SimOpt

Release 1.0

simopt-admin

Dec 06, 2022

CONTENTS

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.

CHAPTER

ONE

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.

CHAPTER

TWO

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 overriden 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] <= 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 overriden 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] <= 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 overriden 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] <= 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 overriden 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_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] <= 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 overriden 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 (*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] <= 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-1'*, *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 overriden 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] <= 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 overriden 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] <= 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 "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 overriden 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] <= 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 toal 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 overriden 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] <= 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 overriden 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] <= 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 overriden 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] <= 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 overriden 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
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
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] <= 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 overriden 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] <= 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 overriden 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] <= 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 overriden 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] <= 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 overriden 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] <= 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 overriden 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 (*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

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 overriden 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 overriden 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] <= 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 occured, and average amount ordered given an order occured.

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 occured

avg_stockout
mean amount of product backordered given a stockout occured

avg_order
mean amount of product ordered given an order occured

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 overriden 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] <= 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 arrvial 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 overriden 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] <= 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='RNDSRCH'*, *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 simulatanious pertubation 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 (candiate_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 weigthed 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 contraint 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 <- ttk.OptionMenu

Type
displays additional information on problem and oracle factors

show_solver_factors(*self*, *args*)

connected to : self.solver_menu <- ttk.OptionMenu

Type
displays additional information on solver factors

run_single_function(*self*, *args*)

connected to : self.run_button <- ttk.Button

Type
completes single-object experiment and invokes Post_Processing_Window class

crossdesign_function(*self*)

connected to : self.crossdesign_button <- ttk.Button

Type
invokes Cross_Design_Window class

clearRow_function(*self*)

connected to : self.clear_button_added <- 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 <- ttk.Button

Type
clears entire experiment queue and resets all lists containing experiment data

add_experiment(*self*)

connected to : self.add_button <- 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_compatability()

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 overriden 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] <= 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 evalaute.
- **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 deterministics 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.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 subsubstream 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 x0 and optimal x*.
- **crn_across_init_opt** (*bool, default=True*) – True if CRN are to be used for post-replications at solutions x0 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 macoreplication and print to csv.

Parameters

- **solve_tols** (*list [float], default = [0.05, 0.10, 0.20, 0.50]*) – Relative optimality gap(s) definining 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 (x0) and optimal solution (x*).

Type
int

crn_across_init_opt

True if CRN used for post-replications at solutions x0 and x*, otherwise False.

Type
bool

x0

Initial solution (x0).

Type
tuple

x0_postreps

Post-replicates at x0.

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 post-replications at solutions recommended at different times, otherwise False.
- **crn_across_macroreps** (*bool*, *default=False*) – True if CRN used for post-replications 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 x0 and optimal x*.
- **crn_across_init_opt** (*bool, default=True*) – True if CRN used for post-replications at solutions x0 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 post-replications at solutions recommended at different times, otherwise False.
- **crn_across_macroreps** (*bool, default=False*) – True if CRN used for post-replications 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 definining 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** (`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 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, x0, 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 definining 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 definining 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 definining 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 x0 and optimal x*.
- **crn_across_init_opt** (*bool, default=True*) – True if CRN used for postreplications at solutions x0 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 definining 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

CHAPTER

THREE

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

INDEX

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y