ASTRODF* method), 126
- get\_parameters\_and\_settings() (*simopt.GUI.Plot\_Window* method), 138
- get\_random\_solution() (*simopt.base.Problem* method), 146
- get\_random\_solution() (*simopt.models.amusementpark.AmusementParkMinDepart* method), 10
- get\_random\_solution() (*simopt.models.chessmm.ChessAvgDifference* method), 14
- get\_random\_solution() (*simopt.models.cntnv.CntNVMaxProfit* method), 21
- get\_random\_solution() (*simopt.models.contam.ContaminationTotalCostCont* method), 27
- get\_random\_solution() (*simopt.models.contam.ContaminationTotalCostDisc* method), 32
- get\_random\_solution() (*simopt.models.dualsourcing.DualSourcingMinCost* method), 38
- get\_random\_solution() (*simopt.models.dynamnews.DynamNewsMaxProfit* method), 44
- get\_random\_solution() (*simopt.models.example.ExampleProblem* method), 49
- get\_random\_solution() (*simopt.models.facilitysizing.FacilitySizingMaxService* method), 55
- get\_random\_solution() (*simopt.models.facilitysizing.FacilitySizingTotalCost* method), 60
- get\_random\_solution()



<i>(simopt.models.fixedsan.FixedSANLongestPath method)</i> , 65	gradient_available <i>(simopt.models.contam.ContaminationTotalCostDisc attribute)</i> , 29
<i>get_random_solution()</i> <i>(simopt.models.hotel.HotelRevenue method)</i> , 71	gradient_available <i>(simopt.models.dualsourcing.DualSourcingMinCost attribute)</i> , 36
<i>get_random_solution()</i> <i>(simopt.models.ironore.IronOreMaxRev method)</i> , 77	gradient_available <i>(simopt.models.dynamnews.DynamNewsMaxProfit attribute)</i> , 42
<i>get_random_solution()</i> <i>(simopt.models.ironore.IronOreMaxRevCnt method)</i> , 80	gradient_available <i>(simopt.models.example.ExampleProblem attribute)</i> , 47
<i>get_random_solution()</i> <i>(simopt.models.mm1queue.MM1MinMeanSojournTime method)</i> , 84	gradient_available <i>(simopt.models.facilitysizing.FacilitySizingMaxService attribute)</i> , 53
<i>get_random_solution()</i> <i>(simopt.models.network.NetworkMinTotalCost method)</i> , 92	gradient_available <i>(simopt.models.facilitysizing.FacilitySizingTotalCost attribute)</i> , 57
<i>get_random_solution()</i> <i>(simopt.models.paramesti.ParamEstiMaxLogLik method)</i> , 96	gradient_available <i>(simopt.models.fixedsan.FixedSANLongestPath attribute)</i> , 63
<i>get_random_solution()</i> <i>(simopt.models.rmitd.RMITDMaxRevenue method)</i> , 102	gradient_available <i>(simopt.models.hotel.HotelRevenue attribute)</i> , 69
<i>get_random_solution()</i> <i>(simopt.models.san.SANLongestPath method)</i> , 107	gradient_available <i>(simopt.models.ironore.IronOreMaxRev attribute)</i> , 75
<i>get_random_solution()</i> <i>(simopt.models.sscont.SSContMinCost method)</i> , 114	gradient_available <i>(simopt.models.ironore.IronOreMaxRevCnt attribute)</i> , 78
<i>get_random_solution()</i> <i>(simopt.models.tableallocation.TableAllocationMaxRev method)</i> , 120	gradient_available <i>(simopt.models.mm1queue.MM1MinMeanSojournTime attribute)</i> , 82
<i>get_rotated_basis()</i> <i>(simopt.solvers.astrodf.ASTRODF method)</i> , 126	gradient_available <i>(simopt.models.network.NetworkMinTotalCost attribute)</i> , 90
<i>get_rotated_basis_interpolation_points()</i> <i>(simopt.solvers.astrodf.ASTRODF method)</i> , 126	gradient_available <i>(simopt.models.paramesti.ParamEstiMaxLogLik attribute)</i> , 94
<i>get_stopping_time()</i> <i>(simopt.solvers.astrodf.ASTRODF method)</i> , 126	gradient_available <i>(simopt.models.rmitd.RMITDMaxRevenue attribute)</i> , 100
gradient_available <i>(simopt.base.Problem attribute)</i> , 142	gradient_available <i>(simopt.models.san.SANLongestPath attribute)</i> , 105
gradient_available <i>(simopt.models.amusementpark.AmusementParkMinDepart attribute)</i> , 8	gradient_available <i>(simopt.models.sscont.SSContMinCost attribute)</i> , 112
gradient_available <i>(simopt.models.chessmm.ChessAvgDifference attribute)</i> , 12	gradient_available <i>(simopt.models.tableallocation.TableAllocationMaxRev attribute)</i> , 118
gradient_available <i>(simopt.models.cntnv.CntNVMaxProfit attribute)</i> , 19	gradient_needed <i>(simopt.base.Solver attribute)</i> , 150
gradient_available <i>(simopt.models.contam.ContaminationTotalCostCont attribute)</i> , 24	gradient_needed <i>(simopt.solvers.adam.ADAM attribute)</i> , 121
	gradient_needed <i>(simopt.solvers.aloe.ALOE attribute)</i> , 122

tribute), 123  
 gradient\_needed (*simopt.solvers.astrodf.ASTRODF attribute*), 125  
 gradient\_needed (*simopt.solvers.neldmd.NelderMead attribute*), 127  
 gradient\_needed (*simopt.solvers.randomsearch.RandomSearch attribute*), 129  
 gradient\_needed (*simopt.solvers.spsa.SPSA attribute*), 131  
 gradient\_needed (*simopt.solvers.strong.STRONG attribute*), 133  
 gradients (*simopt.data\_farming\_base.DesignPoint attribute*), 155

## H

Hotel (*class in simopt.models.hotel*), 66  
 HotelRevenue (*class in simopt.models.hotel*), 68

## I

IronOre (*class in simopt.models.ironore*), 72  
 IronOreMaxRev (*class in simopt.models.ironore*), 74  
 IronOreMaxRevCnt (*class in simopt.models.ironore*), 77  
 iterate() (*simopt.solvers.astrodf.ASTRODF method*), 126

## L

load\_pickle\_file\_function() (*simopt.GUI.Experiment\_Window method*), 137  
 log\_experiment\_results() (*simopt.experiment\_base.ProblemSolver method*), 161  
 log\_group\_experiment\_results() (*simopt.experiment\_base.ProblemsSolvers method*), 163  
 lookup() (*simopt.experiment\_base.Curve method*), 157  
 lower\_bounds (*simopt.base.Problem attribute*), 142  
 lower\_bounds (*simopt.models.chessmm.ChessAvgDifference attribute*), 12  
 lower\_bounds (*simopt.models.cntnv.CntNVMaxProfit attribute*), 18  
 lower\_bounds (*simopt.models.contam.ContaminationTotalCostCont attribute*), 24  
 lower\_bounds (*simopt.models.contam.ContaminationTotalCostDisc attribute*), 29  
 lower\_bounds (*simopt.models.dynamnews.DynamNewsMaxProfit attribute*), 42  
 lower\_bounds (*simopt.models.example.ExampleProblem attribute*), 47  
 lower\_bounds (*simopt.models.facilitysizing.FacilitySizingMaxService attribute*), 53  
 lower\_bounds (*simopt.models.facilitysizing.FacilitySizingTotalCost attribute*), 57

lower\_bounds (*simopt.models.fixedsan.FixedSANLongestPath attribute*), 63  
 lower\_bounds (*simopt.models.hotel.HotelRevenue attribute*), 68  
 lower\_bounds (*simopt.models.mm1queue.MM1MinMeanSojournTime attribute*), 82  
 lower\_bounds (*simopt.models.network.NetworkMinTotalCost attribute*), 90  
 lower\_bounds (*simopt.models.paramesti.ParamEstiMaxLogLik attribute*), 94  
 lower\_bounds (*simopt.models.rmitd.RMITDMaxRevenue attribute*), 100  
 lower\_bounds (*simopt.models.san.SANLongestPath attribute*), 105  
 lower\_bounds (*simopt.models.sscont.SSContMinCost attribute*), 112

## M

maco\_var (*simopt.GUI.Experiment\_Window.self attribute*), 135  
 main() (*in module simopt.GUI*), 139  
 make\_full\_metaexperiment() (*in module simopt.experiment\_base*), 167  
 make\_meta\_experiment\_func() (*simopt.GUI.Experiment\_Window method*), 137  
 max\_difference\_of\_curves() (*in module simopt.experiment\_base*), 168  
 mean\_of\_curves() (*in module simopt.experiment\_base*), 168  
 meta\_experiment\_problem\_solver\_list() (*simopt.GUI.Experiment\_Window method*), 137  
 minmax (*simopt.base.Problem attribute*), 142  
 minmax (*simopt.models.amusementpark.AmusementParkMinDepart attribute*), 7  
 minmax (*simopt.models.chessmm.ChessAvgDifference attribute*), 11  
 minmax (*simopt.models.cntnv.CntNVMaxProfit attribute*), 18  
 minmax (*simopt.models.contam.ContaminationTotalCostCont attribute*), 24  
 minmax (*simopt.models.contam.ContaminationTotalCostDisc attribute*), 29  
 minmax (*simopt.models.dualsourcing.DualSourcingMinCost attribute*), 36  
 minmax (*simopt.models.dynamnews.DynamNewsMaxProfit attribute*), 41  
 minmax (*simopt.models.example.ExampleProblem attribute*), 47  
 minmax (*simopt.models.facilitysizing.FacilitySizingMaxService attribute*), 52  
 minmax (*simopt.models.facilitysizing.FacilitySizingTotalCost attribute*), 57

minmax (*simopt.models.fixedsan.FixedSANLongestPath attribute*), 62  
 minmax (*simopt.models.hotel.HotelRevenue attribute*), 68  
 minmax (*simopt.models.ironore.IronOreMaxRev attribute*), 74  
 minmax (*simopt.models.ironore.IronOreMaxRevCnt attribute*), 78  
 minmax (*simopt.models.mm1queue.MM1MinMeanSojournTime attribute*), 82  
 minmax (*simopt.models.network.NetworkMinTotalCost attribute*), 89  
 minmax (*simopt.models.paramesti.ParamEstiMaxLogLik attribute*), 93  
 minmax (*simopt.models.rmitd.RMITDMaxRevenue attribute*), 100  
 minmax (*simopt.models.san.SANLongestPath attribute*), 104  
 minmax (*simopt.models.sscont.SSContMinCost attribute*), 111  
 minmax (*simopt.models.tableallocation.TableAllocationMaxRev attribute*), 117  
 MM1MinMeanSojournTime (class in *simopt.models.mm1queue*), 81  
 MM1Queue (class in *simopt.models.mm1queue*), 85  
 Model (class in *simopt.base*), 140  
 model (*simopt.base.Problem attribute*), 142  
 model (*simopt.data\_farming\_base.DataFarmingExperiment attribute*), 152  
 model (*simopt.data\_farming\_base.DesignPoint attribute*), 155  
 model (*simopt.models.amusementpark.AmusementParkMinDepart attribute*), 8  
 model (*simopt.models.chessmm.ChessAvgDifference attribute*), 12  
 model (*simopt.models.cntnv.CntNVMaxProfit attribute*), 19  
 model (*simopt.models.contam.ContaminationTotalCostCont attribute*), 25  
 model (*simopt.models.contam.ContaminationTotalCostDisc attribute*), 30  
 model (*simopt.models.dualsourcing.DualSourcingMinCost attribute*), 37  
 model (*simopt.models.dynamnews.DynamNewsMaxProfit attribute*), 42  
 model (*simopt.models.example.ExampleProblem attribute*), 48  
 model (*simopt.models.facilitysizing.FacilitySizingMaxService attribute*), 53  
 model (*simopt.models.facilitysizing.FacilitySizingTotalCost attribute*), 57  
 model (*simopt.models.fixedsan.FixedSANLongestPath attribute*), 63  
 model (*simopt.models.hotel.HotelRevenue attribute*), 69  
 model (*simopt.models.ironore.IronOreMaxRev attribute*), 75  
 model (*simopt.models.ironore.IronOreMaxRevCnt attribute*), 79  
 model (*simopt.models.mm1queue.MM1MinMeanSojournTime attribute*), 83  
 model (*simopt.models.network.NetworkMinTotalCost attribute*), 90  
 model (*simopt.models.paramesti.ParamEstiMaxLogLik attribute*), 94  
 model (*simopt.models.rmitd.RMITDMaxRevenue attribute*), 101  
 model (*simopt.models.san.SANLongestPath attribute*), 105  
 model (*simopt.models.sscont.SSContMinCost attribute*), 112  
 model (*simopt.models.tableallocation.TableAllocationMaxRev attribute*), 118  
 model\_decision\_factors (*simopt.base.Problem attribute*), 143  
 model\_decision\_factors (*simopt.models.amusementpark.AmusementParkMinDepart attribute*), 8  
 model\_decision\_factors (*simopt.models.cntnv.CntNVMaxProfit attribute*), 19  
 model\_decision\_factors (*simopt.models.dualsourcing.DualSourcingMinCost attribute*), 37  
 model\_decision\_factors (*simopt.models.dynamnews.DynamNewsMaxProfit attribute*), 43  
 model\_decision\_factors (*simopt.models.example.ExampleProblem attribute*), 48  
 model\_decision\_factors (*simopt.models.facilitysizing.FacilitySizingMaxService attribute*), 54  
 model\_decision\_factors (*simopt.models.facilitysizing.FacilitySizingTotalCost attribute*), 58  
 model\_decision\_factors (*simopt.models.fixedsan.FixedSANLongestPath attribute*), 64  
 model\_decision\_factors (*simopt.models.hotel.HotelRevenue attribute*), 69  
 model\_decision\_factors (*simopt.models.ironore.IronOreMaxRev attribute*), 75  
 model\_decision\_factors (*simopt.models.ironore.IronOreMaxRevCnt attribute*), 79  
 model\_decision\_factors (*simopt.models.mm1queue.MM1MinMeanSojournTime attribute*), 83  
 model\_decision\_factors

<i>(simopt.models.network.NetworkMinTotalCost attribute)</i> , 90	<i>model_default_factors (simopt.models.ironore.IronOreMaxRevCnt attribute)</i> , 79
<i>model_decision_factors (simopt.models.rmitd.RMITDMaxRevenue attribute)</i> , 101	<i>model_default_factors (simopt.models.mm1queue.MMI MinMeanSojournTime attribute)</i> , 83
<i>model_decision_factors (simopt.models.san.SANLongestPath attribute)</i> , 106	<i>model_default_factors (simopt.models.network.NetworkMinTotalCost attribute)</i> , 90
<i>model_decision_factors (simopt.models.sscont.SSContMinCost attribute)</i> , 112	<i>model_default_factors (simopt.models.paramesti.ParamEstiMaxLogLik attribute)</i> , 94
<i>model_decision_factors (simopt.models.tableallocation.TableAllocationMaxRev attribute)</i> , 118	<i>model_default_factors (simopt.models.rmitd.RMITDMaxRevenue attribute)</i> , 101
<i>model_default_factors (simopt.base.Problem attribute)</i> , 142	<i>model_default_factors (simopt.models.san.SANLongestPath attribute)</i> , 105
<i>model_default_factors (simopt.models.amusementpark.AmusementParkMinDepart attribute)</i> , 8	<i>model_default_factors (simopt.models.sscont.SSContMinCost attribute)</i> , 112
<i>model_default_factors (simopt.models.chessmm.ChessAvgDifference attribute)</i> , 12	<i>model_default_factors (simopt.models.tableallocation.TableAllocationMaxRev attribute)</i> , 118
<i>model_default_factors (simopt.models.cntnv.CntNVMaxProfit attribute)</i> , 19	<i>model_factors (simopt.data_farming_base.DesignPoint attribute)</i> , 155
<i>model_default_factors (simopt.models.contam.ContaminationTotalCostCont attribute)</i> , 25	<i>model_fixed_factors (simopt.base.Problem attribute)</i> , 143
<i>model_default_factors (simopt.models.contam.ContaminationTotalCostDisc attribute)</i> , 30	<i>model_fixed_factors (simopt.models.amusementpark.AmusementParkMinDepart attribute)</i> , 8
<i>model_default_factors (simopt.models.dualsourcing.DualSourcingMinCost attribute)</i> , 37	<i>model_fixed_factors (simopt.models.chessmm.ChessAvgDifference attribute)</i> , 12
<i>model_default_factors (simopt.models.dynamnews.DynamNewsMaxProfit attribute)</i> , 42	<i>model_fixed_factors (simopt.models.cntnv.CntNVMaxProfit attribute)</i> , 19
<i>model_default_factors (simopt.models.example.ExampleProblem attribute)</i> , 48	<i>model_fixed_factors (simopt.models.contam.ContaminationTotalCostCont attribute)</i> , 25
<i>model_default_factors (simopt.models.facilitysizing.FacilitySizingMaxService attribute)</i> , 53	<i>model_fixed_factors (simopt.models.contam.ContaminationTotalCostDisc attribute)</i> , 30
<i>model_default_factors (simopt.models.facilitysizing.FacilitySizingTotalCost attribute)</i> , 58	<i>model_fixed_factors (simopt.models.dualsourcing.DualSourcingMinCost attribute)</i> , 37
<i>model_default_factors (simopt.models.fixedsan.FixedSANLongestPath attribute)</i> , 63	<i>model_fixed_factors (simopt.models.dynamnews.DynamNewsMaxProfit attribute)</i> , 42
<i>model_default_factors (simopt.models.hotel.HotelRevenue attribute)</i> , 69	<i>model_fixed_factors (simopt.models.example.ExampleProblem attribute)</i> , 48
<i>model_default_factors (simopt.models.ironore.IronOreMaxRev attribute)</i> , 75	<i>model_fixed_factors (simopt.models.facilitysizing.FacilitySizingMaxService attribute)</i> , 53



attribute), 54  
 model\_fixed\_factors  
   (simopt.models.facilitysizing.FacilitySizingTotalCost  
   attribute), 58  
 model\_fixed\_factors  
   (simopt.models.fixedsan.FixedSANLongestPath  
   attribute), 64  
 model\_fixed\_factors  
   (simopt.models.hotel.HotelRevenue attribute), 69  
 model\_fixed\_factors  
   (simopt.models.ironore.IronOreMaxRev attribute), 75  
 model\_fixed\_factors  
   (simopt.models.ironore.IronOreMaxRevCnt  
   attribute), 79  
 model\_fixed\_factors  
   (simopt.models.mm1queue.MM1MinMeanSojournTime  
   attribute), 83  
 model\_fixed\_factors  
   (simopt.models.network.NetworkMinTotalCost  
   attribute), 90  
 model\_fixed\_factors  
   (simopt.models.paramesti.ParamEstiMaxLogLik  
   attribute), 94  
 model\_fixed\_factors  
   (simopt.models.rmitd.RMITDMaxRevenue  
   attribute), 101  
 model\_fixed\_factors  
   (simopt.models.san.SANLongestPath attribute),  
   106  
 model\_fixed\_factors  
   (simopt.models.sscont.SSContMinCost attribute),  
   112  
 model\_fixed\_factors  
   (simopt.models.tableallocation.TableAllocationMaxRev  
   attribute), 118  
 module  
   simopt, 175  
   simopt.base, 140  
   simopt.data\_farming\_base, 152  
   simopt.directory, 156  
   simopt.experiment\_base, 156  
   simopt.GUI, 135  
   simopt.models, 121  
   simopt.models.amusementpark, 5  
   simopt.models.chessmm, 11  
   simopt.models.cntnv, 16  
   simopt.models.contam, 22  
   simopt.models.dualsourcing, 33  
   simopt.models.dynamnews, 39  
   simopt.models.example, 45  
   simopt.models.facilitysizing, 50  
   simopt.models.fixedsan, 61  
   simopt.models.hotel, 66

simopt.models.ironore, 72  
 simopt.models.mm1queue, 81  
 simopt.models.network, 87  
 simopt.models.paramesti, 93  
 simopt.models.rmitd, 98  
 simopt.models.san, 103  
 simopt.models.sscont, 108  
 simopt.models.tableallocation, 115  
 simopt.solvers, 135  
 simopt.solvers.adam, 121  
 simopt.solvers.aloe, 123  
 simopt.solvers.astrodf, 125  
 simopt.solvers.neldmd, 127  
 simopt.solvers.randomsearch, 129  
 simopt.solvers.spsa, 130  
 simopt.solvers.strong, 133

## N

n\_design\_pts(simopt.data\_farming\_base.DataFarmingExperiment  
   attribute), 153  
 n\_design\_pts(simopt.data\_farming\_base.DataFarmingMetaExperiment  
   attribute), 153  
 n\_macroreps (simopt.experiment\_base.ProblemSolver  
   attribute), 158  
 n\_objectives (simopt.base.Problem attribute), 141  
 n\_objectives (simopt.models.amusementpark.AmusementParkMinDepart  
   attribute), 7  
 n\_objectives (simopt.models.chessmm.ChessAvgDifference  
   attribute), 11  
 n\_objectives (simopt.models.cntnv.CntNVMaxProfit  
   attribute), 18  
 n\_objectives (simopt.models.contam.ContaminationTotalCostCont  
   attribute), 23  
 n\_objectives (simopt.models.contam.ContaminationTotalCostDisc  
   attribute), 28  
 n\_objectives (simopt.models.dualsourcing.DualSourcingMinCost  
   attribute), 36  
 n\_objectives (simopt.models.dynamnews.DynamNewsMaxProfit  
   attribute), 41  
 n\_objectives (simopt.models.example.ExampleProblem  
   attribute), 47  
 n\_objectives (simopt.models.facilitysizing.FacilitySizingMaxService  
   attribute), 52  
 n\_objectives (simopt.models.facilitysizing.FacilitySizingTotalCost  
   attribute), 56  
 n\_objectives (simopt.models.fixedsan.FixedSANLongestPath  
   attribute), 62  
 n\_objectives (simopt.models.hotel.HotelRevenue at-  
   tribute), 68  
 n\_objectives (simopt.models.ironore.IronOreMaxRev  
   attribute), 74  
 n\_objectives (simopt.models.ironore.IronOreMaxRevCnt  
   attribute), 78

n\_objectives (*simopt.models.mm1queue.MM1MinMeanSojournTime* attribute), 103  
 attribute), 82  
 n\_objectives (*simopt.models.network.NetworkMinTotalCost* attribute), 89  
 n\_objectives (*simopt.models.paramesti.ParamEstiMaxLogLik* attribute), 93  
 n\_objectives (*simopt.models.rmitd.RMITDMaxRevenue* attribute), 100  
 n\_objectives (*simopt.models.san.SANLongestPath* attribute), 104  
 n\_objectives (*simopt.models.sscont.SSContMinCost* attribute), 111  
 n\_objectives (*simopt.models.tableallocation.TableAllocationMaxRevenue* attribute), 117  
 n\_points (*simopt.experiment\_base.Curve* attribute), 156  
 n\_postreps (*simopt.experiment\_base.ProblemSolver* attribute), 158  
 n\_postreps\_init\_opt (*simopt.experiment\_base.ProblemSolver* attribute), 159  
 n\_problems (*simopt.experiment\_base.ProblemsSolvers* attribute), 162  
 n\_reps (*simopt.base.Solution* attribute), 148  
 n\_reps (*simopt.data\_farming\_base.DesignPoint* attribute), 155  
 n\_responses (*simopt.base.Model* attribute), 140  
 n\_responses (*simopt.models.amusementpark.AmusementPark* attribute), 5  
 n\_responses (*simopt.models.chessmm.ChessMatchmaking* attribute), 15  
 n\_responses (*simopt.models.cntnv.CntNV* attribute), 16  
 n\_responses (*simopt.models.contam.Contamination* attribute), 22  
 n\_responses (*simopt.models.dualsourcing.DualSourcing* attribute), 33  
 n\_responses (*simopt.models.dynamnews.DynamNews* attribute), 40  
 n\_responses (*simopt.models.example.ExampleModel* attribute), 45  
 n\_responses (*simopt.models.facilitysizing.FacilitySize* attribute), 51  
 n\_responses (*simopt.models.fixedsan.FixedSAN* attribute), 61  
 n\_responses (*simopt.models.hotel.Hotel* attribute), 66  
 n\_responses (*simopt.models.ironore.IronOre* attribute), 72  
 n\_responses (*simopt.models.mm1queue.MM1Queue* attribute), 86  
 n\_responses (*simopt.models.network.Network* attribute), 87  
 n\_responses (*simopt.models.paramesti.ParameterEstimation* attribute), 97  
 n\_responses (*simopt.models.rmitd.RMITD* attribute), 98  
 n\_responses (*simopt.models.san.SAN* attribute), 103  
 n\_responses (*simopt.models.sscont.SSCont* attribute), 108  
 n\_responses (*simopt.models.tableallocation.TableAllocation* attribute), 115  
 n\_rngs (*simopt.base.Model* attribute), 140  
 n\_rngs (*simopt.models.amusementpark.AmusementPark* attribute), 5  
 n\_rngs (*simopt.models.chessmm.ChessMatchmaking* attribute), 15  
 n\_rngs (*simopt.models.cntnv.CntNV* attribute), 16  
 n\_rngs (*simopt.models.contam.Contamination* attribute), 22  
 n\_rngs (*simopt.models.dualsourcing.DualSourcing* attribute), 33  
 n\_rngs (*simopt.models.dynamnews.DynamNews* attribute), 39  
 n\_rngs (*simopt.models.example.ExampleModel* attribute), 45  
 n\_rngs (*simopt.models.facilitysizing.FacilitySize* attribute), 51  
 n\_rngs (*simopt.models.fixedsan.FixedSAN* attribute), 61  
 n\_rngs (*simopt.models.hotel.Hotel* attribute), 66  
 n\_rngs (*simopt.models.ironore.IronOre* attribute), 72  
 n\_rngs (*simopt.models.mm1queue.MM1Queue* attribute), 86  
 n\_rngs (*simopt.models.network.Network* attribute), 87  
 n\_rngs (*simopt.models.paramesti.ParameterEstimation* attribute), 96  
 n\_rngs (*simopt.models.rmitd.RMITD* attribute), 98  
 n\_rngs (*simopt.models.san.SAN* attribute), 103  
 n\_rngs (*simopt.models.sscont.SSCont* attribute), 108  
 n\_rngs (*simopt.models.tableallocation.TableAllocation* attribute), 115  
 n\_solvers (*simopt.experiment\_base.ProblemsSolvers* attribute), 162  
 n\_stochastic\_constraints (*simopt.base.Problem* attribute), 141  
 n\_stochastic\_constraints (*simopt.models.amusementpark.AmusementParkMinDepart* attribute), 7  
 n\_stochastic\_constraints (*simopt.models.chessmm.ChessAvgDifference* attribute), 11  
 n\_stochastic\_constraints (*simopt.models.cntnv.CntNVMaxProfit* attribute), 18  
 n\_stochastic\_constraints (*simopt.models.contam.ContaminationTotalCostCont* attribute), 24  
 n\_stochastic\_constraints (*simopt.models.contam.ContaminationTotalCostDisc* attribute), 29  
 n\_stochastic\_constraints

	( <i>simopt.models.dualsourcing.DualSourcingMinCost</i> attribute), 36		tribute), 11
n_stochastic_constraints	( <i>simopt.models.dynamnews.DynamNewsMaxProfit</i> attribute), 41	name	( <i>simopt.models.chessmm.ChessMatchmaking</i> attribute), 15
n_stochastic_constraints	( <i>simopt.models.example.ExampleProblem</i> attribute), 47	name	( <i>simopt.models.cntnv.CntNV</i> attribute), 16
n_stochastic_constraints	( <i>simopt.models.facilitysizing.FacilitySizingMaxService</i> attribute), 52	name	( <i>simopt.models.cntnv.CntNVMaxProfit</i> attribute), 18
n_stochastic_constraints	( <i>simopt.models.facilitysizing.FacilitySizingTotalCost</i> attribute), 56	name	( <i>simopt.models.contam.Contamination</i> attribute), 22
n_stochastic_constraints	( <i>simopt.models.fixedsan.FixedSANLongestPath</i> attribute), 62	name	( <i>simopt.models.contam.ContaminationTotalCostCont</i> attribute), 23
n_stochastic_constraints	( <i>simopt.models.hotel.HotelRevenue</i> attribute), 68	name	( <i>simopt.models.contam.ContaminationTotalCostDisc</i> attribute), 28
n_stochastic_constraints	( <i>simopt.models.ironore.IronOreMaxRev</i> attribute), 74	name	( <i>simopt.models.dualsourcing.DualSourcing</i> attribute), 33
n_stochastic_constraints	( <i>simopt.models.ironore.IronOreMaxRevCnt</i> attribute), 78	name	( <i>simopt.models.dualsourcing.DualSourcingMinCost</i> attribute), 35
n_stochastic_constraints	( <i>simopt.models.mm1queue.MMI MinMeanSojournTime</i> attribute), 82	name	( <i>simopt.models.dynamnews.DynamNews</i> attribute), 39
n_stochastic_constraints	( <i>simopt.models.network.NetworkMinTotalCost</i> attribute), 89	name	( <i>simopt.models.dynamnews.DynamNewsMaxProfit</i> attribute), 41
n_stochastic_constraints	( <i>simopt.models.paramesti.ParamEstiMaxLogLik</i> attribute), 93	name	( <i>simopt.models.example.ExampleModel</i> attribute), 45
n_stochastic_constraints	( <i>simopt.models.rmitd.RMITDMaxRevenue</i> attribute), 100	name	( <i>simopt.models.example.ExampleProblem</i> attribute), 46
n_stochastic_constraints	( <i>simopt.models.san.SANLongestPath</i> attribute), 104	name	( <i>simopt.models.facilitysizing.FacilitySize</i> attribute), 50
n_stochastic_constraints	( <i>simopt.models.sscont.SSContMinCost</i> attribute), 111	name	( <i>simopt.models.facilitysizing.FacilitySizingMaxService</i> attribute), 52
n_stochastic_constraints	( <i>simopt.models.tableallocation.TableAllocationMaxRevenue</i> attribute), 117	name	( <i>simopt.models.facilitysizing.FacilitySizingTotalCost</i> attribute), 56
name	( <i>simopt.base.Model</i> attribute), 140	name	( <i>simopt.models.fixedsan.FixedSAN</i> attribute), 61
name	( <i>simopt.base.Problem</i> attribute), 141	name	( <i>simopt.models.fixedsan.FixedSANLongestPath</i> attribute), 62
name	( <i>simopt.base.Solver</i> attribute), 150	name	( <i>simopt.models.hotel.Hotel</i> attribute), 66
name	( <i>simopt.models.amusementpark.AmusementPark</i> attribute), 5	name	( <i>simopt.models.hotel.HotelRevenue</i> attribute), 68
name	( <i>simopt.models.amusementpark.AmusementParkMinDepot</i> attribute), 7	name	( <i>simopt.models.ironore.IronOre</i> attribute), 72
name	( <i>simopt.models.chessmm.ChessAvgDifference</i> attribute), 11	name	( <i>simopt.models.ironore.IronOreMaxRev</i> attribute), 74
		name	( <i>simopt.models.ironore.IronOreMaxRevCnt</i> attribute), 78
		name	( <i>simopt.models.mm1queue.MMI MinMeanSojournTime</i> attribute), 81
		name	( <i>simopt.models.mm1queue.MMI Queue</i> attribute), 85
		name	( <i>simopt.models.network.Network</i> attribute), 87
		name	( <i>simopt.models.network.NetworkMinTotalCost</i> attribute), 89
		name	( <i>simopt.models.paramesti.ParamEstiMaxLogLik</i> attribute), 93
		name	( <i>simopt.models.paramesti.ParameterEstimation</i> attribute), 96
		name	( <i>simopt.models.rmitd.RMITD</i> attribute), 98
		name	( <i>simopt.models.rmitd.RMITDMaxRevenue</i> attribute), 99
		name	( <i>simopt.models.san.SAN</i> attribute), 103
		name	( <i>simopt.models.san.SANLongestPath</i> attribute), 104
		name	( <i>simopt.models.sscont.SSCont</i> attribute), 108
		name	( <i>simopt.models.sscont.SSContMinCost</i> attribute), 111
		name	( <i>simopt.models.tableallocation.TableAllocation</i> attribute), 117

tribute), 115

name (*simopt.models.tableallocation.TableAllocationMaxRevenue* onFrameConfigure\_queue attribute), 117

name (*simopt.solvers.adam.ADAM* attribute), 121

name (*simopt.solvers.aloe.ALOE* attribute), 123

name (*simopt.solvers.astrodf.ASTRODF* attribute), 125

name (*simopt.solvers.neldmd.NelderMead* attribute), 127

name (*simopt.solvers.randomsearch.RandomSearch* attribute), 129

name (*simopt.solvers.spsa.SPSA* attribute), 130

name (*simopt.solvers.strong.STRONG* attribute), 133

NelderMead (class in *simopt.solvers.neldmd*), 127

Network (class in *simopt.models.network*), 87

NetworkMinTotalCost (class in *simopt.models.network*), 89

## O

objective\_curves (*simopt.experiment\_base.ProblemSolver* attribute), 159

objective\_type (*simopt.base.Solver* attribute), 150

objective\_type (*simopt.solvers.adam.ADAM* attribute), 121

objective\_type (*simopt.solvers.aloe.ALOE* attribute), 123

objective\_type (*simopt.solvers.astrodf.ASTRODF* attribute), 125

objective\_type (*simopt.solvers.neldmd.NelderMead* attribute), 127

objective\_type (*simopt.solvers.randomsearch.RandomSearch* attribute), 129

objective\_type (*simopt.solvers.spsa.SPSA* attribute), 130

objective\_type (*simopt.solvers.strong.STRONG* attribute), 133

objectives (*simopt.base.Solution* attribute), 149

objectives\_gradients (*simopt.base.Solution* attribute), 149

onFrameConfigure\_factor\_oracle (*simopt.GUI.Experiment\_Window* attribute), 137

onFrameConfigure\_factor\_oracle () (*simopt.GUI.Experiment\_Window* method), 137

onFrameConfigure\_factor\_problem (*simopt.GUI.Experiment\_Window* attribute), 137

onFrameConfigure\_factor\_problem () (*simopt.GUI.Experiment\_Window* method), 137

onFrameConfigure\_factor\_solver (*simopt.GUI.Experiment\_Window* attribute), 137

onFrameConfigure\_factor\_solver () (*simopt.GUI.Experiment\_Window* method), 137

onFrameConfigure\_queue (*simopt.GUI.Experiment\_Window* attribute), 137

onFrameConfigure\_queue () (*simopt.GUI.Experiment\_Window* method), 137

optimal\_solution (*simopt.base.Problem* attribute), 142

optimal\_solution (*simopt.models.amusementpark.AmusementParkMinDepa* attribute), 8

optimal\_solution (*simopt.models.chessmm.ChessAvgDifference* attribute), 12

optimal\_solution (*simopt.models.cntnv.CntNVMaxProfit* attribute), 19

optimal\_solution (*simopt.models.contam.ContaminationTotalCostCont* attribute), 24

optimal\_solution (*simopt.models.contam.ContaminationTotalCostDisc* attribute), 29

optimal\_solution (*simopt.models.dualsourcing.DualSourcingMinCost* attribute), 36

optimal\_solution (*simopt.models.dynamnews.DynamNewsMaxProfit* attribute), 42

optimal\_solution (*simopt.models.example.ExampleProblem* attribute), 48

optimal\_solution (*simopt.models.facilitysizing.FacilitySizingMaxService* attribute), 53

optimal\_solution (*simopt.models.facilitysizing.FacilitySizingTotalCost* attribute), 57

optimal\_solution (*simopt.models.fixedsan.FixedSANLongestPath* attribute), 63

optimal\_solution (*simopt.models.hotel.HotelRevenue* attribute), 69

optimal\_solution (*simopt.models.ironore.IronOreMaxRev* attribute), 75

optimal\_solution (*simopt.models.ironore.IronOreMaxRevCnt* attribute), 79

optimal\_solution (*simopt.models.mm1queue.MM1MinMeanSojournTim* attribute), 83

optimal\_solution (*simopt.models.network.NetworkMinTotalCost* attribute), 90

optimal\_solution (*simopt.models.paramesti.ParamEstiMaxLogLik* attribute), 94

optimal\_solution (*simopt.models.rmitd.RMITDMaxRevenue* attribute), 101

optimal\_solution (*simopt.models.san.SANLongestPath* attribute), 105

optimal\_solution (*simopt.models.sscont.SSContMinCost* attribute), 112

optimal\_solution (*simopt.models.tableallocation.TableAllocationMaxR* attribute), 118

optimal\_value (*simopt.base.Problem* attribute), 142

optimal\_value (*simopt.models.amusementpark.AmusementParkMinDepa* attribute), 8



optimal_value (simopt.models.chessmm.ChessAvgDifference	plot_meta_function()	
attribute), 12	(simopt.GUI.Experiment_Window	method),
optimal_value (simopt.models.cntnv.CntNVMaxProfit	138	
attribute), 19	plot_progress_curves()	(in module
optimal_value (simopt.models.contam.ContaminationTotalCostConsimopt.experiment_base), 169		
attribute), 24	plot_solvability_cdfs()	(in module
optimal_value (simopt.models.contam.ContaminationTotalCostDiscsimopt.experiment_base), 169		
attribute), 29	plot_solvability_profiles()	(in module
optimal_value (simopt.models.dualsourcing.DualSourcingMinCost simopt.experiment_base), 170		
attribute), 36	plot_terminal_progress()	(in module
optimal_value (simopt.models.dynamnews.DynamNewsMaxProfit simopt.experiment_base), 171		
attribute), 42	plot_terminal_scatterplots()	(in module
optimal_value (simopt.models.example.ExampleProblem simopt.experiment_base), 171		
attribute), 47	Plot_Window (class in simopt.GUI), 138	
optimal_value (simopt.models.facilitysizing.FacilitySizingMaxService simopt.experiment_base), 171		
attribute), 53	post_norm_return_func()	(simopt.GUI.Experiment_Window method),
optimal_value (simopt.models.facilitysizing.FacilitySizingTotalCost 138		
attribute), 57	post_norm_run_function()	
optimal_value (simopt.models.fixedsan.FixedSANLongestPath (simopt.GUI.Post_Normal_Window	method),	
attribute), 63	139	
optimal_value (simopt.models.hotel.HotelRevenue at-tribute), 69	post_norm_setup()	(simopt.GUI.Experiment_Window method),
optimal_value (simopt.models.ironore.IronOreMaxRev 138		
attribute), 75	post_normal_all_function()	
optimal_value (simopt.models.ironore.IronOreMaxRevCnt (simopt.GUI.Experiment_Window	method),	
attribute), 78	138	
optimal_value (simopt.models.mm1queue.MM1MinMeanSojournTime simopt.experiment_base), 171		
attribute), 83	post_normalize()	(in module
optimal_value (simopt.models.network.NetworkMinTotalCost simopt.experiment_base), 171		
attribute), 90	post_normalize() (simopt.data_farming_base.DataFarmingMetaExper	
optimal_value (simopt.models.paramesti.ParamEstiMaxLogLik method), 154		
attribute), 94	post_normalize() (simopt.experiment_base.ProblemsSolvers	
optimal_value (simopt.models.rmitd.RMITDMaxRevenue method), 163		
attribute), 100	post_process_disable_button()	
optimal_value (simopt.models.san.SANLongestPath (simopt.GUI.Experiment_Window	method),	
attribute), 105	138	
optimal_value (simopt.models.sscont.SSContMinCost post_processing_run_function()		
attribute), 112	(simopt.GUI.Post_Processing_Window	method),
optimal_value (simopt.models.tableallocation.TableAllocationMaxRev 139		
attribute), 118	Post_Processing_Window (class in simopt.GUI),	
	139	
	post_rep_function()	
pad_storage() (simopt.base.Solution method), 149	(simopt.GUI.Experiment_Window	method),
ParamEstiMaxLogLik (class in	138	
simopt.models.paramesti), 93	post_rep_meta_function()	
ParameterEstimation (class in	(simopt.GUI.Experiment_Window	method),
simopt.models.paramesti), 96	138	
plot() (simopt.experiment_base.Curve method), 157	post_replicate() (simopt.data_farming_base.DataFarmingMetaExper	
plot_area_scatterplots() (in module	method), 154	
simopt.experiment_base), 168	post_replicate() (simopt.experiment_base.ProblemSolver	
plot_bootstrap_CIs() (in module	method), 161	
simopt.experiment_base), 169	post_replicate() (simopt.experiment_base.ProblemsSolvers	
plot_button() (simopt.GUI.Plot_Window method),	method), 164	
138	print_to_csv() (simopt.data_farming_base.DataFarmingExperiment	

- method), 153
- Problem (class in *simopt.base*), 141
- problem (*simopt.experiment\_base.ProblemSolver* attribute), 158
- problem\_names (*simopt.experiment\_base.ProblemsSolvers* attribute), 162
- problem\_solver\_abbreviated\_name\_to\_unabbreviated\_name() (in module *simopt.GUI*), 139
- problem\_solver\_unabbreviated\_to\_object() (in module *simopt.GUI*), 139
- problem\_var (*simopt.GUI.Experiment\_Window* self attribute), 135
- problems (*simopt.experiment\_base.ProblemsSolvers* attribute), 162
- ProblemSolver (class in *simopt.experiment\_base*), 157
- ProblemsSolvers (class in *simopt.experiment\_base*), 161
- progress\_bar\_test() (*simopt.GUI.Experiment\_Window* method), 138
- progress\_curves (*simopt.experiment\_base.ProblemSolver* attribute), 159
- ## Q
- quantile\_cross\_jump() (in module *simopt.experiment\_base*), 172
- quantile\_of\_curves() (in module *simopt.experiment\_base*), 172
- ## R
- RandomSearch (class in *simopt.solvers.randomsearch*), 129
- read\_experiment\_results() (in module *simopt.experiment\_base*), 172
- read\_group\_experiment\_results() (in module *simopt.experiment\_base*), 173
- rebase() (*simopt.base.Solver* method), 152
- recompute\_summary\_statistics() (*simopt.base.Solution* method), 149
- record\_experiment\_results() (*simopt.experiment\_base.ProblemSolver* method), 161
- record\_group\_experiment\_results() (*simopt.experiment\_base.ProblemsSolvers* method), 164
- replicate() (*simopt.base.Model* method), 141
- replicate() (*simopt.models.amusementpark.AmusementPark* method), 6
- replicate() (*simopt.models.chessmm.ChessMatchmaking* method), 16
- replicate() (*simopt.models.cntnv.CntNV* method), 17
- replicate() (*simopt.models.contam.Contamination* method), 23
- replicate() (*simopt.models.dualsourcing.DualSourcing* method), 35
- replicate() (*simopt.models.dynamnews.DynamNews* method), 41
- replicate() (*simopt.models.example.ExampleModel* method), 46
- replicate() (*simopt.models.facilitysizing.FacilitySize* method), 52
- replicate() (*simopt.models.fixedsan.FixedSAN* method), 62
- replicate() (*simopt.models.hotel.Hotel* method), 67
- replicate() (*simopt.models.ironore.IronOre* method), 73
- replicate() (*simopt.models.mm1queue.MM1Queue* method), 87
- replicate() (*simopt.models.network.Network* method), 88
- replicate() (*simopt.models.paramesti.ParameterEstimation* method), 97
- replicate() (*simopt.models.rmitd.RMITD* method), 99
- replicate() (*simopt.models.san.SAN* method), 104
- replicate() (*simopt.models.sscont.SSCont* method), 110
- replicate() (*simopt.models.tableallocation.TableAllocation* method), 116
- report\_max\_halfwidth() (in module *simopt.experiment\_base*), 173
- report\_statistics() (*simopt.data\_farming\_base.DataFarmingMetaExperiment* method), 154
- response\_dict\_to\_objectives() (*simopt.base.Problem* method), 146
- response\_dict\_to\_objectives() (*simopt.models.amusementpark.AmusementParkMinDepart* method), 10
- response\_dict\_to\_objectives() (*simopt.models.chessmm.ChessAvgDifference* method), 14
- response\_dict\_to\_objectives() (*simopt.models.cntnv.CntNVMaxProfit* method), 21
- response\_dict\_to\_objectives() (*simopt.models.contam.ContaminationTotalCostCont* method), 27
- response\_dict\_to\_objectives() (*simopt.models.contam.ContaminationTotalCostDisc* method), 32
- response\_dict\_to\_objectives() (*simopt.models.dualsourcing.DualSourcingMinCost* method), 39
- response\_dict\_to\_objectives() (*simopt.models.dynamnews.DynamNewsMaxProfit* method), 44
- response\_dict\_to\_objectives()

```

        (simopt.models.example.ExampleProblem
        method), 50
response_dict_to_objectives()
    (simopt.models.facilitysizing.FacilitySizingMaxService
    method), 55
response_dict_to_objectives()
    (simopt.models.facilitysizing.FacilitySizingTotalCost
    method), 60
response_dict_to_objectives()
    (simopt.models.fixedsan.FixedSANLongestPath
    method), 65
response_dict_to_objectives()
    (simopt.models.hotel.HotelRevenue
    method), 71
response_dict_to_objectives()
    (simopt.models.ironore.IronOreMaxRev
    method), 77
response_dict_to_objectives()
    (simopt.models.ironore.IronOreMaxRevCnt
    method), 81
response_dict_to_objectives()
    (simopt.models.mm1queue.MM1MinMeanSojournTime
    method), 85
response_dict_to_objectives()
    (simopt.models.network.NetworkMinTotalCost
    method), 92
response_dict_to_objectives()
    (simopt.models.paramesti.ParamEstiMaxLogLik
    method), 96
response_dict_to_objectives()
    (simopt.models.rmtd.RMTDMaxRevenue
    method), 102
response_dict_to_objectives()
    (simopt.models.san.SANLongestPath
    method), 107
response_dict_to_objectives()
    (simopt.models.sscont.SSContMinCost
    method), 114
response_dict_to_objectives()
    (simopt.models.tableallocation.TableAllocationMaxRev
    method), 120
response_dict_to_objectives_gradients()
    (simopt.base.Problem
    method), 146
response_dict_to_objectives_gradients()
    (simopt.models.contam.ContaminationTotalCostCont
    method), 27
response_dict_to_objectives_gradients()
    (simopt.models.contam.ContaminationTotalCostDisc
    method), 32
response_dict_to_objectives_gradients()
    (simopt.models.facilitysizing.FacilitySizingTotalCost
    method), 60
response_dict_to_stoch_constraints()
    (simopt.base.Problem
    method), 146
response_dict_to_stoch_constraints()
    (simopt.models.amusementpark.AmusementParkMinDepart
    method), 10
response_dict_to_stoch_constraints()
    (simopt.models.chessmm.ChessAvgDifference
    method), 14
response_dict_to_stoch_constraints()
    (simopt.models.cntnv.CntNVMaxProfit
    method), 21
response_dict_to_stoch_constraints()
    (simopt.models.contam.ContaminationTotalCostCont
    method), 28
response_dict_to_stoch_constraints()
    (simopt.models.contam.ContaminationTotalCostDisc
    method), 32
response_dict_to_stoch_constraints()
    (simopt.models.dualsourcing.DualSourcingMinCost
    method), 39
response_dict_to_stoch_constraints()
    (simopt.models.dynamnews.DynamNewsMaxProfit
    method), 44
response_dict_to_stoch_constraints()
    (simopt.models.example.ExampleProblem
    method), 50
response_dict_to_stoch_constraints()
    (simopt.models.facilitysizing.FacilitySizingMaxService
    method), 56
response_dict_to_stoch_constraints()
    (simopt.models.facilitysizing.FacilitySizingTotalCost
    method), 60
response_dict_to_stoch_constraints()
    (simopt.models.fixedsan.FixedSANLongestPath
    method), 66
response_dict_to_stoch_constraints()
    (simopt.models.hotel.HotelRevenue
    method), 71
response_dict_to_stoch_constraints()
    (simopt.models.ironore.IronOreMaxRev
    method), 77
response_dict_to_stoch_constraints()
    (simopt.models.ironore.IronOreMaxRevCnt
    method), 81
response_dict_to_stoch_constraints()
    (simopt.models.mm1queue.MM1MinMeanSojournTime
    method), 85
response_dict_to_stoch_constraints()
    (simopt.models.network.NetworkMinTotalCost
    method), 92
response_dict_to_stoch_constraints()
    (simopt.models.san.SANLongestPath
    method), 108
response_dict_to_stoch_constraints()
    (simopt.models.sscont.SSContMinCost
    method), 114
response_dict_to_stoch_constraints()
    (simopt.base.Problem
    method), 146

```

(simopt.models.tableallocation.TableAllocationMaxRevenue attribute), 120  
 responses (simopt.data\_farming\_base.DesignPoint attribute), 155  
 RMITD (class in simopt.models.rmitd), 98  
 RMITDMaxRevenue (class in simopt.models.rmitd), 99  
 rng\_list (simopt.base.Problem attribute), 143  
 rng\_list (simopt.base.Solution attribute), 148  
 rng\_list (simopt.base.Solver attribute), 150  
 rng\_list (simopt.data\_farming\_base.DesignPoint attribute), 155  
 rng\_list (simopt.models.amusementpark.AmusementParkMinDepot attribute), 8  
 rng\_list (simopt.models.chessmm.ChessAvgDifference attribute), 12  
 rng\_list (simopt.models.cntnv.CntNVMaxProfit attribute), 19  
 rng\_list (simopt.models.contam.ContaminationTotalCostCont attribute), 25  
 rng\_list (simopt.models.contam.ContaminationTotalCostDisc attribute), 30  
 rng\_list (simopt.models.dualsourcing.DualSourcingMinCost attribute), 37  
 rng\_list (simopt.models.dynamnews.DynamNewsMaxProfit attribute), 43  
 rng\_list (simopt.models.example.ExampleProblem attribute), 48  
 rng\_list (simopt.models.facilitysizing.FacilitySizingMaxService attribute), 54  
 rng\_list (simopt.models.facilitysizing.FacilitySizingTotalCost attribute), 58  
 rng\_list (simopt.models.fixedsan.FixedSANLongestPath attribute), 64  
 rng\_list (simopt.models.hotel.HotelRevenue attribute), 69  
 rng\_list (simopt.models.ironore.IronOreMaxRev attribute), 75  
 rng\_list (simopt.models.ironore.IronOreMaxRevCnt attribute), 79  
 rng\_list (simopt.models.mm1queue.MM1MinMeanSojournTime attribute), 83  
 rng\_list (simopt.models.network.NetworkMinTotalCost attribute), 90  
 rng\_list (simopt.models.paramesti.ParamEstiMaxLogLik attribute), 94  
 rng\_list (simopt.models.rmitd.RMITDMaxRevenue attribute), 101  
 rng\_list (simopt.models.san.SANLongestPath attribute), 106  
 rng\_list (simopt.models.sscont.SSContMinCost attribute), 112  
 rng\_list (simopt.models.tableallocation.TableAllocationMaxRevenue attribute), 118  
 rng\_list (simopt.solvers.adam.ADAM attribute), 122  
 rng\_list (simopt.solvers.aloe.ALOE attribute), 124  
 rng\_list (simopt.solvers.astrodf.ASTRODF attribute), 126  
 rng\_list (simopt.solvers.neldmd.NelderMead attribute), 128  
 rng\_list (simopt.solvers.randomsearch.RandomSearch attribute), 130  
 rng\_list (simopt.solvers.spsa.SPSA attribute), 131  
 rng\_list (simopt.solvers.strong.STRONG attribute), 134  
 run () (simopt.data\_farming\_base.DataFarmingExperiment method), 153  
 run () (simopt.data\_farming\_base.DataFarmingMetaExperiment method), 154  
 run () (simopt.experiment\_base.ProblemSolver method), 161  
 run () (simopt.experiment\_base.ProblemsSolvers method), 164  
 run\_meta\_function () (simopt.GUI.Experiment\_Window method), 138  
 run\_row\_function () (simopt.GUI.Experiment\_Window method), 138  
 run\_single\_function (simopt.GUI.Experiment\_Window attribute), 136

## S

SAN (class in simopt.models.san), 103  
 SANLongestPath (class in simopt.models.san), 104  
 save\_edit\_function () (simopt.GUI.Experiment\_Window method), 138  
 save\_plot () (in module simopt.experiment\_base), 173  
 select\_pickle\_file\_fuction () (simopt.GUI.Experiment\_Window method), 138  
 setup\_plot () (in module simopt.experiment\_base), 174  
 show\_problem\_factors (simopt.GUI.Experiment\_Window attribute), 136  
 show\_problem\_factors () (simopt.GUI.Experiment\_Window method), 138  
 show\_problem\_factors2 () (simopt.GUI.Experiment\_Window method), 138  
 show\_solver\_factors (simopt.GUI.Experiment\_Window attribute), 136  
 show\_solver\_factors () (simopt.GUI.Experiment\_Window method), 138



---

```

show_solver_factors2()
    (simopt.GUI.Experiment_Window method), 138
simopt
    module, 175
simopt.base
    module, 140
simopt.data_farming_base
    module, 152
simopt.directory
    module, 156
simopt.experiment_base
    module, 156
simopt.GUI
    module, 135
simopt.models
    module, 121
simopt.models.amusementpark
    module, 5
simopt.models.chessmm
    module, 11
simopt.models.cntnv
    module, 16
simopt.models.contam
    module, 22
simopt.models.dualsourcing
    module, 33
simopt.models.dynamnews
    module, 39
simopt.models.example
    module, 45
simopt.models.facilitysizing
    module, 50
simopt.models.fixedsan
    module, 61
simopt.models.hotel
    module, 66
simopt.models.ironore
    module, 72
simopt.models.mm1queue
    module, 81
simopt.models.network
    module, 87
simopt.models.paramesti
    module, 93
simopt.models.rmitd
    module, 98
simopt.models.san
    module, 103
simopt.models.sscont
    module, 108
simopt.models.tableallocation
    module, 115
simopt.solvers
    module, 135
simopt.solvers.adam
    module, 121
simopt.solvers.aloe
    module, 123
simopt.solvers.astrodf
    module, 125
simopt.solvers.neldmd
    module, 127
simopt.solvers.randomsearch
    module, 129
simopt.solvers.spsa
    module, 130
simopt.solvers.strong
    module, 133
simulate() (simopt.base.Problem method), 147
simulate() (simopt.data_farming_base.DesignPoint method), 155
simulate_up_to() (simopt.base.Problem method), 147
Solution (class in simopt.base), 147
solution_progenitor_rngs (simopt.base.Solver attribute), 150
solve() (simopt.base.Solver method), 152
solve() (simopt.solvers.adam.ADAM method), 122
solve() (simopt.solvers.aloe.ALOE method), 124
solve() (simopt.solvers.astrodf.ASTRODF method), 126
solve() (simopt.solvers.neldmd.NelderMead method), 128
solve() (simopt.solvers.randomsearch.RandomSearch method), 130
solve() (simopt.solvers.spsa.SPSA method), 132
solve() (simopt.solvers.strong.STRONG method), 134
Solver (class in simopt.base), 150
solver (simopt.experiment_base.ProblemSolver attribute), 157
solver_names (simopt.experiment_base.ProblemsSolvers attribute), 162
solver_select_function()
    (simopt.GUI.Plot_Window method), 139
solver_var (simopt.GUI.Experiment_Window.self attribute), 135
solvers (simopt.experiment_base.ProblemsSolvers attribute), 162
sort_and_end_update()
    (simopt.solvers.neldmd.NelderMead method), 129
specifications (simopt.base.Model attribute), 140
specifications (simopt.base.Problem attribute), 143
specifications (simopt.base.Solver attribute), 150
specifications (simopt.models.amusementpark.AmusementPark attribute), 6
specifications (simopt.models.amusementpark.AmusementParkMinDep attribute), 9

```

---

specifications (*simopt.models.chessmm.ChessAvgDifference* attribute), 13

specifications (*simopt.models.chessmm.ChessMatchmaking* attribute), 15

specifications (*simopt.models.cntnv.CntNV* attribute), 17

specifications (*simopt.models.cntnv.CntNVMaxProfit* attribute), 20

specifications (*simopt.models.contam.Contamination* attribute), 22

specifications (*simopt.models.contam.ContaminationTotalCost* attribute), 25

specifications (*simopt.models.contam.ContaminationTotalCostDisc* attribute), 30

specifications (*simopt.models.dualsourcing.DualSourcing* attribute), 33

specifications (*simopt.models.dualsourcing.DualSourcingMinCost* attribute), 37

specifications (*simopt.models.dynamnews.DynamNews* attribute), 40

specifications (*simopt.models.dynamnews.DynamNewsMaxProfit* attribute), 43

specifications (*simopt.models.example.ExampleModel* attribute), 45

specifications (*simopt.models.example.ExampleProblem* attribute), 48

specifications (*simopt.models.facilitysizing.FacilitySizes* attribute), 51

specifications (*simopt.models.facilitysizing.FacilitySizingMaxService* attribute), 54

specifications (*simopt.models.facilitysizing.FacilitySizingTotalCost* attribute), 58

specifications (*simopt.models.fixedsan.FixedSAN* attribute), 61

specifications (*simopt.models.fixedsan.FixedSANLongestPath* attribute), 64

specifications (*simopt.models.hotel.Hotel* attribute), 67

specifications (*simopt.models.hotel.HotelRevenue* attribute), 70

specifications (*simopt.models.ironore.IronOre* attribute), 72

specifications (*simopt.models.ironore.IronOreMaxRev* attribute), 76

specifications (*simopt.models.ironore.IronOreMaxRevCnt* attribute), 79

specifications (*simopt.models.mm1queue.MM1MinMeanSojournTime* attribute), 83

specifications (*simopt.models.mm1queue.MM1Queue* attribute), 86

specifications (*simopt.models.network.Network* attribute), 88

specifications (*simopt.models.network.NetworkMinTotalCost* attribute), 91

specifications (*simopt.models.paramesti.ParamEstiMaxLogLik* attribute), 95

specifications (*simopt.models.paramesti.ParameterEstimation* attribute), 97

specifications (*simopt.models.rmitd.RMITD* attribute), 98

specifications (*simopt.models.rmitd.RMITDMaxRevenue* attribute), 101

specifications (*simopt.models.san.SAN* attribute), 103

specifications (*simopt.models.san.SANLongestPath* attribute), 106

specifications (*simopt.models.sscont.SSCont* attribute), 109

specifications (*simopt.models.sscont.SSContMinCost* attribute), 113

specifications (*simopt.models.tableallocation.TableAllocation* attribute), 115

specifications (*simopt.models.tableallocation.TableAllocationMaxRev* attribute), 119

specifications (*simopt.solvers.adam.ADAM* attribute), 122

specifications (*simopt.solvers.aloe.ALOE* attribute), 123

specifications (*simopt.solvers.astrodf.ASTRODF* attribute), 126

specifications (*simopt.solvers.neldmd.NelderMead* attribute), 128

specifications (*simopt.solvers.randomsearch.RandomSearch* attribute), 129

specifications (*simopt.solvers.spsa.SPSA* attribute), 131

specifications (*simopt.solvers.strong.STRONG* attribute), 133

*SPSA* (class in *simopt.solvers.spsa*), 130

*SSCont* (class in *simopt.models.sscont*), 108

*SSContMinCost* (class in *simopt.models.sscont*), 111

*stochastic\_constraints* (*simopt.base.Solution* attribute), 149

*stochastic\_constraints\_gradients* (*simopt.base.Solution* attribute), 149

*storage\_size* (*simopt.base.Solution* attribute), 148

*STRONG* (class in *simopt.solvers.strong*), 133

*TableAllocation* (class in *simopt.models.tableallocation*), 115

*TableAllocationMaxRev* (class in *simopt.models.tableallocation*), 117

*test\_function* (*simopt.GUI.Experiment\_Window* attribute), 137

*test\_function()* (*simopt.GUI.Cross\_Design\_Window* method), 135

test\_function2() (simopt.GUI.Post\_Normal\_Window method), 139

test\_function2() (simopt.GUI.Post\_Processing\_Window method), 139

timings (simopt.experiment\_base.ProblemSolver attribute), 158

trim\_solver\_results() (in module simopt.experiment\_base), 174

## U

update\_problem\_list\_compatibility() (simopt.GUI.Experiment\_Window method), 138

upper\_bounds (simopt.base.Problem attribute), 142

upper\_bounds (simopt.models.chessmm.ChessAvgDifference attribute), 12

upper\_bounds (simopt.models.cntnv.CntNVMaxProfit attribute), 19

upper\_bounds (simopt.models.contam.ContaminationTotalCostCont attribute), 24

upper\_bounds (simopt.models.contam.ContaminationTotalCostDisc attribute), 29

upper\_bounds (simopt.models.dynamnews.DynamNewsMaxProfit attribute), 42

upper\_bounds (simopt.models.example.ExampleProblem attribute), 47

upper\_bounds (simopt.models.facilitysizing.FacilitySizingMaxService attribute), 53

upper\_bounds (simopt.models.facilitysizing.FacilitySizingTotalCost attribute), 57

upper\_bounds (simopt.models.fixedsan.FixedSANLongestPath attribute), 63

upper\_bounds (simopt.models.hotel.HotelRevenue attribute), 68

upper\_bounds (simopt.models.ironore.IronOreMaxRev attribute), 74

upper\_bounds (simopt.models.ironore.IronOreMaxRevCnt attribute), 78

upper\_bounds (simopt.models.mm1queue.MM1MinMeanSojournTime attribute), 82

upper\_bounds (simopt.models.network.NetworkMinTotalCost attribute), 89

upper\_bounds (simopt.models.paramesti.ParamEstiMaxLogLik attribute), 93

upper\_bounds (simopt.models.rmitd.RMITDMaxRevenue attribute), 100

upper\_bounds (simopt.models.san.SANLongestPath attribute), 105

upper\_bounds (simopt.models.sscont.SSContMinCost attribute), 112

upper\_bounds (simopt.models.network.NetworkMinTotalCost attribute), 90

upper\_bounds (simopt.models.paramesti.ParamEstiMaxLogLik attribute), 94

upper\_bounds (simopt.models.rmitd.RMITDMaxRevenue attribute), 100

upper\_bounds (simopt.models.san.SANLongestPath attribute), 105

upper\_bounds (simopt.models.sscont.SSContMinCost attribute), 112

## V

variable\_type (simopt.base.Problem attribute), 142

variable\_type (simopt.base.Solver attribute), 150

variable\_type (simopt.models.amusementpark.AmusementParkMinDepart attribute), 8

variable\_type (simopt.models.chessmm.ChessAvgDifference attribute), 11

variable\_type (simopt.models.cntnv.CntNVMaxProfit attribute), 18

variable\_type (simopt.models.contam.ContaminationTotalCostCont attribute), 24

variable\_type (simopt.models.contam.ContaminationTotalCostDisc attribute), 29

variable\_type (simopt.models.dualsourcing.DualSourcingMinCost attribute), 36

variable\_type (simopt.models.dynamnews.DynamNewsMaxProfit attribute), 42

variable\_type (simopt.models.example.ExampleProblem attribute), 47

variable\_type (simopt.models.facilitysizing.FacilitySizingMaxService attribute), 53

variable\_type (simopt.models.facilitysizing.FacilitySizingTotalCost attribute), 57

variable\_type (simopt.models.fixedsan.FixedSANLongestPath attribute), 63

variable\_type (simopt.models.hotel.HotelRevenue attribute), 68

variable\_type (simopt.models.ironore.IronOreMaxRev attribute), 74

variable\_type (simopt.models.ironore.IronOreMaxRevCnt attribute), 78

variable\_type (simopt.models.mm1queue.MM1MinMeanSojournTime attribute), 82

variable\_type (simopt.models.network.NetworkMinTotalCost attribute), 89

variable\_type (simopt.models.paramesti.ParamEstiMaxLogLik attribute), 93

variable\_type (simopt.models.rmitd.RMITDMaxRevenue attribute), 100

variable\_type (simopt.models.san.SANLongestPath attribute), 105

variable\_type (simopt.models.sscont.SSContMinCost attribute), 111

variable\_type (simopt.models.tableallocation.TableAllocationMaxRev attribute), 118

variable\_type (simopt.solvers.adam.ADAM attribute), 121

variable\_type (simopt.solvers.aloe.ALOE attribute), 123

variable\_type (simopt.solvers.astrodf.ASTRODF attribute), 125

variable\_type (simopt.solvers.neldmd.NelderMead attribute), 127

variable\_type (simopt.solvers.randomsearch.RandomSearch attribute), 129

variable\_type (simopt.solvers.spsa.SPSA attribute), 131

variable\_type (simopt.solvers.strong.STRONG attribute), 133

vector\_to\_factor\_dict() (simopt.base.Problem method), 147

<code>vector_to_factor_dict()</code> ( <i>simopt.models.amusementpark.AmusementParkMinDepart</i> <i>method</i> ), 10	<code>vector_to_factor_dict()</code> ( <i>simopt.models.san.SANLongestPath</i> <i>method</i> ), 108
<code>vector_to_factor_dict()</code> ( <i>simopt.models.chessmm.ChessAvgDifference</i> <i>method</i> ), 15	<code>vector_to_factor_dict()</code> ( <i>simopt.models.sscont.SSContMinCost</i> <i>method</i> ), 114
<code>vector_to_factor_dict()</code> ( <i>simopt.models.cntnv.CntNVMaxProfit</i> <i>method</i> ), 21	<code>vector_to_factor_dict()</code> ( <i>simopt.models.tableallocation.TableAllocationMaxRev</i> <i>method</i> ), 120
<code>vector_to_factor_dict()</code> ( <i>simopt.models.contam.ContaminationTotalCostCont</i> <i>method</i> ), 28	<code>view_meta_function()</code> ( <i>simopt.GUI.Experiment_Window</i> <i>method</i> ), 138
<code>vector_to_factor_dict()</code> ( <i>simopt.models.contam.ContaminationTotalCostDisc</i> <i>method</i> ), 33	<code>view_one_pot()</code> ( <i>simopt.GUI.Plot_Window</i> <i>method</i> ), 139
<code>vector_to_factor_dict()</code> ( <i>simopt.models.dualsourcing.DualSourcingMinCost</i> <i>method</i> ), 39	<code>viewEdit_function()</code> ( <i>simopt.GUI.Experiment_Window</i> <i>method</i> ), 138
<code>vector_to_factor_dict()</code> ( <i>simopt.models.dynamnews.DynamNewsMaxProfit</i> <i>method</i> ), 45	<b>W</b> <code>widget_list</code> ( <i>simopt.GUI.Experiment_Window.self</i> <i>attribute</i> ), 135
<code>vector_to_factor_dict()</code> ( <i>simopt.models.example.ExampleProblem</i> <i>method</i> ), 50	<b>X</b> <code>x</code> ( <i>simopt.base.Solution</i> <i>attribute</i> ), 148
<code>vector_to_factor_dict()</code> ( <i>simopt.models.facilitysizing.FacilitySizingMaxService</i> <i>method</i> ), 56	<code>x0</code> ( <i>simopt.experiment_base.ProblemSolver</i> <i>attribute</i> ), 159
<code>vector_to_factor_dict()</code> ( <i>simopt.models.facilitysizing.FacilitySizingTotalCost</i> <i>method</i> ), 61	<code>x0_postreps</code> ( <i>simopt.experiment_base.ProblemSolver</i> <i>attribute</i> ), 159
<code>vector_to_factor_dict()</code> ( <i>simopt.models.fixedsan.FixedSANLongestPath</i> <i>method</i> ), 66	<code>x_vals</code> ( <i>simopt.experiment_base.Curve</i> <i>attribute</i> ), 156
<code>vector_to_factor_dict()</code> ( <i>simopt.models.hotel.HotelRevenue</i> <i>method</i> ), 72	<code>xstar</code> ( <i>simopt.experiment_base.ProblemSolver</i> <i>attribute</i> ), 159
<code>vector_to_factor_dict()</code> ( <i>simopt.models.ironore.IronOreMaxRev</i> <i>method</i> ), 77	<code>xstar_postreps</code> ( <i>simopt.experiment_base.ProblemSolver</i> <i>attribute</i> ), 159
<code>vector_to_factor_dict()</code> ( <i>simopt.models.ironore.IronOreMaxRevCnt</i> <i>method</i> ), 81	<b>Y</b> <code>y_vals</code> ( <i>simopt.experiment_base.Curve</i> <i>attribute</i> ), 156
<code>vector_to_factor_dict()</code> ( <i>simopt.models.mm1queue.MM1MinMeanSojournTime</i> <i>method</i> ), 85	
<code>vector_to_factor_dict()</code> ( <i>simopt.models.network.NetworkMinTotalCost</i> <i>method</i> ), 92	
<code>vector_to_factor_dict()</code> ( <i>simopt.models.paramesti.ParamEstiMaxLogLik</i> <i>method</i> ), 96	
<code>vector_to_factor_dict()</code> ( <i>simopt.models.rmitd.RMITDMaxRevenue</i> <i>method</i> ), 102	