MATLAB code for CLAD with MIP model

K. Florios
Department of International and European Economic Studies,
Athens University of Economics and Business
cflorios@aueb.gr

May 7, 2016

CladCompute.m

```
function [value,estimates,time,quality]=CladCompute()

%Main Program: Computes Max score by defining a MILP and calling milp.m
tic
[X,y,w]=readXyw();
[X,mu,sigma]=standardizeX(X,y);
[c,A,b]=definecAb(X,y,w);
[lb,ub, Aeq, beq, n, p, best]=definelbub(X,y);
%[x,cost,feasible]=milp(c,A,b,Aeq,beq,lb,ub,n, best);
 [x,cost,feasible, time]=milp_cplex(c,A,b,Aeq,beq,lb,ub,n, p, best);
estimatesNorm=x
value=cost
status=feasible
runtime=time

estimatesRaw=denormalizeEstimates(estimatesNorm,mu,sigma)
estimates=estimatesRaw


%estimatesRaw=denormalizeEstimates(estimatesNorm,mu,sigma)

estimates=estimatesRaw
value=cost
time=toc
quality=feasible;

end
```

definecAb.m

```
function [c,A,b]=definecAb(X,y,w)

%Defines c,A,b for milp.m

n=size(X,1);
p=size(X,2);

c3=repmat(0,1,n); %gammas integer
c1=repmat(0,1,p);  %betas
c2=repmat(0,1,n);   %phis
c4=repmat(1,1,n);    %sm's
c5=repmat(1,1,n);     %sp's

c=[c3 c1 c2 c4 c5];

d=20;
```

```
%d=5;
for i=1:1:n
    M(i)=abs(X(i,1))*d+abs(X(i,2:end))*repmat(d,1,p-1)';
end

for i=1:n       %constraint 3c in pdf ssrn
   A1(i,1:n)=0; %gammas
   for j=1:p
   A1(i,(n+j))=X(i,j); %betas
   end
   A1(i,(n+p+1):(2*n+p))=-1.*(1:n==i); %phis
   A1(i,(2*n+p+1):(3*n+p))=0;         %sm's
   A1(i,(3*n+p+1):(4*n+p))=0;          %sp's
end

for i=1:n
   b1(i)=0;
end

for i=1:n     %constraint 3e in pdf ssrn
   A2(i,1:n)=M(i).*(1:n==i); %gammas
   for j=1:p
   A2(i,(n+j))=-X(i,j); %betas
   end
   A2(i,(n+p+1):(2*n+p))=1.*(1:n==i);    %phis
   A2(i,(2*n+p+1):(3*n+p))=0;    %sm's
   A2(i,(3*n+p+1):(4*n+p))=0;     %sp's
end

for i=1:n
   b2(i)=M(i);
end

for i=1:n     %constraint 3f in pdf ssrn
  A3(i,1:n)=-M(i).*(1:n==i); %gammas
  A3(i,(n+1):(n+p))=0; %betas
  A3(i,(n+p+1):(2*n+p))=1.*(1:n==i); %phis
  A3(i,(2*n+p+1):(3*n+p))=0; %sm's
  A3(i,(3*n+p+1):(4*n+p))=0; %sp's
end

for i=1:n
   b3(i)=0; % for left censoring at 0
end

A= [A1 ; A2 ; A3];
b= [b1 b2  b3]';

end
```

definelbub.m

```
function [lb,ub, Aeq, beq, n, p, best]=definelbub(X,y)

%Defines lb,ub for milp.m
 d=20;
%d=10;
%d=5;
n=size(X,1);
p=size(X,2);

lb1=repmat(0,1,n);   %gammas
lb2=repmat(-d,1,p);   %betas
%lb3=repmat(-Inf,1,n);  %phis
lb3=repmat(0,1,n);       %phis, left censoring at zero (0)
lb4=repmat(0,1,n);       %sm's
lb5=repmat(0,1,n);        %sp's

lb=[lb1 lb2 lb3 lb4 lb5];

BIGNUM=+Inf;

ub1=repmat(1,1,n);   %gammas
ub2=repmat(d,1,p);    %betas
ub3=repmat(+BIGNUM,1,n); %phis
ub4=repmat(+BIGNUM,1,n);  %sm's
ub5=repmat(+BIGNUM,1,n);   %sp's

ub=[ub1 ub2 ub3 ub4 ub5];


for i=1:n
   Aeq(i,1:n)=0; %gammas
   Aeq(i,(n+1):(n+p))=0; %betas
   Aeq(i,(n+p+1):(2*n+p))=-1.*(1:n==i); %phis
   Aeq(i,(2*n+p+1):(3*n+p))=1.*(1:n==i); %sm's
   Aeq(i,(3*n+p+1):(4*n+p))= -1.*(1:n==i); %sp's
end

for i=1:n
   beq(i)= -y(i);
end


best=+Inf;


end
```

milp.m

```matlab
function [x,cost,feasible]=milp(c,A,b,Aeq,beq,lb,ub,n, best);
% Solves a integer lp using branch and bound proceeding in a
% depth-first manner (recursive)
% c: is cost      A: is constraint matrix b: is constraint vector
% lb: lower bound   ub: upper bound n: number of integer variables
% best: is best solution so far
% Note this traverses along the right most branch
% unlike the usual way of checking both branches before continuing. Also,
% it assumes first n variables must be integer.
% Written by Kevin Tomsovic
% Note: Last accessed, Autumn 2007, from K. Tomsovic's homepage.
% Note: by K.Florios, Date: 4.4.2016

% Initialization
feasible=0;       % 1 - Feasible, 0 - Infeasible
tol=1e-3;         % Tolerance on integers
%%%tol=1e-6;         % Tolerance on integers
cost=inf;

% Solve current linprog, suppress error messages
options=optimset('display','off');
[x,cost,how]=linprog(c,A,b,Aeq,beq,lb,ub,[],options);

% If infeasible or cost is greater than bound (best solution found)
%   then stop and return with infinite cost
if how<=0 | cost>best, cost=inf; return; end; %KJF comments out 1.4.2016

% Take first non integer u and branch
branch=0;
for i=1:n  % Split LP on first non-integer u
   if abs(x(i)-round(x(i)))>tol, branch=i; break; end;
end;

% If integer solution then return optimal solution
if branch==0, feasible=1; % If valid integer solution (branch==0), return
   return;
else % Solve subproblems by putting new upper and lower bounds on branch variable
      % Proceeding depth-first
   lba=lb; uba=ub;
   uba(branch)=floor(x(branch)); lba(branch)=ceil(x(branch));
   % Solve left branch (x<=floor(x))
   [x,cost,feasible]=milp(c,A,b,Aeq,beq,lb,uba,n,best);

   % Solve right branch (x>=ceil(x))with new best
   best=min(cost,best);
   [xb,cost,how]=milp(c,A,b,Aeq,beq,lba,ub,n,best);
   if cost<best, x=xb; % Better solution found
```

```
   else cost=best;
   end;
   feasible=feasible | how; % Feasible solution found?
end;
return;
```

milp_cplex.m

```
function [x,deviation,feasible, time]=milp_cplex(c,A,b,Aeq,beq,lb,ub,n, p, best);
% Solves a mixed integer lp using gurobi 5.0.1
% c: is objective function coefficients A: is constraint matrix b: is constraint
    vector
% lb: lower bound ub: upper bound n: number of 0-1 variables
% best: is best solution so far
% Note this uses the MATLAB/Gurobi Interface documented at
% http://www.gurobi.com/documentation/5.0/reference-manual/node650
% Also, it assumes first n variables must be integer.
% The MIP equations of the maximum score estimator are available at
% Florios. K, Skouras, S. (2008) Exact computation of maximum weighted
% score estimators, Journal of Econometrics 146, 86-91.
% Written by Kostas Florios, July 20, 2012
%
% gurobi 5.0.1
% cd c:/Users/jones/gurobi500/win64/matlab
% gurobi_setup
%
% cd C:\gurobi501\win32\matlab
% gurobi_setup
%
% cplex 12.6
% addpath('C:\Program
    Files\IBM\ILOG\CPLEX_Studio_Preview126\cplex\matlab\x86_win32')
% Greg corei3 2016
% addpath('D:\Program
    Files\IBM\ILOG\CPLEX_Studio_Preview126\cplex\matlab\x86_win32')

model.Aineq = sparse(A) ;
model.f = c ;
model.bineq = b ;
model.Aeq = sparse(Aeq) ;
model.beq = beq' ;
model.lb = lb ;
model.ub = ub ;
%model.ctype = [repmat('B',size(b,2),1) ; repmat('C',size(c,2)-size(b,2),1)]' ;
model.ctype = [repmat('B',n,1) ; repmat('C',p,1) ; repmat('C',n,1) ;
    repmat('C',n,1) ; repmat('C',n,1)]' ;


opt = cplexoptimset('cplex') ;
```

```
opt.mip.display = 4 ;
opt.mip.interval = 1000 ;

opt.timelimit= 1800 ;
opt.mip.tolerances.mipgap = 0.00 ;
opt.parallel = 1 ;
opt.threads = 1 ;
opt.mip.strategy.file = 3 ;
opt.workmem = 1024 ;
opt.emphasis.mip = 3 ;

opt.exportmodel =
    'G:\KOSTAS\Core-i5-Laptop\matlab2011\milp-cplex-2016-for-CLAD\myModel.lp' ;

model.options = opt;


[x,fval,exitflag,output] = cplexmilp(model) ;

  fprintf('Optimization returned status: %s\n', output.message);
  fprintf('Objective Value: %e\n', fval);
  fprintf('(Wall clock) Time elapsed (s): %e\n', output.time);
  fprintf('Decision variables: show only the betas\n');
  %disp(x)


x=x((n+1):(n+p))
deviation=fval;
feasible=output.message;
time=output.time;
return;
```

---

readXyw.m

```
function [X,y,w]=readXyw()

%Reads X,y,w of given max score problem

X=load('X_200_4.txt');
%X=[X(:,2) X(:,3) X(:,4) X(:,5)];
X=X(:,2:end);
y=load('ys_200_4.txt');
y=[y(:,2)];
%w=load('w_numeric5.txt');
%w=[w(:,2)];
w=repmat(1,size(X,1),1);

end
```

---

denormalizeEstimates.m

```
function [estimatesRaw]=denormalizeEstimates(estimatesNorm,mu,sigma);

%denormalized estimatesNorm obtained by Gurobi MIP to estimatesRaw, which
%are meaningful to the user

%quick and dirty implementation, based on GAMS and Fortran Analogues
p=size(estimatesNorm,1);
betaNorm=estimatesNorm;

for j=1:1:p
    betaRaw(j)=0;
    betaHelp(j)=0;
end

for j=1:1:p
    if (sigma(j) ~=0)
        betaHelp(j)= betaNorm(j)./sigma(j);
    end
    if (sigma(j) ==0)
        for jj=1:1:p
            if (sigma(jj) ~=0)
                betaHelp(j)=betaHelp(j)-betaNorm(jj).*mu(jj)./sigma(jj) ;
            else
                jj0=jj;
            end
        end
        betaHelp(j)=betaHelp(j)+betaNorm(jj0);
    end
end

for j=1:1:p
   %%betaRaw(j)=betaHelp(j)./betaHelp(1);
   betaRaw(j)=betaHelp(j);
end

estimatesRaw=betaRaw;

end
```

standardizeX.m

```
function [X,mu,sigma]=standardizeX(X,y)

%Standardizes X
mu=mean(X);
sigma=std(X);
```

```
testX=(X-repmat(mu,size(X,1),1)) ./ repmat(sigma,size(X,1),1);
p=size(X,2);
for j=1:1:p
if isnan(testX(:,j))
   X(:,j) = X(:,j)
else
    X(:,j) = testX(:,j)
end

end
```

1 Contained functions

1. **CladCompute.m** → main
2. **definecAb.m** → defines matrices c,A,b
3. **definelbub.m** → defines bounds lb,ub and matrices Aeq,beq
4. **denormalizeEstimates.m** → post-processing
5. **milp.m** → calls linprog
6. **milp_cplex.m** → calls cplexmilp commercial function
7. **readXyw.m** → input from files X.txt and ys.txt
8. **standardizeX.m** → pre-processing

2 Guidelines for usage

We present the source code in Matlab in order to compute the CLAD estimator exactly (when technically possible) using MIP following (Bilias, Florios, & Skouras, 2013). There are several options on which MIP solver to use in MATLAB. Here are two options - for MATLAB ® versions before R2014a:

- the milp.m solver by Prof. Kevin Tomsovic, for small scale examples ($N \leq 30$), which uses the linprog() command for the LP subproblems solution
- the cplexmilp.m solver by IBM ILOG CPLEX OPTIMIZATION STUDIO 12.6 ® which is formatted in the milp_cplex() function we wrote, and is the state-of-the-art solver for $N \approx 500$

For users of versions after R2014a (including R2014a):

- the intlinprog() solver by MATHWORKS ®, can be used for medium scale problems (N=200-400 observations)

Typical values for p are $p \in \{2,3,4,5,6\}$. Often CPLEX can tackle problems up to $N \approx 500$ and $p \leq 5$. [1].

[1] It is possible to choose the DGPs in such a way that problems with $N \approx 800$ and $p = 5$ can be solved

References

Bilias, Y., Florios, K., & Skouras, S. (2013). *Exact computation of censored least absolute deviations estimators* (Technical Report). Athens University of Economics and Business. (Available at SSRN: http://ssrn.com/abstract=2372588)