Linearly Constrained Convex Programming with Python

October 14, 2017

1 Introduction

This document describes how to use COPL_LC from Python. COPL_LC is a general purpose optimization solver for linearly constrained convex programming. It is written in Fortran and developed in Computational Optimization Laboratory, Department of Management Science, The University of Iowa, Iowa City, IA 52242, USA.

2 Problem Definition and Algorithm Solution

The LCCP problem can be expressed as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0 \end{aligned} \tag{1}$$

where $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^{m \times n}$ and $f(x)$ is a continuous convex function.

To obtain solution of (1) with high precision, COPL_LC solver applies primal dual interior point method. Interior point methods ([2],[3]) have been well studied and achieved great success in the last century. COPL_LC applies a primal dual homogeneous algorithm ([1]) for implementation. It has the benefit that the algorithm will generate a solution when the problem is solvable, or will detect infeasibility or unboundedness. Moreover, advanced implementation techniques and sparse matrix factorization are applied to take advantage of structures of Hessian matrices. To see more details of algorithm implementation, please refer to the source code. This is the Python interface of COPL_LC, which is developed and maintained by SHUFE solver team.

3 Installation

The Python code lies in `lccp.py` file. In order to compile the code, run makefile as follows:

```
make all
```

To remove the .so file, simply run:

```
make clean
```

4 Optimization Models

To run COPL_LC, users need to define the following parameters.

- A: the matrix in linear constraint in `csr_sparse` format.
- b: the vector in linear constraint in Python `list` or `numpy` array formats.
- H: the upper half of the Hessian matrix in `csr_sparse` format.

After running the code, the triplet $(x, y,$ `status`$)$ is returned.

- x: a vector of primal solution.
- y: a vector of dual solution.
- `status`: an integer about the solver condition. [0=optimal, 1=primal unbounded, 2=dual unbounded, 3=iteration maximum, 4=numerical difficulty, 5=primal or dual infeasible, 6=insufficient space]

In this section, we provide subroutines to run code for the following models:

4.1 Linear Programming

Linear programming solves problems of the following form:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0 \end{aligned} \tag{2}$$

It should be noted that for linear programming, there is no need to set Hessian matrix.

COPL_LC for linear programming

```
1
2  def lclp(A, b, c):
3      """LCCP for Linear Programming
4          min  <c, x>
5          s.t.  A x = b
6                   x >= 0
7      Args:
8          A: m by n sparse matrix
9          b, c: list or numpy array object
10     """
```

In the following demo, we show how to call `lclp` for solving linear programming problems. We construct a simple case of A, b, c using the following code.

Example

$$\begin{aligned} \min \quad & -x_1 - x_2 \\ \text{subject to} \quad & x_1 + 2x_2 = 3, 2x_1 + x_2 = 3, x_1 \geq 0, x_2 \geq 0 \end{aligned}$$

We input codes below to solve this problem.

```
In [1]: from lccp import lclp
        import numpy as np
        from scipy.sparse import csr_matrix
        import lccp_lp

In [2]: rows = [0, 0, 1, 1]
        cols = [0, 1, 0, 1]
        data = [1, 2, 2, 1]
        A = csr_matrix((data, (rows, cols)), shape=(2, 2))
        b = [3, 3]
        c = [-1, -1]

In [3]: x,y,status=lclp(A, b, c)

In [4]: print(x)

[ 1.  1.]


In [5]: print(y)

[-0.33333333 -0.33333333]


In [6]: print(status)

0
```

The following returned message summarizes the optimization progress.

```
1  ================================================
2  COPL_LC --- Linearly Constrained Convex Programming
3              by  COL, U of Iowa
4  ================================================
5
6
7                    The LCCP model
8  ------------------------------------------------
9
10                   variables   :          2
11                   constraints:          2
12                 nonzeros in A:          4
13                 nonzeros in H:          0
14
15               COPL_LC Running Progress
16 ------------------------------------------------
17
18 --iter =   0  Complementarity gap  = 1.000E+00
19               Primal obj. value    =-2.00000000000000E+00
20               Dual   obj. value    = 0.00000000000000E+00
21               Primal feas. residual= 0.000E+00
22               Dual   feas. residual= 1.333E+00
23
24 --iter =   1  Complementarity gap  = 5.000E-04
25               Primal obj. value    =-2.00000000000000E+00
26               Dual   obj. value    =-1.99899995326996E+00
27               Primal feas. residual= 1.480E-16
```

```
                Dual    feas. residual= 1.998E-03

--iter =   2  Complementarity gap  = 2.500E-07
                Primal obj. value     =-2.00000000000000E+00
                Dual    obj. value    =-1.99999949995327E+00
                Primal feas. residual= 1.480E-16
                Dual    feas. residual= 1.000E-06

--iter =   3  Complementarity gap  = 1.250E-10
                Primal obj. value     =-2.00000000000000E+00
                Dual    obj. value    =-1.99999999974997E+00
                Primal feas. residual= 1.480E-16
                Dual    feas. residual= 5.001E-10

--iter =   4  Complementarity gap  = 6.251E-14
                Primal obj. value     =-2.00000000000000E+00
                Dual    obj. value    =-1.99999999999988E+00
                Primal feas. residual= 2.961E-16
                Dual    feas. residual= 2.500E-13

-- exit : optimality is obtained :) on iter          4

          Primal obj. value     =-2.00000000000000E+00
          Dual    obj. value    =-1.99999999999988E+00
          Primal feas. residual= 2.961E-16
          Dual    feas. residual= 2.500E-13

            Primal and Dual Solutions
-------------------------------------------------------

   j               x(j)                z(j)              r_D(j)
   1 0.100000000000E+01 0.625116833294E-13 -.125011112573E-12
   2 0.100000000000E+01 0.625116833294E-13 -.125011112573E-12

   i               y(i)              r_P(i)
   1 -.333333333333E+00 -.888178419700E-15
   2 -.333333333333E+00 0.000000000000E+00

          Termination Status and Statistics
-------------------------------------------------------

Program terminated where termination status  =    0
0=optimal, 1=primal unbounded, 2=dual unbounded
3=iteration maximum, 4=numerical difficulty
5=primal or dual infeasible, 6=insufficient space
7=data file error

     time for data readin =       0.00
          problem solving =       0.00
    total time in seconds =       0.00
```

4.2 Quadratic Programming

Quadratic programming solves problems of the following form:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}x^T Hx + c^T x \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0 \end{aligned} \tag{3}$$

where $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^{m\times n}$ and H is a positive semi-definite matrix. COPL_LC uses very efficient data structures, one only needs to enter the upper-triangle of H.

COPL_LC for quadratic programming

```
1
2  def lcqp(A, H, b, c):
3      """LCCP for Qudratic Programming
4          min    1/2 x^THx + c^Tx
5          s.t.     A x = b
6                     x >= 0
7      Args:
8          A: m by n sparse matrix
9          H: n by n Hessian matrix, in csr_matrix format. This is used to
10         allocate space for the Hessian matrix.
11         b, c: list or numpy array object
12     """
```

In the following demo, we show how to call `lcqp` for solving quadratic programming problems. We construct a simple case of A, b, c and H using the following code.

Example

$$\begin{aligned} \min \quad & (1/2)\|x\|^2 + e^{\mathrm{T}}x \\ \text{subject to} \quad & e^{\mathrm{T}}x = 1, x \geq 0 \end{aligned}$$

We input code below to solve this problem.

```
In [1]: from lccp import lcqp
        import numpy as np
        from scipy.sparse import csr_matrix
        import lccp_qp

In [2]: rows = [0] * 10
        cols = np.arange(10)
        A = csr_matrix(([1.0] * 10, (rows, cols)))
        indptr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], dtype=np.int64)
        indices = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=np.int64)
        data = np.array([1.0] * 10)
        H = csr_matrix((data, indices, indptr), shape=(10, 10))
        b = [1.0]
        c = [1.0] * 10

In [3]: x,y,status=lcqp(A, H, b, c)

In [4]: print(x)

[ 0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1]


In [5]: print(y)

[ 1.1]


In [6]: print(status)

0
```

The following returned message summarizes the optimization progress.

```
1  ==================================================
2  COPL_LC --- Linearly Constrained Convex Programming
3              by  COL, U of Iowa
4  ==================================================
5
6
7                    The LCCP model
8  ----------------------------------------------------
9
10                  variables   :          10
11                  constraints:           1
12                nonzeros in A:          10
13                nonzeros in H:          10
14
15               COPL_LC Running Progress
16 ----------------------------------------------------
17
18 --iter =   0  Complementarity gap  = 2.000E+00
19               Primal obj. value    = 1.50000000000000E+01
20               Dual   obj. value    =-5.00000000000000E+00
21               Primal feas. residual= 8.182E-01
22               Dual   feas. residual= 0.000E+00
23
24 --iter =   1  Complementarity gap  = 1.800E-01
```

```
                Primal obj. value    = 1.05000000000000E+00
                Dual   obj. value    =-7.50000000000000E-01
                Primal feas. residual= 4.441E-16
                Dual   feas. residual= 0.000E+00

--iter =   2  Complementarity gap  = 9.000E-05
                Primal obj. value    = 1.05000000000000E+00
                Dual   obj. value    = 1.04909995794296E+00
                Primal feas. residual= 5.551E-17
                Dual   feas. residual= 0.000E+00

--iter =   3  Complementarity gap  = 4.500E-08
                Primal obj. value    = 1.05000000000000E+00
                Dual   obj. value    = 1.04999954995794E+00
                Primal feas. residual= 5.551E-17
                Dual   feas. residual= 0.000E+00

--iter =   4  Complementarity gap  = 2.250E-11
                Primal obj. value    = 1.05000000000000E+00
                Dual   obj. value    = 1.04999999977497E+00
                Primal feas. residual= 0.000E+00
                Dual   feas. residual= 0.000E+00

--iter =   5  Complementarity gap  = 1.126E-14
                Primal obj. value    = 1.05000000000000E+00
                Dual   obj. value    = 1.04999999999989E+00
                Primal feas. residual= 0.000E+00
                Dual   feas. residual= 0.000E+00

-- exit : optimality is obtained :) on iter          5

        Primal obj. value    = 1.05000000000000E+00
        Dual   obj. value    = 1.04999999999989E+00
        Primal feas. residual= 0.000E+00
        Dual   feas. residual= 0.000E+00

          Primal and Dual Solutions
-------------------------------------------------

  j               x(j)                z(j)              r_D(j)
  1 0.100000000000E+00 0.112576614697E-12 0.000000000000E+00
  2 0.100000000000E+00 0.112576614697E-12 0.000000000000E+00
  3 0.100000000000E+00 0.112576614697E-12 0.000000000000E+00
  4 0.100000000000E+00 0.112576614697E-12 0.000000000000E+00
  5 0.100000000000E+00 0.112576614697E-12 0.000000000000E+00
  6 0.100000000000E+00 0.112576614697E-12 0.000000000000E+00
  7 0.100000000000E+00 0.112576614697E-12 0.000000000000E+00
  8 0.100000000000E+00 0.112576614697E-12 0.000000000000E+00
  9 0.100000000000E+00 0.112576614697E-12 0.000000000000E+00
 10 0.100000000000E+00 0.112576614697E-12 0.000000000000E+00

  i               y(i)              r_P(i)
  1 0.110000000000E+01 0.000000000000E+00

        Termination Status and Statistics
-------------------------------------------------

Program terminated where termination status  =    0
```

```
  0=optimal, 1=primal unbounded, 2=dual unbounded
  3=iteration maximum, 4=numerical difficulty
  5=primal or dual infeasible, 6=insufficient space
  7=data file error

       time for data readin =        0.00
            problem solving =        0.00
      total time in seconds =        0.00
[ 0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1]
[ 1.1]
0
```

4.3 General Convex Programming

COPL_LC can be applied to general convex programs as well. For achieving this goal, users should define three functions in Fortran:

1. a function to compute the objective,
2. a function to compute the gradient vector,
3. a function to compute the Hessian matrix.

The above functions should be defined in `fun_lc.f`. Next, run the following command

```
make lc
```

In the future, we will provide an interface for users to define functions in Python.

References

[1] Erling D Andersen and Yinyu Ye. On a homogeneous algorithm for the monotone complementarity problem. *Mathematical Programming*, 84(2):375–399, 1999.

[2] Stephen J Wright. *Primal-dual interior-point methods*. SIAM, 1997.

[3] Yinyu Ye. Interior point algorithms-theory and analysis., 1998.