pyOpt Reference

Release 1.2.0

Ruben E. Perez, Peter W. Jansen

July 31, 2014

CONTENTS

1 Introduction **1**
- 1.1 Features . . . 1
- 1.2 Prerequisites . . . 1
- 1.3 Usage . . . 1

2 Code Structure **3**
- 2.1 Optimization . . . 3
- 2.2 Optimizer . . . 7
- 2.3 Objective . . . 8
- 2.4 Variable . . . 9
- 2.5 Constraint . . . 9
- 2.6 Gradient . . . 9
- 2.7 History . . . 10

3 Optimizers **13**
- 3.1 SNOPT - Sparse NOlinear OPTimizer . . . 13
- 3.2 NLPQL - Non-Linear Programming by Quadratic Lagrangian . . . 15
- 3.3 NLPQLP - NonLinear Programming with Non-Monotone and Distributed Line Search . . . 16
- 3.4 FSQP - Feasible Sequential Quadratic Programming . . . 17
- 3.5 SLSQP - Sequential Least Squares Programming . . . 18
- 3.6 PSQP - Preconditioned Sequential Quadratic Programming . . . 19
- 3.7 ALGENCAN - Augmented Lagrangian with GENCAN . . . 20
- 3.8 FILTERSD . . . 21
- 3.9 MMA - Method of Moving Asymptotes . . . 22
- 3.10 GCMMA - Globally Convergent Method of Moving Asymptotes . . . 23
- 3.11 CONMIN - CONstrained function MINimization . . . 24
- 3.12 MMFD - Modified Method of Feasible Directions . . . 25
- 3.13 KSOPT - Kreisselmeier–Steinhauser Optimizer . . . 26
- 3.14 COBYLA - Constrained Optimization BY Linear Approximation . . . 27
- 3.15 SDPEN - Sequential Penalty Derivative-free method for Nonlinear constrained optimization . . . 28
- 3.16 SOLVOPT - SOLver for local OPTimization problems . . . 29
- 3.17 ALPSO - Augmented Lagrangian Particle Swarm Optimizer . . . 30
- 3.18 NSGA2 - Non Sorting Genetic Algorithm II . . . 31
- 3.19 ALHSO - Augmented Lagrangian Harmony Search Optimizer . . . 32
- 3.20 MIDACO - Mixed Integer Distributed Ant Colony Optimization . . . 33

4 License **35**

5 Citing **37**

6 Acknowledgments 39

7 Glossary 41

Python Module Index 43

CHAPTER

ONE

INTRODUCTION

This is the documentation for pyOpt, an object-oriented framework for formulating and solving optimization problems in an efficient, reusable and portable manner. The goal is to provide an easy-to-use optimization framework with access to a variety of integrated optimization algorithms which are accessible through a common interface.

The focus is on formulating and solving nonlinear constrained optimization problems, be they large or small. But the formulation of unconstrained, integer or mixed-integer and discrete optimization problems is also supported. The framework also allows for easy integration of additional optimization software that is programmed in Fortran, C, C++, and other languages.

1.1 Features

Some of the main features of the pyOpt framework are:

- Problem-Optimizer Independence: Object-oriented constructs allow for true separation between the optimization problem formulation and its solution by different optimizers.
- Flexible Optimizer Integration : The interface allows for easy integration of gradient-based, gradient-free, and population-based optimization algorithms.
- Parallelization Capability: The framework can solve optimization problems where the function evaluations from the model applications run in parallel environments. For gradient-based optimizers, it can also automate the evaluation of gradients in parallel, and for gradient-free optimizers it can distribute function evaluations.
- History and Warm-Restart Capability: The user has the option to store the solver evaluation history during the optimization process. A partial history can also be used to ‘’warm-restart” the optimization.

1.2 Prerequisites

pyOpt requires at least **Python 2.4** to run, as well as the numpy ,1.0 or higher, package. Most optimizers supported by the framework require a **C** or **Fortran** compiler compatible with f2py . To link C based optimizers swig 1.3 or higher is also required. To take advantage of the parallel computing capabilities build into pyOpt the python package mpi4py is needed.

1.3 Usage

See the `Quickguide` for an introduction on how to use the pyOpt framework. Examples showing how to use some of the special features can be found in `Examples` .

CHAPTER

TWO

CODE STRUCTURE

pyOpt has been developed to facilitate reuse and extensibility with the premise that the definition of an optimization problem should be independent of the optimizer. Details for the different classes used in the package can be found below, their relationship is also illustrated in the unified modeling language class diagram below.

2.1 Optimization

class pyOpt_optimization.**Optimization**(*name*, *obj_fun*, *var_set=None*, *obj_set=None*, *con_set=None*, *use_groups=False*, **args*, ***kwargs*)

Optimization Problem Class

Optimization Problem Class Initialization

Arguments:

- name -> STR: Solution name
- opt_func -> FUNC: Objective function

Keyword arguments:

- var_set -> INST: Variable set, *Default* = None
- obj_set -> INST: Objective set, *Default* = None
- con_set -> INST: Constraints set, *Default* = None
- use_groups -> BOOL: Use of group identifiers flag, *Default* = False

Documentation last updated: May. 23, 2011 - Ruben E. Perez

ListAttributes()

Print Structured Attributes List

Documentation last updated: March. 24, 2008 - Ruben E. Perez

addCon(**args*, ***kwargs*)

Add Constraint into Constraints Set

Documentation last updated: March. 27, 2008 - Ruben E. Perez

addConGroup(*name*, *ncons*, *type='i'*, ***kwargs*)

Add a Group of Constraints into Constraints Set

Arguments:

- name -> STR: Constraint group name

•ncons -> INT: Number of constraints in group

Keyword arguments:

•type -> STR: Constraint type ('i'-inequality, 'e'-equality), *Default* = 'i'

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

addObj(**args*, ***kwargs*)
Add Objective into Objectives Set

Documentation last updated: March. 27, 2008 - Ruben E. Perez

addSol(**args*, ***kwargs*)
Add Solution into Solution Set

Documentation last updated: May. 07, 2008 - Ruben E. Perez

addVar(**args*, ***kwargs*)
Add Variable into Variables Set

Documentation last updated: March. 27, 2008 - Ruben E. Perez

addVarGroup(*name*, *nvars*, *type='c'*, *value=0.0*, ***kwargs*)
Add a Group of Variables into Variables Set

Arguments:

•name -> STR: Variable Group Name

•nvars -> INT: Number of variables in group

Keyword arguments:

•type -> STR: Variable type ('c'-continuous, 'i'-integer, 'd'-discrete), *Default* = 'c'

•value ->INT/FLOAT: Variable starting value, *Default* = 0.0

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

delCon(*i*)
Delete Constraint *i* from Constraints Set

Arguments:

•i -> INT: Constraint index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

delObj(*i*)
Delete Objective *i* from Objectives Set

Arguments:

•i -> INT: Objective index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

delSol(*i*)
Delete *i* Solution from Solutions Set

Arguments:

•i -> INT: Solution index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

delVar(*i*)
Delete Variable *i* from Variables Set

Arguments:

•i -> INT: Variable index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

delVarGroup(*name*)
Delete Variable Group *name* from Variables Set

Arguments:

•name -> STR: Variable group name

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

firstavailableindex(*set*)
List First Unused Index from Variable Objects List

Arguments:

•set -> LIST: Set to find frist available index of

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

getCon(*i*)
Get Constraint *i* from Constraint Set

Arguments:

•i -> INT: Constraint index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

getConSet()
Get Constraints Set

Documentation last updated: March. 27, 2008 - Ruben E. Perez

getObj(*i*)
Get Objective *i* from Objectives Set

Arguments:

•i -> INT: Objective index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

getObjSet()
Get Objectives Set

Documentation last updated: March. 27, 2008 - Ruben E. Perez

getSol(*i*)
Get Solution *i* from Solution Set

Arguments:

•i -> INT: Solution index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

getSolSet()
Get Solutions Set

Documentation last updated: May. 07, 2008 - Ruben E. Perez

getVar(*i*)
Get Variable *i* from Variables Set

Arguments:

- i -> INT: Variable index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

getVarGroups()
Get Variables Groups Set

Documentation last updated: June. 25, 2009 - Ruben E. Perez

getVarSet()
Get Variables Set

Documentation last updated: March. 27, 2008 - Ruben E. Perez

setCon(*i*, **args*, ***kwargs*)
Set Constraint *i* into Constraints Set

Arguments:

- i -> INT: Constraint index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

setObj(*i*, **args*, ***kwargs*)
Set Objective *i* into Objectives Set

Arguments:

- i -> INT: Objective index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

setSol(*i*, **args*, ***kwargs*)
Set Solution *i* into Solution Set

Arguments:

- i -> INT: Solution index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

setVar(*i*, **args*, ***kwargs*)
Set Variable *i* into Variables Set

Arguments:

- i -> INT: Variable index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

solution(*i*)
Get Solution from Solution Set

Arguments:

- i -> INT: Solution index

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

write2file(*outfile=''*, *disp_sols=False*, ***kwargs*)
Write Structured Optimization Problem to file

Keyword arguments:

•outfile -> STR/INST: File name or file instance, *Default* = ‘’

•disp_sols -> BOOL: Display solutions flag, *Default* = False.

•solutions -> LIST: List of solution indexes.

Documentation last updated: May. 9, 2008 - Peter W. Jansen

class pyOpt_optimization.**Solution**(*optimizer*, *name*, *obj_fun*, *opt_time*, *opt_evals*, *opt_inform*, *var_set=None*, *obj_set=None*, *con_set=None*, *options_set=None*, *myrank=0*, **args*, ***kwargs*)

Bases: pyOpt_optimization.Optimization

Optimization Solution Class

Solution Class Initialization

Arguments:

•optimizer -> STR: Optimizer name

•name -> STR: Optimization problem name

•opt_time -> FLOAT: Solution total time

•opt_evals -> INT: Number of function evaluations

Keyword arguments:

•var_set -> INST: Variable set, *Default* = {}

•obj_set -> INST: Objective set, *Default* = {}

•con_set -> INST: Constraints set, *Default* = {}

•options_set -> Options used for solution, *Default* = {}

•myrank -> INT: Process identification for MPI evaluations, *Default* = 0

Documentation last updated: Feb. 03, 2011 - Peter W. Jansen

write2file(*outfile*)

Write Structured Solution to file

Arguments:

•outfile -> STR: Output file name

Documentation last updated: May. 9, 2008 - Peter W. Jansen

2.2 Optimizer

class pyOpt_optimizer.**Optimizer**(*name={}*, *category={}*, *def_options={}*, *informs={}*, **args*, ***kwargs*)

Abstract Class for Optimizer Object

Optimizer Class Initialization

Keyword arguments:

•name -> STR: Optimizer name, *Default* = {}

•category -> STR: Optimizer category, *Default* = {}

•def_options -> DICT: Deafult options, *Default* = {}

•informs -> DICT: Calling routine informations texts, *Default* = {}

Documentation last updated: Feb. 03, 2011 - Peter W. Jansen

ListAttributes()
Print Structured Attributes List

Documentation last updated: March. 24, 2008 - Ruben E. Perez

flushFiles()
Flush Output Files (Calling Routine)

Documentation last updated: August. 09, 2009 - Ruben E. Perez

getInform(*infocode=None*)
Get Optimizer Result Information (Calling Routine)

Keyword arguments:

•infocode -> INT: information code key

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

getOption(*name*)
Get Optimizer Option Value (Calling Routine)

Arguments:

•name -> STR: Option name

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

setOption(*name*, *value=None*)
Set Optimizer Option Value (Calling Routine)

Arguments:

•name -> STR: Option Name

Keyword arguments:

•value -> FLOAT/INT/BOOL: Option Value, *Default* = None

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

2.3 Objective

class pyOpt_objective.**Objective**(*name*, *value=0.0*, *optimum=0.0*)
Optimization Objective Class

Objective Class Initialization

Arguments:

•name -> STR: Objective Group Name

Keyword arguments:

•value-> FLOAT: Initial objective value, *Default* = 0.0

•optimum-> FLOAT: Optimum objective value, *Default* = 0.0

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

ListAttributes()
Print Structured Attributes List

Documentation last updated: March. 10, 2008 - Ruben E. Perez

2.4 Variable

class pyOpt_variable.**Variable**(*name*, *type='c'*, *value=0.0*, **args*, ***kwargs*)

Optimization Variable Class

Variable Class Initialization

Arguments:

- name -> STR: Variable Name

Keyword arguments:

- type -> STR: Variable Type ('c'-continuous, 'i'-integer, 'd'-discrete), *Default* = 'c'
- value -> INT/FLOAT: Variable Value, *Default* = 0.0
- lower -> INT/FLOAT: Variable Lower Value
- upper -> INT/FLOAT: Variable Upper Value
- choices -> LIST: Variable Choices

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

2.5 Constraint

class pyOpt_constraint.**Constraint**(*name*, *type='i'*, **args*, ***kwargs*)

Optimization Constraint Class

Constraint Class Initialization

Arguments:

- name -> STR: Variable Name

Keyword arguments:

- type -> STR: Variable Type ('i'-inequality, 'e'-equality), *Default* = 'i'
- lower -> INT: Variable Lower Value
- upper -> INT: Variable Upper Value
- choices -> DICT: Variable Choices

Documentation last updated: Feb. 03, 2011 - Peter W. Jansen

ListAttributes()

Print Structured Attributes List

Documentation last updated: March. 10, 2008 - Ruben E. Perez

2.6 Gradient

class pyOpt_gradient.**Gradient**(*opt_problem*, *sens_type*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Abstract Class for Optimizer Gradient Calculation Object

Optimizer Gradient Calculation Class Initialization

Arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Sensitivity type ('FD', 'CS', or function)

Keyword arguments:

•sens_mode -> STR: Parallel flag [''-serial,'pgc'-parallel], *Default* = ''

•sens_step -> INT: Step size, *Default* = {} [=1e-6(FD), 1e-20(CS)]

Documentation last updated: Feb. 03, 2011 - Peter W. Jansen

getGrad(*x*, *group_ids*, *f*, *g*, **args*, ***kwargs*)

Get Gradient

Arguments:

•x -> ARRAY: Design variables

•group_ids -> DICT: Group identifications

•f -> ARRAY: Objective values

•g -> ARRAY: Constraint values

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

getHess(**args*, ***kwargs*)

Get Hessian

Documentation last updated: June. 20, 2010 - Ruben E. Perez

2.7 History

class pyOpt_history.**History**(*filename*, *mode*, *optimizer=None*, *opt_prob=None*, **args*, ***kwargs*)

Abstract Class for Optimizer History Object

Optimizer History Class Initialization

Arguments:

•filename -> STR: Name for .bin and .cue file

•mode -> STR: Either read ('r') or write ('w') mode

Keyword arguments:

•optimizer -> INST: Opimizer class instance, *Default* = None

•opt_prob -> STR: Optimization Problem Name, *Default* = None

Documentation last updated: April. 14, 2010 - Peter W. Jansen

close()

Close Optimizer History Files

Documentation last updated: December. 11, 2009 - Ruben E. Perez

overwrite(*bin_data*, *index*)

Overwrite Data on Optimizer History Files

Arguments:

•bin_data -> ARRAY: Data to overwrite old data

•index -> INT: Starting index of old data

Documentation last updated: Feb. 03, 2011 - Peter W. Jansen

read(*index=[], ident=['obj']*)
Read Data from Optimizer History Files

Keyword arguments:

•index -> LIST,SCALAR: Index (list), [0,-1] for all, [] internal count, -1 for last, *Default* = []

•ident -> STR: Indentifier, *Default* = 'obj'

Documentation last updated: April. 14, 2010 - Peter W. Jansen

write(*bin_data*, *cue_data*)
Write Data to Optimizer History Files

Arguments:

•bin_data -> LIST/ARRAY: Data to be written to binary file

•cue_data -> STR: Variable identifier for cue file

Documentation last updated: Feb. 07, 2011 - Peter W. Jansen

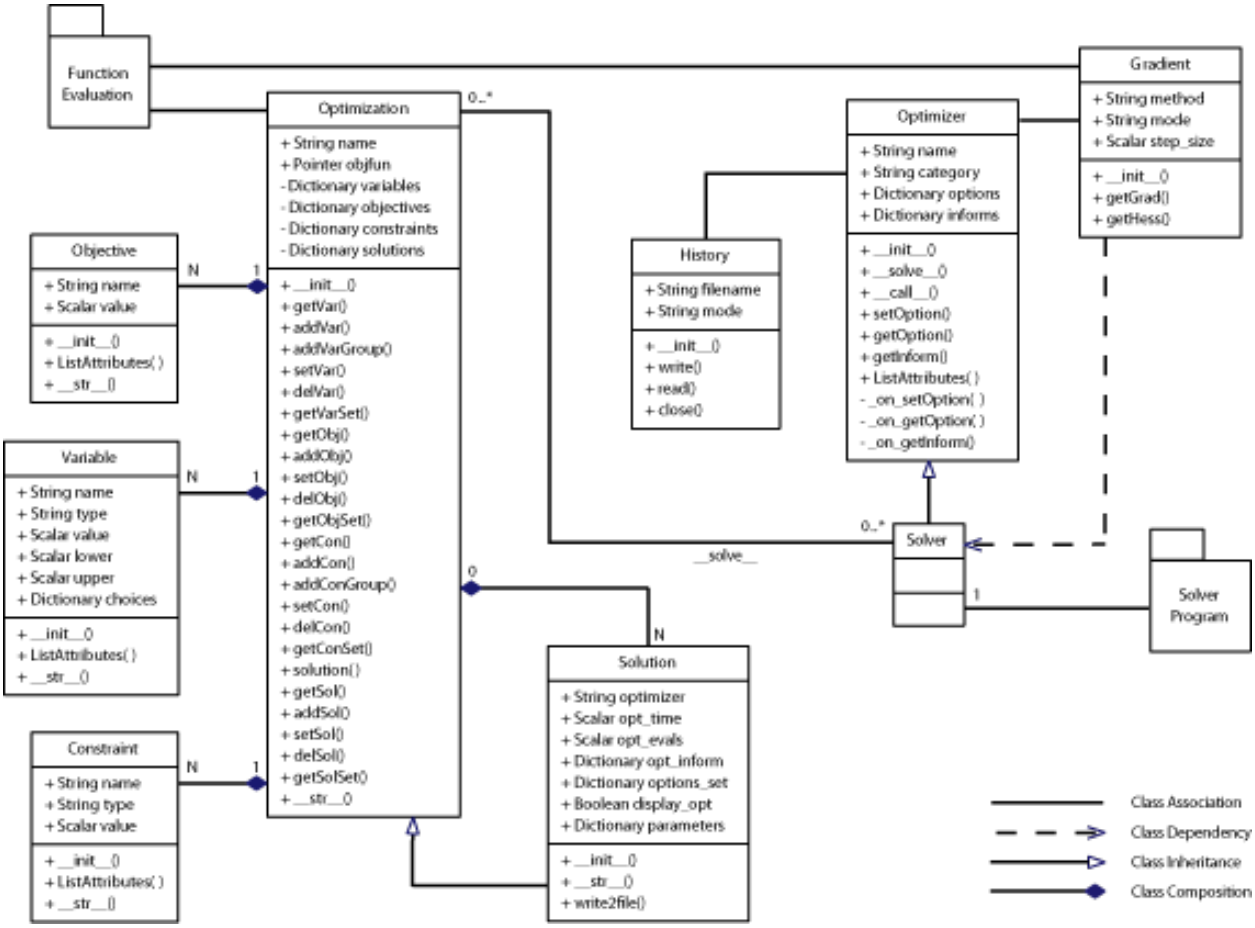

Figure 2.1: pyOpt class relationship diagram

CHAPTER

THREE

OPTIMIZERS

Different type of open-source and licensed optimizers that solve the general nonlinear optimization problem have been integrated into the package. Details for the optimizer can be found below.

3.1 SNOPT - Sparse NOlinear OPTimizer

SNOPT is a sparse nonlinear optimizer that is particularly useful for solving large-scale constrained problems with smooth objective functions and constraints. The algorithm consists of a sequential quadratic programming (SQP) algorithm that uses a smooth augmented Lagrangian merit function, while making explicit provision for infeasibility in the original problem and in the quadratic programming subproblems. The Hessian of the Lagrangian is approximated using a limited-memory quasi-Newton method. [Gill2002] [LICENSE]

class `pySNOPT.`**`SNOPT`**(*pll_type=None*, **args*, ***kwargs*)

Bases: `pyOpt.pyOpt_optimizer.Optimizer`

SNOPT Optimizer Class - Inherited from Optimizer Abstract Class

SNOPT Optimizer Class Initialization

Keyword arguments:

- pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

`__solve__`(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *disp_opts=False*, *store_hst=False*, *hot_start=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

- opt_problem -> INST: Optimization instance
- sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'
- store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True
- disp_opts -> BOOL: Flag to display options in solution text, *Default* = False
- store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False
- hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False
- sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''
- sens_step -> FLOAT: Sensitivity setp size, *Default* = { } [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: Feb. 2, 2011 - Peter W. Jansen

3.1.1 Optimizer Options

Name	Type	Default Value	Notes
Major print level	int	1	Majors Print (1 - line major iteration log)
Minor print level	int	1	Minors Print (1 - line minor iteration log)
Print file	str	'SNOPT_print.out'	Print File Name (specified by subroutine snInit)
iPrint	int	18	Print File Output Unit
Summary file	str	'SNOPT_summary.out'	Summary File Name (specified by subroutine snInit)
iSumm	int	19	Summary File Output Unit
Print frequency	int	100	Minors Log Frequency on Print File
Summary frequency	int	100	Minors Log Frequency on Summary File
Solution	str	'Yes'	Print Solution on the Print File
Suppress options listing	type(None)	None	options are normally listed
System information	str	'No'	Print System Information on the Print File
Problem Type	str	'Minimize'	Alternatives 'Maximize', 'Feasible point'
Objective row	int	1	has precedence over ObjRow (snOptA)
Infinite bound	float	1.0e+20	Infinite Bound Value
Major feasibility tolerance	float	1.0e-6	Target Nonlinear Constraint Violation
Major optimality tolerance	float	1.0e-6	Target Complementarity Gap
Minor feasibility tolerance	float	1.0e-6	For Satisfying the QP Bounds
Verify level	int	0	Gradients Check Flag
Scale option	int	1	Scaling (1 - linear constraints and variables)
Scale tolerance	float	0.9	Scaling Tolerance
Scale Print	type(None)	None	Default: scales are not printed
Crash tolerance	float	0.1	
Linesearch tolerance	float	0.9	smaller for more accurate search
Pivot tolerance	float	3.7e-11	epsilon^{2/3}
QPSolver	str	'Cholesky'	Default: Cholesky
Crash option	int	3	3 - first basis is essentially triangular
Elastic mode	str	'No'	start with elastic mode until necessary
Elastic weight	float	1.0e+4	used only during elastic mode
Iterations limit	int	10000	or 20*ncons if that is more
Partial price	int	1	10 for large LPs
Start	str	'Cold'	has precedence over argument start
Major iterations limit	int	1000	or ncons if that is more
Minor iterations limit	int	500	or 3*ncons if that is more
Major step limit	float	2.0	
Superbasics limit	int	None	n1 + 1, n1 = number of nonlinear variables
Derivative level	int	3	NOT ALLOWED IN snOptA
Derivative option	int	1	ONLY FOR snOptA
Derivative linesearch	type(None)	None	
Nonderivative linesearch	type(None)	None	
Function precision	float	3.0e-13	epsilon^0.8 (almost full accuracy)
Difference interval	float	5.5e-7	Function precision^(1/2)
Central difference interval	float	6.7e-5	Function precision^(1/3)
New superbasics limit	int	99	controls early termination of QPs

Continued on next page

Table 3.1 – continued from previous page

Name	Type	Default Value	Notes
Objective row	int	1	row number of objective in F(x)
Penalty parameter	float	0.0	initial penalty parameter
Proximal point method	int	1	1 - satisfies linear constraints near x0
Reduced Hessian dimension	int	2000	or Superbasics limit if that is less
Violation limit	int	10.0	unscaled constraint violation limit
Unbounded step size	float	1.0e+18	
Unbounded objective	float	1.0e+15	
Hessian full memory	type(None)	None	default if n1 <= 75
Hessian limited memory	type(None)	None	default if n1 > 75
Hessian frequency	int	999999	for full Hessian (never reset)
Hessian updates	int	10	for limited memory Hessian
Hessian flush	int	999999	no flushing
Check frequency	int	60	test row residuals l2norm(Ax - sk)
Expand frequency	int	10000	for anti-cycling procedure
Factorization frequency	int	50	100 for LPs
Save frequency	int	100	save basis map
LU factor tolerance	float	3.99	for NP (100.0 for LP)
LU update tolerance	float	3.99	for NP (10.0 for LP)
LU singularity tolerance	float	3.2e-11	
LU partial pivoting	type(None)	None	default threshold pivoting strategy
LU rook pivoting	type(None)	None	threshold rook pivoting
LU complete pivoting	type(None)	None	threshold complete pivoting
Old basis file	int	0	input basis map
New basis file	int	0	output basis map
Backup basis file	int	0	output extra basis map
Insert file	int	0	input in industry format
Punch file	int	0	output Insert data
Load file	int	0	input names and values
Dump file	int	0	output Load data
Solution file	int	0	different from printed solution
Total character workspace	int	500	lencw: 500
Total integer workspace	int	None	leniw: 500 + 100 * (m+n)
Total real workspace	int	None	lenrw: 500 + 200 * (m+n)
User character workspace	int	500	
User integer workspace	int	500	
User real workspace	int	500	
Debug level	int	0	0 - Normal, 1 - for developers
Timing level	int	3	3 - print cpu times

3.2 NLPQL - Non-Linear Programming by Quadratic Lagrangian

NLPQL is a sequential quadratic programming (SQP) method which solves problems with smooth continuously differentiable objective function and constraints. The algorithm uses a quadratic approximation of the Lagrangian function and a linearization of the constraints. To generate a search direction a quadratic subproblem is formulated and solved. The line search can be performed with respect to two alternative merit functions, and the Hessian approximation is updated by a modified BFGS formula. [Schitt1986] [LICENSE]

class `pyNLPQL.`**`NLPQL`**(*pll_type=None*, **args*, ***kwargs*)

Bases: `pyOpt.pyOpt_optimizer.Optimizer`

NLPQL Optimizer Class - Inherited from Optimizer Abstract Class

NLPQL Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *disp_opts=False*, *store_hst=False*, *hot_start=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = { } [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2011 - Peter W. Jansen

3.2.1 Optimizer Options

Name	Type	Default Value	Notes
Accurancy	float	1e-6	Convergence Accurancy
ScaleBound	float	1e30	
maxFun	int	20	Maximum Number of Function Calls During Line Search
maxIt	int	500	Maximum Number of Iterations
iPrint	int	2	Output Level (0-None, 1-Final, 2-Major, 3-Major/Minor, 4-Full)
mode	int	0	NLPQL Mode (0 - Normal Execution, 1 to 18 - See Manual)
iout	int	6	Output Unit Number|
lmerit	bool	True	Merit Function (True: L2 Augmented Penalty, False: L1 Penalty)
lql	bool	False	QP Solver (True - Quasi-Newton, False - Cholesky)
iFile	str	'NLPQL.out'	Output File Name

3.3 NLPQLP - NonLinear Programming with Non-Monotone and Distributed Line Search

NLPQLP is an update of NLPQL for which a non-monotone line search is implemented allowing for increases in the merit function in case of numerical instabilities. The changes made to the original code also enable the ability to perform parallel function evaluations during line search. [Dai2008] and [Schitt2011] [LICENSE]

class pyNLPQLP.**NLPQLP**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

NLPQLP Optimizer Class - Inherited from Optimizer Abstract Class

NLPQLP Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *store_hst=False*, *hot_start=False*, *disp_opts=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = {} [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2013 - Peter W. Jansen

3.3.1 Optimizer Options

Name	Type	Default Value	Notes
ACC	float	1e-8	Convergence Accurancy
ACCQP	float	1e-12	QP Solver Convergence Accurancy
STPMIN	float	1e-10	Minimum Step Length
MAXFUN	int	10	Maximum Number of Function Calls During Line Search
MAXIT	int	100	Maximum Number of Outer Iterations
RHOB	float	0.0	BFGS-Update Matrix Initialization Parameter
IPRINT	int	2	Output Level (0-None, 1-Final, 2-Major, 3-Major/Minor, 4-Full)
MODE	int	0	NLPQL Mode (0 - Normal Execution, 1 to 18 - See Manual)
IOUT	int	6	Output Unit Number|
LQL	bool	True	QP Solver (True - Quasi-Newton, False - Cholesky)
IFILE	str	'NLPQLP.out'	Output File Name

3.4 FSQP - Feasible Sequential Quadratic Programming

This code implements an SQP approach that is modified to generate feasible iterates. In addition to handling general single objective constrained nonlinear optimization problems, the code is also capable of handling multiple competing linear and nonlinear objective functions (minimax), linear and nonlinear inequality constraints, as well as linear and nonlinear equality constraints. [Lawrence1996] [LICENSE]

class pyFSQP.**FSQP**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

FSQP Optimizer Class - Inherited from Optimizer Abstract Class

FSQP Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *store_hst=False*, *hot_start=False*, *disp_opts=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = {} [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2011 - Ruben E. Perez

3.4.1 Optimizer Options

Name	Type	Default Value	Notes
mode	int	100	FSQP Mode
iprint	int	2	Output Level (0- None, 1- Final, 2- Major, 3- Major Details)
miter	int	500	Maximum Number of Iterations
bigbnd	float	1e10	Plus Infinity Value
epstol	float	1e-8	Convergence Tolerance
epseqn	float	0	Equality Constraints Tolerance
iout	int	6	Output Unit Number
ifile	str	'FSQP.out'	Output File Name

3.5 SLSQP - Sequential Least Squares Programming

SLSQP optimizer is a sequential least squares programming algorithm which uses the Han–Powell quasi–Newton method with a BFGS update of the B–matrix and an L1–test function in the step–length algorithm. The optimizer uses a slightly modified version of Lawson and Hanson's NNLS nonlinear least-squares solver. [Kraft1988] [LICENSE]

class pySLSQP.**SLSQP**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

SLSQP Optimizer Class - Inherited from Optimizer Abstract Class

SLSQP Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *disp_opts=False*, *store_hst=False*, *hot_start=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = { } [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2011 - Peter W. Jansen

3.5.1 Optimizer Options

Name	Type	Default Value	Notes
ACC	float	1e-6	Convergence Accurancy
MAXIT	int	50	Maximum Iterations
IPRINT	int	1	Output Level (neg-None, 0-Screen, 1-File)
IOUT	int	6	Output Unit Number
IFILE	str	'SLSQP.out'	Output File Name

3.6 PSQP - Preconditioned Sequential Quadratic Programming

This optimizer implements a sequential quadratic programming method with a BFGS variable metric update. [LICENSE]

class pyPSQP.**PSQP**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

PSQP Optimizer Class - Inherited from Optimizer Abstract Class

PSQP Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *store_hst=False*, *hot_start=False*, *disp_opts=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = { } [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2011 - Ruben E. Perez

3.6.1 Optimizer Options

Name	Type	Default Value	Notes
XMAX	float	1e16	Maximum Stepsize
TOLX	float	1e-16	Variable Change Tolerance
TOLC	float	1e-6	Constraint Violation Tolerance
TOLG	float	1e-6	Lagrangian Gradient Tolerance
RPF	float	1e-4	Penalty Coefficient
MIT	int	1000	Maximum Number of Iterations
MFV	int	2000	Maximum Number of Function Evaluations
MET	int	2	Variable Metric Update (1 - BFGS, 2 - Hoshino)
MEC	int	2	Negative Curvature Correction (1-None, 2-Powell's Correction)
IPRINT	int	2	Output Level (0 - None, 1 - Final, 2 - Iter)
IOUT	int	6	Output Unit Number
IFILE	str	'PSQP.out'	Output File Name

3.7 ALGENCAN - Augmented Lagrangian with GENCAN

ALGENCAN solves the general non-linear constrained optimization problem without resorting to the use of matrix manipulations. It uses instead an Augmented Lagrangian approach which is able to solve extremely large problems with moderate computer time. [Andreani2007] [LICENSE]

class pyALGENCAN.**ALGENCAN**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

ALGENCAN Optimizer Class - Inherited from Optimizer Abstract Class

ALGENCAN Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *disp_opts=False*, *store_hst=False*, *hot_start=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = { } [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2011 - Peter W. Jansen

3.7.1 Optimizer Options

Name	Type	Default Value	Notes
epsfeas	float	1e-8	Feasibility Convergence Accurancy
epsopt	float	1e-8	Optimality Convergence Accurancy
efacc	float	1e-4	Feasibility Level for Newton-KKT Acceleration
eoacc	float	1e-4	Optimality Level for Newton-KKT Acceleration
checkder	bool	False	Check Derivatives Flag
iprint	int	2	Output Level (0 - None, 10 - Final, >10 - Iter Details)
ifile	str	'ALGENCAN.out'	Output File Name
ncomp	int	6	Print Precision

3.8 FILTERSD

FILTERSD uses a generalization of Robinson's method, globalised by using a filter and trust region. The code makes use of a Ritz values approach Linear Constraint Problem solver. Second derivatives and storage of an approximate reduced Hessian matrix is avoided using a limited memory spectral gradient approach based on Ritz values. [LICENSE]

class `pyFILTERSD.`**FILTERSD**(*pll_type=None*, **args*, ***kwargs*)

Bases: `pyOpt.pyOpt_optimizer.Optimizer`

FILTERSD Optimizer Class - Inherited from Optimizer Abstract Class

FILTERSD Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *store_hst=False*, *hot_start=False*, *disp_opts=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = {} [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2013 - Ruben E. Perez

3.8.1 Optimizer Options

Name	Type	Default Value	Notes
rho	float	1.0	Initial Trust Region Radius
htol	float	1.0e-6	Constraint Feasibilities Tolerance
rgtol	float	1.0e-5	Reduced Gradient l2 norm Tolerance
maxit	int	1000	Maximum Number of Iterations
maxgr	int	1.0e+5	Gradient Calls Upper Limit
ubd	float	1.0e+5	Constraint Violation Upper Bound
dchk	int	0	Derivative Check Flag (0 - no check, 1 - check)
dtol	float	1.0e-8	Derivative Check Tolerance
iprint	int	1	Print Flag (0-None, 1-Iter, 2-Debug)
iout	int	6	Output Unit Number
ifile	str	'FILTERSD.out'	Output File Name

3.9 MMA - Method of Moving Asymptotes

This is an implementation of the method of moving asymptotes (MMA). MMA uses a special type of convex approximation. For each step of the iterative process, a strictly convex approximating subproblem is generated and solved. The generation of these subproblems is controlled by the so-called moving asymptotes, which both stabilize and speed up the convergence of the general process. [Svanberg1987] [LICENSE]

class `pyMMA.`**`MMA`**(*pll_type=None*, **args*, ***kwargs*)

Bases: `pyOpt.pyOpt_optimizer.Optimizer`

MMA Optimizer Class - Inherited from Optimizer Abstract Class

MMA Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *disp_opts=False*, *store_hst=False*, *hot_start=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = { } [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2011 - Peter W. Jansen

3.9.1 Optimizer Options

Name	Type	Default Value	Notes
MAXIT	int	1000	Maximum Iterations
GEPS	float	1e-6	Dual Objective Gradient Tolerance
DABOBJ	float	1e-6	Min absolute change in the objective to indicate convergence
DELOBJ	float	1e-6	Min relative change in the objective to indicate convergence
ITRM	int	2	Number of consecutive iterations to indicate convergence
IPRINT	int	1	Output Level (neg-None, 0-Screen, 1-File)
IOUT	int	6	Output Unit Number
IFILE	str	'MMA.out'	Output File Name

3.10 GCMMA - Globally Convergent Method of Moving Asymptotes

GCMMA is a variant of the original method of moving asymptotes *MMA - Method of Moving Asymptotes* algorithm. The variant extends the original MMA functionality and guarantees convergence to some local minimum from any feasible starting point. [Svanberg1995] [LICENSE]

class pyGCMMA.**GCMMA**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

GCMMA Optimizer Class - Inherited from Optimizer Abstract Class

GCMMA Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *disp_opts=False*, *store_hst=False*, *hot_start=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = { } [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call,

Documentation last updated: February. 2, 2011 - Peter W. Jansen

3.10.1 Optimizer Options

Name	Type	Default Value	Notes
MAXIT	int	1000	Maximum Iterations
INNMAX	int	10	Maximum Inner Iterations
GEPS	float	1e-6	Dual Objective Gradient Tolerance
DABOBJ	float	1e-6	Min absolute change in the objective to indicate convergence
DELOBJ	float	1e-6	Min relative change in the objective to indicate convergence
ITRM	int	2	Number of consecutive iterations to indicate convergence
IPRINT	int	1	Output Level (neg-None, 0-Screen, 1-File)
IOUT	int	6	Output Unit Number
IFILE	str	'GCMMA.out'	Output File Name

3.11 CONMIN - CONstrained function MINimization

This optimizer implements the method of feasible directions. CONMIN solves the nonlinear programming problem by moving from one feasible point to an improved one by choosing at each iteration a feasible direction and step size that improves the objective function. [Vanderplaats1973] [LICENSE]

class pyCONMIN.**CONMIN**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

CONMIN Optimizer Class - Inherited from Optimizer Abstract Class

CONMIN Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *store_hst=False*, *hot_start=False*, *disp_opts=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = { } [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call

Documentation last updated: February. 2, 2011 - Ruben E. Perez

3.11.1 Optimizer Options

Name	Type	Default Value	Notes
ITMAX	int	1e4	Maximum Number of Iterations
DELFUN	float	1e-6	Objective Relative Tolerance
DABFUN	float	1e-6	Objective Absolute Tolerance
ITRM	int	2	
NFEASCT	int	20	
IPRINT	int	2	Print Control (0 - None, 1 - Final, 2,3,4,5 - Debug)
IOUT	int	6	Output Unit Number
IFILE	str	'CONMIN.out'	Output File Name

3.12 MMFD - Modified Method of Feasible Directions

This optimizer is an extension of the method of feasible directions *CONMIN - CONstrained function MINimization* algorithm. MMFD utilizes the direction-finding sub-problem from the Method of Feasible Directions to find a search direction but does not require the addition of a large number of slack variables associated with inequality constraints. [Vanderplaats1983] [LICENSE]

class pyMMFD.**MMFD**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

MMFD Optimizer Class - Inherited from Optimizer Abstract Class

MMFD Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *store_hst=False*, *hot_start=False*, *disp_opts=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = {} [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2011 - Ruben E. Perez

3.12.1 Optimizer Options

Name	Type	Default Value	Notes
IOPT	int	0	Feasible Directions Approach (0 - MMFD, 1 - MFD)
IONED	int	0	One-Dimensional Search Method (0,1,2,3)
CT	float	-3e-2	Constraint Tolerance
DABOBJ	float	1e-3	Objective Absolute Tolerance (DABOBJ*abs(f(x)))
CTMIN	float	4e-3	Active Constraint Tolerance
DABOBJ	float	1e-3	Objective Absolute Tolerance (DABOBJ*abs(f(x)))
DELOBJ	float	1e-3	Objective Relative Tolerance
THETAZ	float	1e-1	Push-Off Factor
PMLT	float	1e1	Penalty multiplier for equality constraints
ITMAX	int	4e2	Maximum Number of Iterations
ITRMOP	int	3	Consecutive Iterations Iterations for Convergence
IPRINT	int	2	Print Control (0 - None, 1 - Final, 2 - Iters)
IFILE	str	'MMFD.out'	Output File Name

3.13 KSOPT - Kreisselmeier–Steinhauser Optimizer

This code reformulates the constrained problem into an unconstrained one using a composite Kreisselmeier–Steinhauser objective function to create an envelope of the objective function and set of constraints. The envelope function is then optimized using a sequential unconstrained minimization technique (SUMT). At each iteration, the unconstrained optimization problem is solved using the Davidon–Fletcher–Powell (DFP) algorithm. [Wrenn1989] [LICENSE]

class pyKSOPT.**KSOPT**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

KSOPT Optimizer Class - Inherited from Optimizer Abstract Class

KSOPT Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *store_hst=False*, *hot_start=False*, *disp_opts=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = {} [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2011 - Ruben E. Perez

3.13.1 Optimizer Options

Name	Type	Default Value	Notes
ITMAX	int	4e2	Maximum Number of Iterations
RDFUN	float	1e-4	Objective Convergence Relative Tolerance
RHOMIN	float	5	Initial KS multiplier
RHOMAX	float	100.0	Final KS multiplier
IPRINT	int	2	Print Control (0 - None, 1 - Final, 2 - Iters)
IOUT	int	6	Output Unit Number
IFILE	str	'KSOPT.out'	Output File Name

3.14 COBYLA - Constrained Optimization BY Linear Approximation

COBYLA is an implementation of Powell's nonlinear derivative–free constrained optimization that uses a linear approximation approach. The algorithm is a sequential trust–region algorithm that employs linear approximations to the objective and constraint functions, where the approximations are formed by linear interpolation at n + 1 points in the space of the variables and tries to maintain a regular–shaped simplex over iterations. [Powell1994] [LICENSE]

class `pyCOBYLA.`**`COBYLA`**(*pll_type=None*, **args*, ***kwargs*)

Bases: `pyOpt.pyOpt_optimizer.Optimizer`

COBYLA Optimizer Class - Inherited from Optimizer Abstract Class

COBYLA Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

`__solve__`(*opt_problem={}*, *store_sol=True*, *disp_opts=False*, *store_hst=False*, *hot_start=False*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2011 - Peter W. Jansen

3.14.1 Optimizer Options

Name	Type	Default Value	Notes
RHOBEG	float	0.5	Initial Variables Change
RHOEND	float	1.0e-6	Convergence Accurancy
IPRINT	int	2	Print Flag (0-None, 1-Final, 2,3-Iteration)
MAXFUN	int	3500	Maximum Number of Iterations
IOUT	int	6	Output Unit Number
IFILE	str	'COBYLA.out'	Output File Name

3.15 SDPEN - Sequential Penalty Derivative-free method for Nonlinear constrained optimization

SDPEN is a derivative-free algorithm for local general constrained optimization problems. The algorithm solves the original nonlinear constrained optimization problem by a sequence of approximate minimizations of a merit function where penalization of constraint violation is progressively increased. At each sequence, a line-search based method is used with convergence to stationary points enforced using a suitable combination of the penalty parameter updating and different sampling strategies. [Liuzzi2010] [LICENSE]

class `pySDPEN.`**`SDPEN`**(*pll_type=None*, **args*, ***kwargs*)

Bases: `pyOpt.pyOpt_optimizer.Optimizer`

SDPEN Optimizer Class - Inherited from Optimizer Abstract Class

SDPEN Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: August. 09, 2012 - Ruben E. Perez

`__solve__`(*opt_problem={}*, *store_sol=True*, *disp_opts=False*, *store_hst=False*, *hot_start=False*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: August. 09, 2012 - Ruben E. Perez

3.15.1 Optimizer Options

Name	Type	Default Value	Notes
alfa_stop	float	1.0e-6	Convergence Accurancy
nf_max	int	5000	Maximum Number of Function Evaluations
iprint	int	0	Print Flag (<0-None, 0-Final, 1,2-Iteration)
iout	int	6	Output Unit Number
ifile	str	'SDPEN.out'	Output File Name

3.16 SOLVOPT - SOLver for local OPTimization problems

SOLVOPT is a modified version of Shor's r–algorithm with space dilation to find a local minimum of nonlinear and non–smooth problems . The algorithm handles constraints using an exact penalization method. [Kuntsevich1997] [LICENSE]

class pySOLVOPT.**SOLVOPT**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

SOLVOPT Optimizer Class - Inherited from Optimizer Abstract Class

SOLVOPT Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *sens_type='FD'*, *store_sol=True*, *store_hst=False*, *hot_start=False*, *disp_opts=False*, *sens_mode=''*, *sens_step={}*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•sens_type -> STR/FUNC: Gradient type, *Default* = 'FD'

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

•sens_mode -> STR: Flag for parallel gradient calculation, *Default* = ''

•sens_step -> FLOAT: Sensitivity setp size, *Default* = {} [corresponds to 1e-6 (FD), 1e-20(CS)]

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2011 - Peter W. Jansen

3.16.1 Optimizer Options

Name	Type	Default Value	Notes
xtol	float	1.0e-4	Variables Tolerance
ftol	float	1.0e-6	Objective Tolerance
maxit	int	15000	Maximum Number of Iterations
iprint	int	1	Output Level (-1-None, 0-Final, N-each Nth iter)
gtol	float	1.0e-8	Constraints Tolerance
spcdil	float	2.5	Space Dilation
iout	int	6	Output Unit Number
ifile	str	'SOLVOPT.out'	Output File Name

3.17 ALPSO - Augmented Lagrangian Particle Swarm Optimizer

This is a parallel augmented Lagrange multiplier particle swarm optimizer developed in Python. It solves non-linear non-smooth constrained problems using an augmented Lagrange multiplier approach to handle constraints. [Jansen2011] [LICENSE]

class pyALPSO.**ALPSO**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

ALPSO Optimizer Class - Inherited from Optimizer Abstract Class

ALPSO Optimizer Class Initialization

Keyword arguments:

- pll_type -> STR: ALPSO Parallel Implementation (None, SPM- Static, DPM- Dynamic, POA-Parallel Analysis), *Default* = None

Documentation last updated: February. 2, 2011 - Ruben E. Perez

__solve__(*opt_problem={}*, *store_sol=True*, *disp_opts=False*, *xstart=[]*, *store_hst=False*, *hot_start=False*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

- opt_problem -> INST: Optimization instance
- store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True
- disp_opts -> BOOL: Flag to display options in solution text, *Default* = False
- xstart -> : , *Default* = []
- store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False
- hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 2, 2011 - Ruben E. Perez

3.17.1 Optimizer Options

Name	Type	Default Value	Notes
SwarmSize	int	40	Number of Particles (Depends on Problem dimensions)

Continued on next page

Table 3.2 – continued from previous page

Name	Type	Default Value	Notes
maxOuterIter	int	200	Maximum Number of Outer Loop Iterations (Major Iterations)
maxInnerIter	int	6	Maximum Number of Inner Loop Iterations (Minor Iterations)
minInnerIter	int	6	Minimum Number of Inner Loop Iterations (Minor Iterations)
dynInnerIter	int	0	Dynamic Number of Inner Iterations Flag
stopCriteria	int	1	Stopping Criteria Flag (0 - maxIters, 1 - convergence)
stopIters	int	5	Consecutively Number of Iterations for Convergence
etol	float	1e-3	Absolute Tolerance for Equality constraints
itol	float	1e-3	Absolute Tolerance for Inequality constraints
rtol	float	1e-2	Relative Tolerance for Lagrange Multipliers
atol	float	1e-2	Absolute Tolerance for Lagrange Function
dtol	float	1e-1	Relative Tolerance in Particles Distance to Terminate (GCPSO)
printOuterIters	int	0	Number of Iterations Before Print Outer Loop Information
printInnerIters	int	0	Number of Iterations Before Print Inner Loop Information
rinit	float	1.0	Initial Penalty Factor
xinit	int	0	Initial Position Flag (0 - no position, 1 - position given)
vinit	float	1.0	Initial Velocity of Particles Normalized in [-1,1] Space
vmax	float	2.0	Maximum Velocity of Particles Normalized in [-1,1] Space
c1	float	2.0	Cognitive Parameter
c2	float	1.0	Social Parameter
w1	float	0.99	Initial Inertia Weight
w2	float	0.55	Final Inertia Weight
ns	int	15	Consecutive Successes Before Radius will be Increased (GCPSO)
nf	int	5	Consecutive Failures Before Radius will be Increased (GCPSO)
dt	float	1	Time step
vcrazy	float	1e-4	Craziness Velocity
fileout	int	1	Flag to Turn On Output to filename
filename	str	'ALPSO.out'	Output File Name
seed	float	0	Random Number Seed (0 - Auto-Seed based on time clock)
HoodSize	int	40	Number of Neighbours of Each Particle
HoodModel	str	'gbest'	Neighbourhood Model (dl/slring, wheel, Spatial, sfrac
HoodSelf	int	1	Selfless Neighbourhood Model
Scaling	int	1	Design Variables Scaling (0- no scaling, 1- scaling [-1,1])

3.18 NSGA2 - Non Sorting Genetic Algorithm II

This optimizer is a non-dominating sorting genetic algorithm that solves non-convex and non-smooth single and multiobjective optimization problems. The algorithm attempts to perform global optimization, while enforcing constraints using a tournament selection-based strategy. [Deb2002] [LICENSE]

class pyNSGA2.**NSGA2**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

NSGA2 Optimizer Class - Inherited from Optimizer Abstract Class

NSGA2 Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *store_sol=True*, *disp_opts=False*, *store_hst=False*, *hot_start=False*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 16, 2011 - Peter W. Jansen

3.18.1 Optimizer Options

Name	Type	Default Value	Notes
PopSize	int	100	Population Size (a Multiple of 4)
maxGen	int	150	Maximum Number of Generations
pCross_real	float	0.6	Probability of Crossover of Real Variable (0.6-1.0)
pMut_real	float	0.2	Probablity of Mutation of Real Variables (1/nreal)
eta_c	float	10.0	Distribution Index for Crossover (5-20) must be > 0
eta_m	float	20.0	Distribution Index for Mutation (5-50) must be > 0
pCross_bin	float	0.0	Probability of Crossover of Binary Variable (0.6-1.0)
pMut_real	float	0.0	Probability of Mutation of Binary Variables (1/nbits)
PrintOut	int	1	Flag to Turn On Output to files (0-None, 1-Subset, 2-All)
seed	float	0.0	Random Number Seed (0 - Auto-Seed based on time clock)

3.19 ALHSO - Augmented Lagrangian Harmony Search Optimizer

This code is an extension of the harmony search optimizer [Geem2001] that handles constraints using an augmented Lagrange multiplier approach similar to that implemented in ALPSO. [LICENSE]

class pyALHSO.**ALHSO**(*pll_type=None*, **args*, ***kwargs*)

Bases: pyOpt.pyOpt_optimizer.Optimizer

ALHSO Optimizer Class - Inherited from Optimizer Abstract Class

ALHSO Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

__solve__(*opt_problem={}*, *store_sol=True*, *disp_opts=False*, *store_hst=False*, *hot_start=False*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 17, 2011 - Peter W. Jansen

3.19.1 Optimizer Options

Name	Type	Default Value	Notes
hms	int	5	Memory Size [1,50]
hmcr	float	0.95	Probability rate of choosing from memory [0.7,0.99]
par	float	0.65	Pitch adjustment rate [0.1,0.99]
dbw	int	2000	Variable Bandwidth Quantization
maxoutiter	int	2e3	Maximum Number of Outer Loop Iterations (Major Iterations)
maxinniter	int	2e2	Maximum Number of Inner Loop Iterations (Minor Iterations)
stopcriteria	int	1	Stopping Criteria Flag
stopiters	int	10	Consecutively Number of Outer Iterations for convergence
etol	float	1e-6	Absolute Tolerance for Equality constraints
itol	float	1e-6	Absolute Tolerance for Inequality constraints
atol	float	1e-6	Absolute Tolerance for Objective Function
rtol	float	1e-6	Relative Tolerance for Objective Function
prtoutiter	int	0	Number of Iterations Before Print Outer Loop Information
prtinniter	int	0	Number of Iterations Before Print Inner Loop Information
xinit	int	0	Initial Position Flag (0 - no position, 1 - position given)
rinit	float	1.0	Initial Penalty Factor
fileout	int	1	Flag to Turn On Output to filename
filename	str	'ALHSO.out'	Output File Name
seed	float	0	Random Number Seed (0 - Auto-Seed based on time clock)
scaling	int	1	Design Variables Scaling (0- no scaling, 1- scaling [-1,1])

3.20 MIDACO - Mixed Integer Distributed Ant Colony Optimization

This optimizer implements an extended ant colony optimization to solve non-convex nonlinear programming problems. The algorithm handles constraints using an oracle penalty method. [Schluter2009] [LICENSE]

class `pyMIDACO.`**`MIDACO`**(*pll_type=None*, **args*, ***kwargs*)

Bases: `pyOpt.pyOpt_optimizer.Optimizer`

MIDACO Optimizer Class - Inherited from Optimizer Abstract Class

MIDACO Optimizer Class Initialization

Keyword arguments:

•pll_type -> STR: Parallel Implementation (None, 'POA'-Parallel Objective Analysis), *Default* = None

Documentation last updated: Feb. 16, 2010 - Peter W. Jansen

`__solve__`(*opt_problem={}*, *store_sol=True*, *disp_opts=False*, *store_hst=False*, *hot_start=False*, **args*, ***kwargs*)

Run Optimizer (Optimize Routine)

Keyword arguments:

•opt_problem -> INST: Optimization instance

•store_sol -> BOOL: Store solution in Optimization class flag, *Default* = True

•disp_opts -> BOOL: Flag to display options in solution text, *Default* = False

•store_hst -> BOOL/STR: Flag/filename to store optimization history, *Default* = False

•hot_start -> BOOL/STR: Flag/filename to read optimization history, *Default* = False

Additional arguments and keyword arguments are passed to the objective function call.

Documentation last updated: February. 17, 2011 - Peter W. Jansen

3.20.1 Optimizer Options

Name	Type	Default Value	Notes
ACC	float	0	Accuracy for constraint violation (0 - Use internal default)
ISEED	int	0	Seed for random number generator (0 - Use internal default)
FSTOP	int	0	Objective Function Stopping Value (0 - disabled)
AUTOSTOP	int	0	Automatic stopping criteria (0 - disable, [1,500] - from local to global)
ORACLE	float	0	Oracle parameter for constrained problems (0 - Use internal default)
FOCUS	int	0	Focus of search process around best solution (0 - Use internal default)
ANTS	int	0	Number of iterates (ants) (0 - Use internal default)
KERNEL	int	0	Size of the solution archive (0 - Use internal default)
CHARACTER	int	0	Internal custom parameters (see MIDACO manual for options)
MAXEVAL	int	10000	Maximal function evaluations
MAXTIME	int	86400	Maximal time limit, in seconds
IPRINT	int	1	Output Level (<0 - None, 0 - Screen, 1 - File)
PRINTEVAL	int	10000	Print history after every PRINTEVAL evaluation
IOUT1	int	36	History output unit number
IOUT2	int	37	Best solution output unit number
IFILE1	str	'MIDACO_HIST.out'	History output file name
IFILE2	str	'MIDACO_BEST.out'	Best output file name
LKEY	str	'-'	License Key (Default - Limited Key)

CHAPTER

FOUR

LICENSE

CHAPTER

FIVE

CITING

When citing pyOpt in a manuscript or publication, please use the following:

Perez R.E., Jansen P.W., and Martins J.R.R.A. (2012) pyOpt: A Python-Based Object-Oriented Framework for Nonlinear Constrained Optimization, Structures and Multidisciplinary Optimization, 45(1):101-118. DOI

Note that the citation can be exported in multiple formats by following the DOI link and clicking on Export Citation. Alternatively you can use the following:

BibTeX

```
@ARTICLE{pyopt-paper,
  author = {Ruben E. Perez and Peter W. Jansen and Joaquim R. R. A. Martins},
  title = {py{O}pt: A {P}ython-Based Object-Oriented Framework for Nonlinear Constrained Optimizatio
  journal = {Structures and Multidisciplinary Optimization},
  year = {2012},
  volume = {45},
  number = {1},
  pages = {101--118},
  doi = {10.1007/s00158-011-0666-3}
}
```

We encourage to include as well appropriate citation(s) for the optimization algorithm(s) that were used within pyOpt. Details of the authors and appropriate citations for the optimization algorithms can be found in the reference section under *Optimizers*.

CHAPTER

SIX

ACKNOWLEDGMENTS

pyOpt was originally written by Ruben Perez, and Peter Jansen with the help and encouragement of many others.

The pyOpt authors graciously thank all those who have made contributions to the pyOpt development in the form of code contributions, bug reports (and fixes), documentation, and input on design, features, and desired functionality.

Special thanks to the following contributors who have made valuable contributions (roughly in order of first contribution by date)

- Dr. Joaquim Martins provided the initial wrapping interface to SNOPT 6, as well as encouraged, provided ideas, and supported the authors during the genesis of the framework development, and has tirelessly promote the use of pyOpt
- Sandy Mader helped in the implementation of the NLPQL and CONMIN wrappers, provided a method to use SNOPT with MPI enabled analyses, tested the integrated optimizers on high-fidelity optimization problems, and reported usability feedback
- Chris Marriage provided an updated wrapping interface to the SNOPT 7 version, and reported usability feedback
- Andrew Lambe provided a basic wrapping interfaces to MMA and GCMMA, suggested a callback storage support for optimizers with separate objective/constraint and gradient functions, and reported usability feedback on both the framework and the website
- Steve Choi added examples from the Schittkowski test problems for nonlinear optimization, and reported usability feedback
- Gaetan Kenway suggested the grouping/ungrouping of design variables, contributed a proof of concept for an history saving scheme, and reported usability feedback
- Benjamin Chalovich developed the setup build functionality to automate the wrapper compilation of optimizers and developed the initial setup install functionality
- Kenneth Moore (NASA) provided an interface between pyOpt and the openMDAO framework, tested the installation functionality including installation on a VirtualEnv, reported usability feedback and provided thread-safe changes for the SLSQP optimizer.
- Daniel Rudmin proofread and edited the documentation.
- Dr. Martin Schlueter provided a redistributed version of the MIDACO optimization code.
- Dr. K.A. Weinman (DLR) provided updated __init__ files to fix circular dependencies in the code.
- Mueen Nawaz, (Intel) provided a bug-fixing update for the SLSQP optimizer.
- Simon Rudolph (Technische Universitat Munchen) provided a bug-fixing update NLPQLP optimizer.
- Oliver Schmitt <oliver.schmitt@ltm.uni-erlangen.de> provided via Andrew Lambe installation procedures for the code under OpenSUSE systems.

- Herbert Schilling provided improvements to support unconstrained optimization problems for SNOPT, CONMIN and COBYLA.

CHAPTER

SEVEN

GLOSSARY

Ant Colony Optimization (ACO) Population-based stochastic global optimization algorithm based on the behavior of ant colonies, particularly their ability to collectively determine shortest paths through the cumulative affect of pheromones.

Automatic Differentiation A process for evaluating derivatives of a function that depends only on an algorithmic specification of the function, such as a computer program.

Constraint Restriction that a design variables must satisfy, typically denoted in a mathematical program standard form as an inequality, g(x) <= 0, or equality, h(x)=0.

Genetic algorithm (GA) Population-based stochastic global optimization algorithm inspired by the mechanisms of genetics, evolution, and survival of the fittest.

Global Maximum (or Minimum) A feasible solution that maximizes (or minimizes) the value of the objective function over the entire design space feasible region.

Global Optimizer Optimization method that implements an algorithm that is designed to find a globally optimal solution for various kinds of nonconvex programming problems.

Local Maximum (or Minimum) A feasible solution that maximizes (or minimizes) the value of the objective function within a local neighborhood of that solution.

Lower Bound A constraint that specifies a minimum feasible value of an individual design variable.

Numerical Optimization Mathematical techniques and procedures used to make a system or design as effective and/or functional as possible

Objective Function The (real-valued) function to be optimized, typically denoted in a mathematical program standard form as f.

Particle Swarm Optimization (PSO) Population-based stochastic global optimization algorithm based on the optimal swarm behavior of animals, like bird flocking and bees.

Sequential Linear Programming (SLP) Gradient-based local optimization algorithm based on solving successive first order approximations of a nonlinear programming problem objective subject to a linearization of the constraints. The linear approximations are usually done by using the first-order Taylor expansion.

Sequential Quadratic Programming (SQP) Gradient-based local optimization algorithm based on solving successive second order approximations of a nonlinear programming problem objective subject to a linearization of the constraints. The approximations are usually done by using the second-order Taylor expansion.

Sequential Unconstrained Minimization Technique (SUMT) Convex programming algorithm that convert the original constrained optimization problem to a sequence of unconstrained optimization problems whose optimal solutions converge to an optimal solution for the original problem.

Upper Bound A constraint that specifies a maximum feasible value of an individual design variable.

PYTHON MODULE INDEX

p

pyALGENCAN, 20
pyALHSO, 32
pyALPSO, 30
pyCOBYLA, 27
pyCONMIN, 24
pyFILTERSD, 21
pyFSQP, 17
pyGCMMA, 23
pyKSOPT, 26
pyMIDACO, 33
pyMMA, 22
pyMMFD, 25
pyNLPQL, 15
pyNLPQLP, 16
pyNSGA2, 31
pyOpt_constraint, 9
pyOpt_gradient, 9
pyOpt_history, 10
pyOpt_objective, 8
pyOpt_optimization, 3
pyOpt_optimizer, 7
pyOpt_variable, 9
pyPSQP, 19
pySDPEN, 28
pySLSQP, 18
pySNOPT, 13
pySOLVOPT, 29