OPTALG Documentation

Release 1.1.8

Tomas Tinoco De Rubira

Nov 30, 2019

Contents

Welcome! This is the documentation for OPTALG version 1.1.8, last updated Nov 30, 2019.

What is OPTALG?

OPTALG is a Python package that provides algorithms, wrappers, and tools for solving large and sparse optimization problems.

License

OPTALG is released under the BSD 2-clause license.

Contents

CHAPTER 1

Getting Started

This section describes how to get started with OPTALG.

1.1 Installation

In order to install OPTALG, the following tools are needed:

- Linux and Mac OS X:
  - C compiler
  - Make
  - Python (2.7 or 3.6)
  - pip
- Windows:
  - Anaconda (for Python 2.7)
  - MinGW (use `pip install -i https://pypi.anaconda.org/carlkl/simple mingwpy`)
  - 7-Zip (update system path to include the 7z executable, typically in `C:\Program Files\7-Zip`)

After getting these tools, the OPTALG Python module can be installed using:

```
pip install numpy cython
pip install optalg
```

By default, no wrappers are built for any external solvers. If the environment variable `OPTALG_IPOPT` has the value `true` during the installation, OPTALG will download and build the solver IPOPT for you, and then build its Python wrapper. Similarly, if the environment variables `OPTALG_CLP` amd `OPTALG_CBC` have the value `true` during the installation, OPTLAG will download and build the solvers Clp and Cbc for you, and then build their Python wrappers.

Note: Currently, the installation with Clp and Cbc does not work on Windows.

To install the module from source, the code can be obtained from https://github.com/ttinoco/OPTALG, and then the following commands can be executed on the terminal or Anaconda prompt from the root directory of the package:

```
pip install numpy cython
python setup.py install
```

Running the unit tests can be done with:

```
pip install nose
python setup.py build_ext --inplace
nosetests -s -v
```

CHAPTER 2

Optimization Solvers

In OPTALG, optimization solvers are objects of type `OptSolver`, and optimization problems are objects of type `OptProblem` and represent general problems of the form

$$\begin{aligned} \text{minimize} \quad & \varphi(x) & \\ \text{subject to} \quad & Ax = b & : \lambda \\ & f(x) = 0 & : \nu \\ & l \le x \le u & : \pi, \mu \\ & Px \in \mathbb{Z}^m, & \end{aligned}$$

where P is a matrix that extracts a sub-vector of x.

Before solving a problem with a specific solver, the solver parameters can be configured using the method `set_parameters()`. Then, the `solve()` method can be invoked with the problem to be solved as its argument. The status, optimal primal variables, and optimal dual variables can be extracted using the class methods `get_status()`, `get_primal_variables()`, and `get_dual_variables()`, respectively.

2.1 NR

This solver, which corresponds to the class `OptSolverNR`, solves problems of the form

$$\begin{aligned} \text{find} \quad & x \\ \text{subject to} \quad & Ax = b \\ & f(x) = 0 \end{aligned}$$

using the Newton-Raphson algorithm. It requires the number of variables to be equal to the number of constraints.

2.2 Clp and ClpCMD

These are wrappers of the solver Clp from COIN-OR. They corresponds to the classes *OptSolverClp* and *OptSolverClpCMD*, and solve problems of the form

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b \quad && : \lambda \\ & l \leq x \leq u \quad && : \pi, \mu. \end{aligned}$$

Linear optimization problems solved with these solvers must be instances of the class *LinProblem*, which is a subclass of *OptProblem*.

2.3 Cbc and CbcCMD

These are wrappers of the solver Cbc from COIN-OR. They correspond to the classes *OptSolverCbc* and *OptSolverCbcCMD*, and solve problems of the form

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & l \leq x \leq u \\ & Px \in \mathbb{Z}^m. \end{aligned}$$

Mixed-integer linear optimization problems solved with these solvers must be instances of the class *MixIntLinProblem*, which is a subclass of *OptProblem*.

2.4 CplexCMD

This is a wrapper of the solver CPLEX and uses a command-line interface. It corresponds to the class *OptSolverCplexCMD* and solves problems of type *MixIntLinProblem*.

2.5 IQP

This solver, which corresponds to the class *OptSolverIQP*, solves convex quadratic problems of the form

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} x^T H x + g^T x \\ \text{subject to} \quad & Ax = b \quad && : \lambda \\ & l \leq x \leq u \quad && : \pi, \mu \end{aligned}$$

using a primal-dual interior-point algorithm. Quadratic problems solved with this solver must be instances of the class *LinProblem*, which is a subclass of *OptProblem*. The following example shows how to solve the quadratic problem

$$\begin{aligned} \text{minimize} \quad & 3x_1 - 6x_2 + 5x_1^2 - 2x_1 x_2 + 5x_2^2 \\ \text{subject to} \quad & x_1 + x_2 = 1 \\ & 0.2 \leq x_1 \leq 0.8 \\ & 0.2 \leq x_2 \leq 0.8 \end{aligned}$$

using *OptSolverIQP*:

```
>>> import numpy as np
>>> from optalg.opt_solver import OptSolverIQP, QuadProblem

>>> g = np.array([3.,-6.])
>>> H = np.array([[10.,-2],
...               [-2.,10]])

>>> A = np.array([[1.,1.]])
>>> b = np.array([1.])

>>> u = np.array([0.8,0.8])
>>> l = np.array([0.2,0.2])

>>> problem = QuadProblem(H,g,A,b,l,u)

>>> solver = OptSolverIQP()

>>> solver.set_parameters({'quiet': True,
...                         'tol': 1e-6})

>>> solver.solve(problem)

>>> print solver.get_status()
solved
```

Then, the optimal primal and dual variables can be extracted, and feasibility and optimality can be checked as follows:

```
>>> x = solver.get_primal_variables()
>>> lam,nu,mu,pi = solver.get_dual_variables()

>>> print x
[ 0.20  0.80 ]

>>> print x[0] + x[1]
1.00

>>> print l <= x
[ True  True ]

>>> print x <= u
[ True  True ]

>>> print pi
[ 9.00e-01  1.80e-06 ]

>>> print mu
[ 1.80e-06  9.00e-01 ]

>>> print np.linalg.norm(g+np.dot(H,x)-np.dot(A.T,lam)+mu-pi)
1.25e-15

>>> print np.dot(mu,u-x)
2.16e-06

>>> print np.dot(pi,x-l)
2.16e-06
```

2.6 INLP

This solver, which corresponds to the class `OptSolverINLP`, solves general nonlinear optimization problems of the form

$$\begin{aligned} \text{minimize} \quad & \varphi(x) \\ \text{subject to} \quad & Ax = b \quad && : \lambda \\ & f(x) = 0 \quad && : \nu \\ & l \leq x \leq u \quad && : \pi, \mu \end{aligned}$$

using a primal-dual interior-point algorithm. It computes Newton steps for solving modified KKT conditions and does not have any global convergence guarantees.

2.7 AugL

This solver, which corresponds to the class `OptSolverAugL`, solves optimization problems of the form

$$\begin{aligned} \text{minimize} \quad & \varphi(x) \\ \text{subject to} \quad & Ax = b \quad && : \lambda \\ & f(x) = 0 \quad && : \nu \\ & l \leq x \leq u \quad && : \pi, \mu \end{aligned}$$

using an Augmented Lagrangian algorithm. It requires the objective function φ to be convex.

2.8 Ipopt

This is a wrapper of the solver IPOPT from COIN-OR. It corresponds to the class `OptSolverIpopt`, and solves optimization problems of the form

$$\begin{aligned} \text{minimize} \quad & \varphi(x) \\ \text{subject to} \quad & Ax = b \quad && : \lambda \\ & f(x) = 0 \quad && : \nu \\ & l \leq x \leq u \quad && : \pi, \mu. \end{aligned}$$

CHAPTER 3

API Reference

3.1 Linear Solvers

optalg.lin_solver.**new_linsolver**(*name='default'*, *prop='unsymmetric'*)
Creates a linear solver.

Parameters

name [string]

prop [string]

Returns

solver [LinSolver]

class optalg.lin_solver.lin_solver.**LinSolver**(*prop='unsymmetric'*)
Linear solver class.

Parameters

prop [{symmetric, unsymmetric}]

analyze(*self*, *A*)
Analyzes structure of A.

Parameters

A [matrix]

analyzed = None
Flag that specifies whether the matrix has been analyzed.

factorize(*self*, *A*)
Factorizes A.

Parameters

A [matrix]

factorize_and_solve (*self*, *A*, *b*)
Factorizes A and solves Ax=b.

Returns

x [vector]

is_analyzed (*self*)
Determine whether the matrix has been analyzed.

Returns

flags [{True, False}]

name = None
Name (string)

prop = None
Linear system property {'symmetric', 'unsymmetric'}.

solve (*self*, *b*)
Solves system Ax=b.

Parameters

b: vector

Returns

x [vector]

class optalg.lin_solver.mumps.**LinSolverMUMPS** (*prop='unsymmetric'*)
Linear solver based on MUMPS.

class optalg.lin_solver.superlu.**LinSolverSUPERLU** (*prop='unsymmetric'*)
Linear solver based on SuperLU.

class optalg.lin_solver.umfpack.**LinSolverUMFPACK** (*prop='unsymmetric'*)
Linear solver based on UMFPACK.

3.2 Optimization Problems

class optalg.opt_solver.problem.**OptProblem**
Class for representing general optimization problems.

Parameters

problem [Object]

A = None
Matrix for linear equality constraints

H_combined = None
Linear combination of Hessians of nonlinear constraints

Hphi = None
Objective function Hessian (lower triangular)

J = None
Jacobian of nonlinear constraints

P = None
Integer flags (boolean array)

b = None
Right-hand side for linear equality constraints

combine_H (*self*, *coeff*, *ensure_psd=False*)
Forms and saves a linear combination of the individual constraint Hessians.

Parameters

coeff [vector]

ensure_psd [{`True`,``False``}]

eval (*self*, *x*)
Evaluates the objective value and constraints at the give point.

Parameters

x [vector]

f = None
Nonlinear equality constraint function

get_num_linear_equality_constraints (*self*)
Gets number of linear equality constraints.

Returns

num [int]

get_num_nonlinear_equality_constraints (*self*)
Gets number of nonlinear equality constraints.

Returns

num [int]

get_num_primal_variables (*self*)
Gets number of primal variables.

Returns

num [int]

gphi = None
Objective function gradient

l = None
Lower limits

lam = None
Lagrande multipliers for linear equality constraints

mu = None
Lagrande multipliers for upper limits

nu = None
Lagrande multipliers for nonlinear equality constraints

phi = None
Objective function value

pi = None
Lagrande multipliers for lower limits

recover_dual_variables (*self*, *lam*, *nu*, *mu*, *pi*)
Recovers dual variables for original problem.

Parameters

lam [ndarray]

nu [ndarray]

mu [ndarray]

pi [ndarray]

recover_primal_variables (*self*, *x*)

Recovers primal variables for original problem.

Parameters

x [ndarray]

show (*self*, *inf=100000000.0*)

Displays information about the problem.

to_lin (*self*)

Converts problem to linear problem.

Returns

p [*LinProblem*]

to_mixintlin (*self*)

Converts problem to mixed integer linear problem.

Returns

p [*MixIntLinProblem*]

to_quad (*self*)

Converts problem to quadratic problem.

Returns

p [*LinProblem*]

u = None

Upper limits

wrapped_problem = None

Wrapped problem

x = None

Initial point

class optalg.opt_solver.problem_lin.**LinProblem** (*c*, *A*, *b*, *l*, *u*, *x=None*, *lam=None*, *mu=None*, *pi=None*)

Linear program class.

Parameters

c [vector]

A [matrix]

l [vector]

u [vector]

x [vector]

class optalg.opt_solver.problem_mixintlin.**MixIntLinProblem**(*c*, *A*, *b*, *l*, *u*, *P*, *x=None*)

Mixed integer linear program class.

Parameters

c [vector]

A [matrix]

l [vector]

u [vector]

P [boolean array]

class optalg.opt_solver.problem_quad.**QuadProblem**(*H*, *g*, *A*, *b*, *l*, *u*, *x=None*, *lam=None*, *mu=None*, *pi=None*)

Quadratic program class.

Parameters

H [symmetric matrix]

g [vector]

A [matrix]

l [vector]

u [vector]

x [vector]

3.3 Optimization Solvers

class optalg.opt_solver.opt_solver.**OptSolver**

Optimization solver class.

add_callback(*self*, *c*)

Adds callback funtion to solver.

Parameters

c [Function]

add_termination(*self*, *t*)

Adds termination condition to solver.

Parameters

t [Function]

callbacks = None

List of callback functions.

fdata = None

Function data container.

get_dual_variables(*self*)

Gets dual variables.

Returns

lam [vector]

nu [vector]

mu [vector]

pi [vector]

get_error_msg(*self*)
Gets solver error message.

Returns

message [string]

get_iterations(*self*)
Gets number of iterations.

Returns

iters [int]

get_primal_variables(*self*)
Gets primal variables.

Returns

variables [ndarray]

get_results(*self*)
Gets results.

Returns

results [dictionary]

get_status(*self*)
Gets solver status.

Returns

status [string]

info_printer = None
Information printer (function).

is_status_solved(*self*)
Determines whether the solver solved the given problem.

Returns

flag [{`True`, `False`}]

line_search(*self*, *x*, *p*, *F*, *GradF*, *func*, *smax=inf*, *maxiter=40*)
Finds steplength along search direction p that satisfies the strong Wolfe conditions.

Parameters

x [current point (ndarray)]

p [search direction (ndarray)]

F [function value at *x* (float)]

GradF [gradient of function at *x* (ndarray)]

func [function of *x* that returns function object with attributes *F* and *GradF* (function)]

smax [maximum allowed steplength (float)]

Returns

s [stephlength that satisfies the Wolfe conditions (float).]

parameters = None
Parameters dictionary.

reset (*self*)
Resets solver data.

set_error_msg (*self*, *msg*)
Sets solver error message.

Parameters

msg [string]

set_info_printer (*self*, *printer*)
Sets function for printing algorithm progress.

Parameters

printer [Function.]

set_parameters (*self*, *parameters*)
Sets solver parameters.

Parameters

parameters [dict]

set_status (*self*, *status*)
Sets solver status.

Parameters

status [string]

solve (*self*, *problem*)
Solves optimization problem.

Parameters

problem [OptProblem]

supports_properties (*self*, *properties*)
Checks whether solver supports properties.

Parameters

properties: list

Returns

flag [{`True`, `False`}]

terminations = None
List of termination conditions.

class optalg.opt_solver.nr.**OptSolverNR**
Newton-Raphson algorithm.

class optalg.opt_solver.iqp.**OptSolverIQP**
Interior-point quadratic program solver.

class optalg.opt_solver.inlp.**OptSolverINLP**
Interior-point non-linear programming solver.

class optalg.opt_solver.augl.**OptSolverAugL**
Augmented Lagrangian algorithm.

class optalg.opt_solver.ipopt.**OptSolverIpopt**
Interior point nonlinear optimization algorithm from COIN-OR.

class optalg.opt_solver.clp.**OptSolverClp**
Linear programming solver from COIN-OR.

class optalg.opt_solver.clp_cmd.**OptSolverClpCMD**
Linear programming solver from COIN-OR (via command-line interface, version 1.15.3).

class optalg.opt_solver.cbc.**OptSolverCbc**
Mixed integer linear "branch and cut" solver from COIN-OR.

class optalg.opt_solver.cbc_cmd.**OptSolverCbcCMD**
Mixed integer linear "branch and cut" solver from COIN-OR (via command-line interface, version 2.8.5).

class optalg.opt_solver.cplex_cmd.**OptSolverCplexCMD**
CPLEX solver interface (via command-line interface).

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

A

B

C

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

W

X