SPPA Reference (v1.0.8)

---

class sppa.SPPA.**SPPA**(*name*=None)

SPPA MINLP solver for solving MINLP problems with Sequential Piecewise Planar Approximation (SPPA)

name name of the problem.

Methods

set_objective(*expr*, *name*=None)

Set the objective of the optimization problem.

expr Expr instance. Either a Var instance, NlinExpr instance, or a linear combination of both with or without numerical constants.

name name of the objective function.

add_equality_constraint(*expr*, *name*=None, *con_tol*=1E-4)

Add an equality constraint to the optimization problem where expr=0 is the desired constraint.

expr Expr instance. Either a Var instance, NlinExpr instance, or a linear combination of both with or without numerical constants.

name name of the equality constraint.

con_tol tolerance of the constraint. The degree of tolerance that is used to decide if a constraint is violated.

add_inequality_constraint(*expr*, *name*=None, *con_tol*=1E-4)

Add an inequality constraint to the optimization problem where expr≥0 is the desired constraint.

expr Expr instance. Either a Var instance, NlinExpr instance, or a linear combination of both with or without numerical constants.

name name of the inequality constraint.

con_tol tolerance of the constraint. The degree of tolerance that is used to decide if a constraint is violated.

compile(*solver*='cbc', *n_pieces*=3, *initial_n_pieces*=None, *initial_ep_gap*=0.0, *variable_order*=None, *contract_frac*=0.7)

Compiles the optimization problem. Must be called before solving. If compiled and the problem changes (i.e. objective and constraints change), then compile() must be called again before solving.

solver	string of solver to use. 'cbc' is the open-source COIN Branch-and-Cut solver. Included by default with PuLP. 'cplex' is also supported if the Python API of CPLEX is installed.
n_pieces	number of piecewise segments in each optimization variable at every iteration. Cannot be smaller than 3.
initial_n_pieces	number of piecewise segments for the first iteration. Can be None which defaults to *n_pieces*. Cannot be smaller than 3.
initial_ep_gap	ep_gap of the MILP solver at iteration 1. If changed to a non-zero value, optimization problem may not converge in later iterations.
variable_order	order in which variables are printed in output logs and files. None for the order they were added, 'alpha-num' for alpha-numerical order.
contract_frac	the contraction factor for each variable at every SPPA iteration. Must be between 0.1 and 0.99.

set_termination_criteria(ftol=1E-6, *xtol*=1E-6, *computation_time*=None, min_iterations=5, *max_iterations*=100, *infeasible_allowed*=False, *con_tol*=None, *tol_wait*=5)

Define termination criteria for the optimization problem. A termination criteria is inactive if set to None. At least one termination criteria must not be None. Termination criterias are: ftol, xtol, computation_time, and max_iterations. To avoid numerical issues when dealing with higher precision exceeding machine epsilon, ftol and/or xtol should be set.

ftol	the objective function tolerance.
xtol	the tolerance of the solution. If all optimization variables satisfy the tolerance between iterations, the solver will terminate.
computation_time	the computation time allocated to the SPPA solver. Will only terminate between SPPA iterations.
min_iterations	the minimum number of iterations of the SPPA solver. Can be None.
max_iterations	the maximum number of iterations of the SPPA solver. Can be None.
infeasible_allowed	True or False indicating whether the SPPA solver is allowed to terminate on an infeasible solution. If True, SPPA will terminate once any other termination criteria is met. If False, SPPA will only terminate if the solution is feasible and if at least one other termination criteria is met.

con_tol if not None, gives the option of setting a constraint tolerance for all constraints at once.

tol_wait the minimum number of iterations to wait for *xtol* or *ftol* to be satisfied before terminating

print(*return_str*=False)

Print the compiled optimization problem in human readable form to the screen.

return_str if True, will not print to screen but return the output as a string.

write(*filename*=None)

Write the compiled optimization problem in human readable form to a file.

filename: write to the filename provided. A file extension of '.sspa' will be automatically appended. If filename=None, the problem name will be used for the filename.

solve(*verbose*=1, *milp_msg*=False)

Solve the optimization problem.

verbose if a print out of the SPPA solver is desired during the optimization, use verbose=1, else use verbose=None or verbose=0.

milp_msg if a print out of the MILP solver is desired durint the optimization, use milp_msg=True, else use milp_msg=False.

[returns] *instance of object type SolverResult.*

---

class sppa.SPPA.**SolverResult**(…)

Result returned by the SPPA solver.

Attributes

value the optimal objective value found by the solver.

solution the solution corresponding to *value*.

iterations the number of iterations taken by the solver.

exit_flag the exit_flag of the solver.

message a string describing the circumstances of termination.

Methods

print_message()

Print message to screen.

---

class sppa.Operations.**Var**(*var_name*, *low_bound*=None, *up_bound*=None, *var_type*='cont', *expand*=False, *breakpoint_fun*=None, ***breakpoint_kwargs*)

Optimization variable of the optimization problem. Optimization variables are identified by their variable names and not by the address of the object. Different objects with the same *var_name* but different attributes will raise an exception.

var_name	variable name of the optimization variable. A constant may also be initialized by passing the constant to var_name without assigning the other arguments.
low_bound	the lower bound of the optimization variable i.e. the minimum value the variable is able to assume. Must not be None if the variable is involved in a nonlinear expression.
up_bound	the upper bound of the optimization variable i.e. the maximum value the variable is able to assume. Must not be None if the variable is involved in a nonlinear expression.
var_type	the variable type of the optimization variable. Can be 'cont' for continuous, 'int' for integer, or 'bin' for binary.
expand	this can be True only for integer variables. If True and if the integer exists in a nonlinear expression, the integer variable is evaluated for all values in [low_bound, up_bound] for every nonlinear expression it is involved in instead of only *n_pieces*+1 times at every iteration.
breakpoint_fun	a function to determine where the breakpoints are when decomposing. Must accept 3 positional arguments *a*, *b*, and *n_vertices*, where *a* is the first vertex, *b* is the last vertex, and *n_vertices* is the total number of vertices. Additional arguments are passed with keyword arguments ***breakpoint_kwargs*.
breakpoint_kwargs	Additional keyword arguments for the *breakpoint_fun*; see above.

class sppa.Operations.**NlinExpr**(*fun_nlin*, **vars_fun*, ***kwargs*)

Nonlinear expression of the optimization problem. Nonlinear expressions are identified by fun_nlin, *vars_fun, and **kwargs, not by the object address.

fun_nlin	the nonlinear function defining the nonlinear expression.
***vars_fun**	non-keyworded arguments which are the ordered optimization variables of *fun_nlin*.
****kwargs**	keyworded arguments of *fun_nlin*. These are constants and are called together with *fun_nlin* whenever the nonlinear expression needs to be evaluated.

Inherited Methods (of *Var* and *NlinExpr* from *Expr*)

print_expr(*return_str*=False)

Print a human-readable string of the expression to screen.

return_str	if True, will not print to screen but instead return a string.
[returns]	a string of the print-out if *return_str* is True.

evaluate_expression(*variables_values*)

Evaluate the expression from a dictionary of values.

variables_values	a dictionary with keys and values corresponding to variable names and their respective numerical values.
[returns]	value of expression evaluated with variable_values