OPTALG Documentation

Release 1.0

Tomas Tinoco De Rubira

March 13, 2016

Welcome! This is the documentation for OPTALG, last updated March 13, 2016.

What is OPTALG?

OPTALG is a Python package with optimization algorithms.

License

OPTALG is released under the BSD 2-clause license.

Contact

If you have any questions about OPTALG or if you are interested in collaborating, send me an email:

- Tomas Tinoco De Rubira (ttinoco5687@gmail.com).

Documentation Contents

CHAPTER

ONE

GETTING STARTED

This section describes how to get started with OPTALG. In particular, it covers required packages, installation, and provides a quick example showing how to use this package.

- Numpy (>=1.8.2)
- Scipy (>=0.13.3)
- MUMPS (==4.10) (optional)

1.1 Download

The latest version of OPTALG can be downloaded from https://github.com/ttinoco/OPTALG.

1.2 Installation

The OPTALG Python module can be installed using:

```
> sudo python setup.py install
```

from the root directory of the package. If MUMPS is not available, then the option `--no_mumps` should be added to the above command.

1.3 Example

As a quick example of how to use OPTALG, consider the task of solving a quadratic program. This can be done as follows:

```
>>> coming soon
```

CHAPTER TWO

OPTIMIZATION SOLVERS

In OPTALG, optimization solvers are objects derived from the `OptSolver` class, and optimization problems are objects derived from the `OptProblem` class, which represents general problems of the form

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

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 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 solves problems of the form

$$\begin{array}{ll} \text{find} & x \\ \text{subject to} & Ax = b \\ & f(x) = 0. \end{array}$$

using the Newton-Raphson algorithm.

2.2 IQP

This solver, which is represented by the class `OptSolverIQP`, solves convex quadratic problems of the form

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

using an interior point method. Quadratic problems solved with this solver must be objects derived from the class `QuadProblem`, which is a subclass of `OptProblem`. The following example shows how to solve the quadratic problem

$$\begin{aligned} \text{minimize} \quad & 3x_1 - 6x_2 + 5x_1^2 - 2x_1x_2 + 5x_2^2 \\ \text{subject to} \quad & x_1 + x_2 = 1 \\ & 0.2 \leq x_1 \leq 0.8 \\ & 0.2 \leq x_2 \leq 0.8 \end{aligned}$$

using the `OptSolverIQP` solver:

```
>>> 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
```

2.3 LCCP

This solver solves convex linearly-constrained problems of the form

$$\begin{array}{lll} \text{minimize} & \varphi(x) & \\ \text{subject to} & Ax = b & : \lambda \\ & l \leq x \leq u. & : \pi, \mu \end{array}$$

using an interior point method.

2.4 AugL

This solver solves convex or non-convex optimization problems of the form

$$\begin{array}{lll} \text{minimize} & \varphi(x) & \\ \text{subject to} & Ax = b & : \lambda \\ & f(x) = 0. & : \nu \end{array}$$

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

CHAPTER

THREE

API REFERENCE

3.1 Linear Solvers

`optalg.lin_solver.`**`new_linsolver`**`(`*name*, *prop*`)`

Creates a linear solver.

Parameters **name** : string

prop : string

Returns **solver** : `LinSolver`

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

Linear solver class.

Parameters **prop** : {`symmetric`, `unsymmetric`}

`analyze``(`*A*`)`

Analyzes structure of A.

Parameters **A** : matrix

`analyzed` **= None**

Flag that specifies whether the matrix has been analyzed.

`factorize``(`*A*`)`

Factorizes A.

Parameters **A** : matrix

`factorize_and_solve``(`*A*, *b*`)`

Factorizes A and solves Ax=b.

Returns **x** : vector

`is_analyzed``()`

Determine whether the matrix has been analyzed.

Returns **flags** : {`True`, `False`}

`prop` **= None**

Linear system property {`'symmetric'`, `'unsymmetric'`}.

`solve``(`*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.

3.2 Optimization Solvers

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

`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

`b` **= None**
Right-hand side for linear equality constraints

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

`f` **= None**
Nonlinear equality constraint function

`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

`show`()
Displays information about the problem.

`u` **= None**
Upper limits

`x` **= None**
Initial point

class `optalg.opt_solver.`**`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

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

Optimization solver class.

`add_callback`(*c*)

Adds callback funtion to solver.

Parameters **c** : Function

`add_termination`(*t*)

Adds termination condition to solver.

Parameters **t** : Function

`callbacks` **= None**

List of callback functions.

`fdata` **= None**

Function data container.

`get_dual_variables`()

Gets dual variables.

Returns **lam** : vector

nu : vector

mu : vector

pi : vector

`get_error_msg`()

Gets solver error message.

Returns **message** : string

`get_iterations`()

Gets number of iterations.

Returns **iters** : int

`get_primal_variables`()

Gets primal variables.

Returns **variables** : ndarray

`get_results`()

Gets results.

Returns **results** : dictionary

`get_status`()

Gets solver status.

Returns **status** : string

`info_printer` = None
Information printer (function).

`is_status_solved`()
Determines whether the solver solved the given problem.

Returns **flag** : {`True`, `False`}

`line_search`(*x*, *p*, *F*, *GradF*, *func*, *smax=inf*)
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`()
Resets solver data.

`set_error_msg`(*msg*)
Sets solver error message.

Parameters **msg** : string

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

Parameters **printer** : Function.

`set_parameters`(*parameters*)
Sets solver parameters.

Parameters **parameters** : dict

`set_status`(*status*)
Sets solver status.

Parameters **status** : string

`solve`(*problem*)
Solves optimization problem.

Parameters **problem** : OptProblem

`terminations` = None
List of termination conditions.

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

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

class `optalg.opt_solver.`**`OptSolverLCCP`**

Interior-point solver for linearly-constrained convex programs.

class `optalg.opt_solver.`**`OptSolverAugL`**

Augmented Lagrangian algorithm.

3.3 Stochastic Optimization Solvers

CHAPTER FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

PYTHON MODULE INDEX