ECON 512: Computation project

Chen Zhang

April 27, 2018

1 Introduction

A lot of econometric problems work well in theory, but they may be computational challenging. For example, many models of extremum estimators are known to be difficult to compute due to highly nonconvex criterion functions with many local optima (but well pronounced global optimum), e.g. instruement quantile regression, censored and nonlinear quantile regression. In Chernozhukov and Hong (2003, JoE) they proposed a class of estimators, quasi-Bayesian estimators or Laplace type estimator (LTE), which are defined similar to Bayesian estimators and can be computed by MCMC method. This estimator is computational attractive, because it transforms the optimization problem of extremum estimators to an numerical integration problem, which does not suffer to the problem of nonconvexity of objective function.

2 Laplacian or quasi-Bayesian estimator

This paper takes advantage of LTE to investigate computation problem in censored data. Consider the model

Suppose we have a MLE estimator defined as

$$\hat{\theta}_{MLE} = \arg\sup_{\theta\in\Theta} L_n(\theta),$$

where $L_n(\theta)$ is the log-likelihood function. To implement Bayesian estimation for any prior

$\pi(\theta)$, the posterior is

$$p(\theta|y,x) = e^{L_n(\theta)}\pi(\theta),$$

where $L_n(\theta)$ is the objective function of maximum likelihood estimator, the likelihood function. We can see there is a natural connection between maximum likelihood and Bayesian method.

Now, we consider a general extremum estimator problem

$$\hat{\theta}_{EE} = \arg\sup_{\theta\in\Theta} Q_n(\theta),$$

where $Q_n(\theta)$ can be an objective function of any extremum estimator. If $Q_n(\theta) = \frac{1}{n}L_n(\theta)$ is the objective function of MLE, for any prior $\pi(\theta)$ we can use $e^{nQ_n(\theta)}\pi(\theta)$ as posterior and do Bayesian with no effort. If it is not, the transformation

$$p_n(\theta) = \frac{e^{nQ_\theta}\pi(\theta)}{\int_\Theta e^{nQ_n\pi(\theta)}\ \mathrm{d}\theta},$$

is a proper distribution density and can be used as posterior, called here the *quasi-posterior*. Here $\pi(\theta)$ is a weight function or prior probability density that is strictly positive over Θ. Note that $p_n(\theta)$ is generally not a true posterior in Bayesian sense, since $nQ_n(\theta)$ may not be a likelihood.

The quasi-posterior mean is then defined as

$$\hat{\theta} = \int_\Theta \theta p_n(\theta)\ \mathrm{d}\theta = \int_\Theta \theta \frac{e^{nQ_n}\pi(\theta)}{\int_\Theta e^{nQ_n}\pi(\theta)\ \mathrm{d}\theta}.$$

Follow the spirit of Bayesian, using Markov chain Monte Carlo method, we can draw a Markov chain,

$$S = (\theta^{(1)}, \theta^{(2)}, ..., \theta^{(B)}),$$

whose marginal density is given by $p_n(\theta)$. Then the estimate $\hat{\theta}$ can be computed as

$$\hat{\theta} = \frac{1}{B}\sum_{i=1}^{B}\theta^{(i)}.$$

The confidence interval can be constructed based on the quantile of $S = (\theta^{(1)}, \theta^{(2)}, ..., \theta^{(B)})$.

3 Censored median regression

In this section, I will apply the quasi-Bayesian estimator to a simulated censored median model and compare it with the extensively used iterative linear programming algorithm.

3.1 The model

Consider the model

$$\begin{aligned} Y_1 =& \alpha_1 X\beta_1 + u_1, \\ Y_2 =& \alpha_2 X\beta_2 + u_2, \\ X \sim& N(0, I_3), \\ \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \sim& N(0, \Sigma), \\ & Y^1 \text{ is observed when } Y^1 > Y^2. \end{aligned}$$

We can think of Y_2 as an alternative of Y_1, Y_1 is chosen only if $Y_1 > Y_2$. So the data of Y_1 will be censored at Y_2.

Since the focus of this paper is on the computational aspect of solving censored model, I simplified the model by assuming that there's no endogeneity and it can be easily transformed to a simple censored model by the following steps. First consider the transformation

$$\begin{aligned} Y_3 =& Y_1 - Y_2 \\ =& (\alpha_1 - \alpha_2) + X(\beta_1 - \beta_2) + (u_1 - u_2) \\ =& \theta_0 + X\theta + \varepsilon, \end{aligned}$$

where $\theta = \beta_1 - \beta_2$, $\varepsilon = u_1 - u_2$. Although in this model the censored point Y_2 is not fixed, by assumption we can observe full data of Y_2. Therefore, we can get a consistent estimate β_2 in the first step with no effort. Then plug this consistent estimator $\hat{\beta}_2$ into the above transformation and get the following simple censored model

$$\begin{aligned} Y^* =& \theta_0 + X\theta + \varepsilon, \\ X \sim N(0, I_3), \quad & \varepsilon \sim N(0, X_2^2 I), \quad Y = \max(0, Y^*). \end{aligned}$$

I used $\theta_0 = -6$, $\theta = (3, 3, 3)'$ to generate the data, which generates about 80% censoring.

3.2 Estimation

To estimate the median of the above model, the estimator is based on Powell's objective function $Q_n(\theta) = -\frac{1}{n}\sum_{i=1}^{n}|Y_i - \max(0, \theta_0 + X\theta)|$. The extremum estimator is defined as

$$\hat{\theta}_{EE} = \arg\max_{\theta\in\Theta} Q_n(\theta).$$

The MCMC run is based on the quasi-posterior

$$p_n(\theta) = \frac{e^{nQ_\theta}\pi(\theta)}{\int_\Theta e^{nQ_n\pi(\theta)}\,\mathrm{d}\theta}.$$

4 Computation and issues

Figure.1 is the surface of the objective function, it illustrates the computation difficulty for the optimization problem. We can see the surface of the objective function is featured by many flat area, which makes it hard to search for the maxima. Linear programming is usually been used to solve this maximization problem, I will illustrate some issues in using linear program in next section. Our quasi-bayesian estimator is an alternative to the extremum estimator solved by linear programming.

4.1 quasi-Bayesian method

The MCMC step is based on Metroplis-Hastings random walk algorithm with some modification. The prior $\pi(\theta)$ is chosen uniformly over $[\theta-10, \theta+10]$. The first issue encountered is adjusting the variance of the simulated Markov chain. Because each element of parametrer θ has different scale (in my example, θ_0 is as twice large as $\theta_1, \theta_2, \theta_3$), if the updating is based on the same distribution over all the parameter, it may result in a very large MSE on some component of the estimator.

It is known that if the variance of proposed update is too large, it will have a low acceptance rate; if the variance is too small, it will have a low acceptance rate. I tried to adjust the proposed random walk innovation on each component separately, so that the variance is adjusted according to acceptance rate.

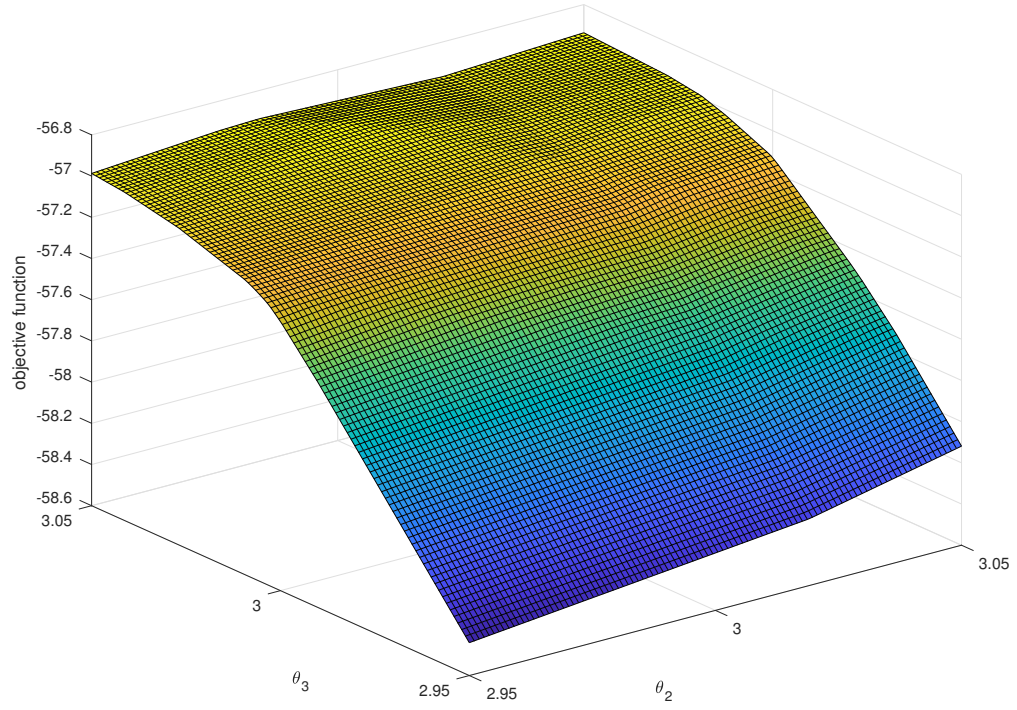

Figure 1: Surface of objective function

1. First, I have to modify Metroplis-Hastings algorithm so that each component updated separately. This step is easy, I modified the updating step to 4 sequential steps, each step only update one component.

2. Then the algorithm added a scale parameter for each component. Every time the updating variance is based on this scale parameter. If the last 100 draws has a low acceptance rate, the scale parameter will adjust, based on the deviation to a specific acceptance rate, to a smaller value and vice versa.

By this modification, the algorithm can adjust the variance automatically so that the acceptance rate is roughly to a specific value. In this paper, the acceptance rate is set to 0.5.

4.2 Iterative Linear Programming

Linear programming is widely used to solve a quantile regression problem. But in my example, the censored quantile problem is not linear because of its censored feature, and thus cannot be formulated as linear program. Buchinsky(1994) has proposed a Iterative Linear Programming algorithm (ILP) to solve censored quantile regression. Although this ILP works in many empirical circumstances, there is no guarantee for the convergence of ILP. It also has a serious problem of converging to a local minimum of zero in our example. ILP is described as following steps:

1. Start with some $\hat{\theta}^{(0)}$ for $j = 0$.

2. Compute $X\hat{\theta}^{(j)}$ and collect the subsample $S_j = \{i : x_i'\hat{\theta}^{(j)} \geq 0$.

3. Use only subsample S_j, run the standard quantile regression with linear programming. With new $\hat{\theta}^{(j+1)}$, compute S_{j+1}.

4. If $S_{j+1} = S_j$, then stop and set estimate to $\hat{\theta}^{(j+1)}$. Otherwise, set $j = j + 1$ and repeat step.3.

In some cases, the ILP doesn't converge. More worse, because its convergent criterion is based on the subsample $S_{j+1} = S_j$, the subsample S_j may update to a very different one if the sample is featured by a large censoring. Therefore, simply setting a maximum iteration $maxit$ and take the estimate $\hat{\beta}^{(maxit)}$ may lead to a local minum that is far away from the global one. In my simulation example, I oberved that although the subsample does not converge but it always alters within a small number of subsets of full sample. For example, with sample size 400, S_j will first converge to some subsample of size 32 or 30 and then altered between these subsamples. Although the estimate does not converge, it ranges within deviation of 1. To fix this problem, I first run a burn in sequence, then average the estimate in a keep sequence. Since not converging only happens in some rare cases and deviation is acceptable, this will not affect the overall performance of the estimator in a large repetition.

Another serious and more common issue of ILP is that it sometimes converge to a local minimum of 0, it will be reported in the result of next section.

5 Result

The result is based on sample size of 400 and 100 repetition.

	RMSE	MAD	Median bias
n = 1600			
QBE-mean	0.307	0.105	-0.006
QBE-median	0.291	0.100	-0.005
ILP (11)	0.227	0.074	0.006
	2.653	0.737	0.005
CH(2003)			
QBE-mean	0.155	0.121	0.009
QBE-median	0.155	0.121	0.002
ILP (7)	0.134	0.106	0.067
	3.547	0.511	-0.384

Table 1: Performance of estimators

The number in the parentheses in the ILP result is the times that ILP converge to a local minimum of 0. The first row for the ILP reports the performance of the algorithm among the subset of simulations for which ILP does not converge to a local minimum at 0. The second row reports athe results for all simulations, including those for which ILP converges to 0. The LTE never converge to 0. We can see, when the local minima are excluded ILP performs slightly better than LTEs. But when the local minima are included in the ILP results, LTEs did much better. We can see LTE is a more robust alternative to ILP in this simulated exercise. Indeed, not only in the censored model, LTE should work more robust in any setting that convergence of computing extremum estimators could be a problem. Figure.2 shows the histograms of simulation with 400 repetitions.

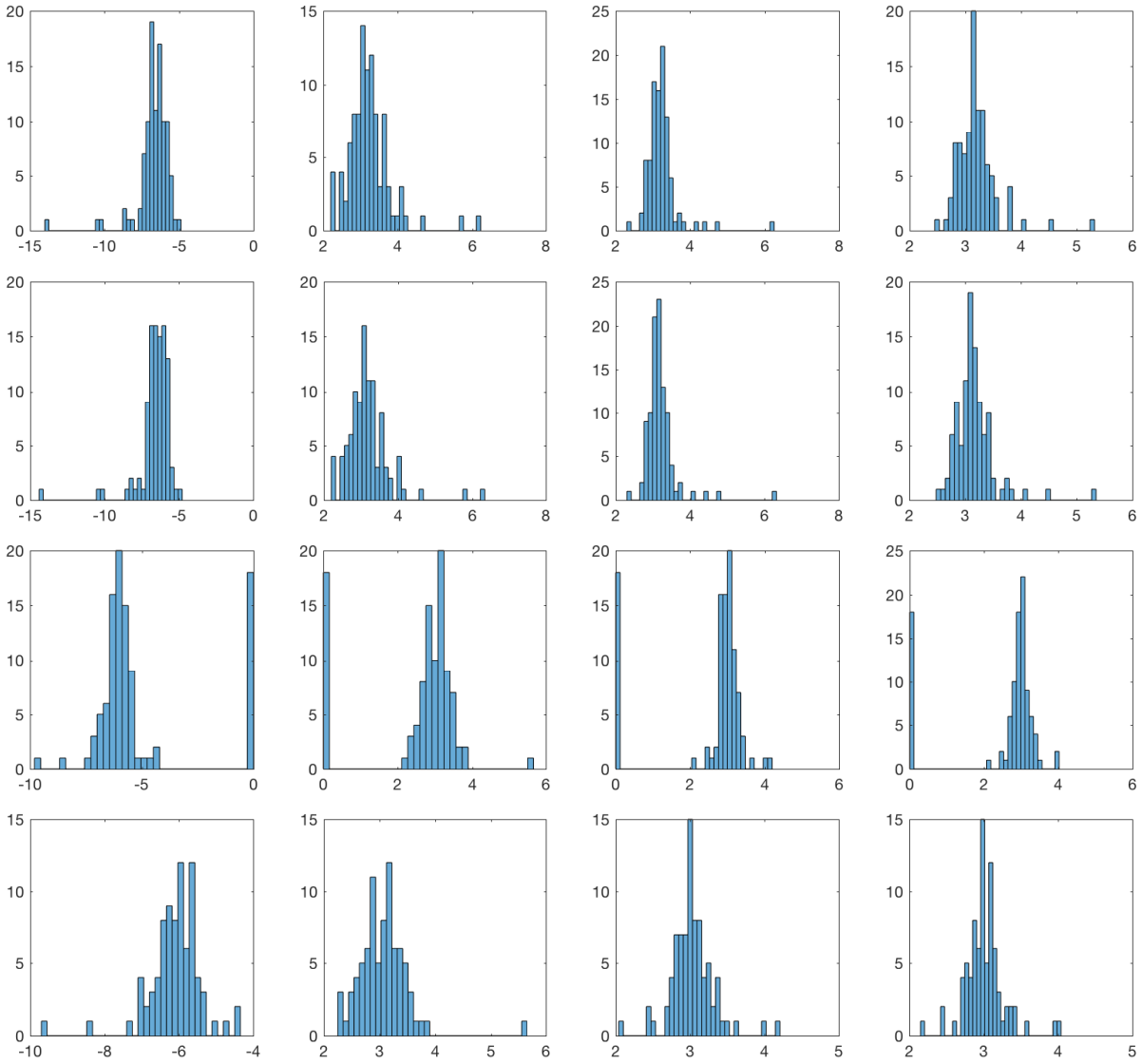

Figure 2: The histograms in each row are for quasi-posterior mean, quasi-posterior median, ILP and ILP exclude local minimum of zero, respectively. Each row contains histograms of four components of the estimator β.

References

Buchinsky, M. (1994). Changes in the us wage structure 1963-1987: Application of quantile regression. *Econometrica: Journal of the Econometric Society*, 405–458.

Chernozhukov, V. and H. Hong (2003). An mcmc approach to classical estimation. *Journal of Econometrics 115*(2), 293–346.

Appendix A. Schematics of Code

Except for main.m, the code contains the following routines and functions,

1. `cqr_ilp.m` is the function that used to compute censored quantile regression by Iterative Linear Programming. Input is (independent var, dependent var, censored point, quantile, initial value, number of burn in iterations, numbers of iterations that are kept). Output is (estimate, keep sequence, whether converge to a local minimum).

2. `dgp.m` is the data generating process function of the simulated model. Input is (sample size, true parameter, random seed).

3. `histo.m` is the subroutine that plots histograms of the estimation results.

4. `mc_chain.m` is the subroutine used to plot the simulated Markov Chain in quasi-Bayesian method. And also plot the surface of objective function.

5. `mcmc_run.m` is the function run the MCMC procedure. Input is (dependent var, independent var, initial var, quantile, length of burn in sequence, length of keep sequence, preset acceptance rate,random seed). Output is (estimate, number of times accepted).

6. `obj.m` is objective function of censored quantile model. Input is (dependent var, independent var, parameter, quantile).

7. `rq.m` is the quantile regression solver provided online by Roger Koenker.

Appendix B. Matlab Code

main.m

```
clear, clc;


beta = [-6;3;3;3];
tau = 0.5;
y_c = 0;
d = length(beta);
n = 1600;
R = 100;
seed = randseed(133,R);


% generate data
y = zeros(n,R);
x = zeros(n,d,R);
for i=1:R
    [y(:,i) x(:,:,i)] = dgp(n,beta,seed(i));
end
% ols initial value
initval = zeros(4,R);
for i=1:R
    initval(:,i) = (x(:,:,i)'*x(:,:,i))\x(:,:,i)'*y(:,i);
end

%% LTE method
% uniform prior
prmax = beta+10;
prmin = beta-10;

% MCMC run
burnin_lte = 10000;
keep_lte = 10000;
accrate = 0.5;
r_lte = 1;
theta_lte = zeros(keep_lte,4,R);
accept = zeros(r_lte,4);
lte_median = zeros(R,4);
lte_mean = zeros(R,4);

while r_lte<=R
    r_lte
```

```
    theta_lte(:,:,r_lte) = mcmc_run(y(:,r_lte),x(:,:,r_lte),...
                                    initval(:,r_lte),tau,
                                       burnin_lte,...
                                    keep_lte,accrate,seed(r_lte
                                       ));
    lte_mean(r_lte,:) = mean(theta_lte(:,:,r_lte));
    lte_median(r_lte,:) = median(theta_lte(:,:,r_lte));
    r_lte = r_lte+1;
end

% accept = mean(iaccept)/(4*keep);
% lte_mean ----------------------------------------------
MSE_lte_mean = var(lte_mean)+(mean(lte_mean)-beta').^2;
RMSE_lte_mean = sqrt(MSE_lte_mean);
MAD_lte_mean = mad(lte_mean);
mbias_lte_mean = median(lte_mean) - beta';
% lte_median --------------------------------------------
MSE_lte_median = var(lte_median)+(mean(lte_median)-beta').^2;
RMSE_lte_median = sqrt(MSE_lte_median);
MAD_lte_median = mad(lte_median);
mbias_lte_median = median(lte_median) - beta';
% -------------------------------------------------------




%% Iterative Linear Programming
r_ilp = 1;
burnin_ilp = 1000;
keep_ilp = 1000;
theta_ilp = zeros(keep_ilp,4,R);
localmin = 0; % count the times localmin been found
out = 0; % count times converge to a localmin of zero

while r_ilp<=R
    r_ilp
    [ret theta_ilp(:,:,r_ilp) lm] = cqr_ilp(x(:,:,r_ilp),y(:,
       r_ilp),...
                                            y_c,tau,initval(:,
                                               r_ilp), ...
                                            burnin_ilp,keep_ilp
                                               );
    if norm(ret)==0
        out = out +1;
    end
```

```
    ilp_est(r_ilp,:) = ret';
    localmin = localmin +lm;
    r_ilp = r_ilp+1;
end
ilp_exzero = ilp_est(ilp_est(:,2)~=0,:);


% ilp estimates ------------------------------------------------
MSE_ilp_all = var(ilp_est)+(mean(ilp_est)-beta').^2;
RMSE_ilp_all = sqrt(MSE_ilp_all);
MAD_ilp_all = mad(ilp_est);
mbias_ilp_all = median(ilp_est) - beta';
% ilp estimates excluded local convergence to 0 -------------
MSE_ilp_exzero = var(ilp_exzero)+(mean(ilp_exzero)-beta').^2;
RMSE_ilp_exzero = sqrt(MSE_ilp_exzero);
MAD_ilp_exzero = mad(ilp_exzero);
mbias_ilp_exzero = median(ilp_exzero) - beta';
% ------------------------------------------------------------

kk = 10:10:10000;
```

cqr_ilp.m

```
function [ret,ilp_simul,locmin] = cqr_ilp(x,y,y_c,tau,initval,
   burnin,keep)

iter_b = 1;
iter_k = 1;
locmin = 0;

M = [y,x];
beta = initval;
ilp_simul = zeros(keep,length(initval));

while iter_b<=burnin
    x_temp = x(x*beta>=y_c,:);
    y_temp = y(x*beta>=y_c);
    M_new = [y_temp, x_temp];
    beta_new = rq(x_temp,y_temp,tau);
    if length(M)==length(M_new)
        if  M~=M_new
            iter_b=iter_b+1;
            beta = beta_new;
            M = M_new;
        else
```

```
            ret = beta;
            locmin = 1;
            return
        end
    elseif length(M)~=length(M_new)
        iter_b=iter_b+1;
        beta = beta_new;
        M = M_new;
    else
        ret = beta;
        locmin = 1;
        return;
    end

end

beta_keep = zeros(length(initval),keep);
beta_keep(:,1) = beta;
while iter_k<=keep
    x_temp = x(x*beta_keep(:,iter_k)>=y_c,:);
    y_temp = y(x*beta_keep(:,iter_k)>=y_c);
    M_new = [y_temp, x_temp];
    beta_new = rq(x_temp,y_temp,tau);
    if length(M)==length(M_new)
        if  M~=M_new
            beta_keep(:,iter_k) = beta_new;
            iter_k=iter_k+1;
            M = M_new;
        else
            ret = beta_keep(:,iter_k);
            locmin = 1;
            return
        end
    elseif length(M)~=length(M_new)
        beta_keep(:,iter_k) = beta_new;
        iter_k=iter_k+1;
        M = M_new;
    else
        ret = beta_keep(:,iter_k);
        locmin = 1;
        return;
    end
end
ret = beta_new;
ilp_simul = beta_keep';
```

```
end
```

dgp.m

```
function [y x] = dgp(n,beta,seed)

rng(seed);

x = randn(n,3);
x = [ones(n,1) x];
u = randn(n,1);
u = x(:,2).^2.*u;

y_star = x*beta +u;

y = max(0,y_star);
```

histo.m

```
edge = [linspace(beta(1)-1.5,beta(1)+1.5,25); ...
    linspace(beta(2)-0.75,beta(2)+0.75,25); ...
    linspace(beta(3)-0.75,beta(3)+0.75,25); ...
    linspace(beta(4)-0.75,beta(4)+0.75,25)];
nbins = 20;

% MCMC histogram
f1 = figure('Name','Histogram LTE-mean');
for i=1:4
    subplot(1,4,i);
    histogram(lte_mean(:,i),edge(i,:),'Normalization','
       probability');
end
f2 = figure('Name','Histogram LTE-median');
for i=1:4
    subplot(1,4,i);
    histogram(lte_median(:,i),edge(i,:),'Normalization','
       probability');
end

% ILP histogram
ed = [linspace(-7.5,0,38); ...
    linspace(0,3.75,38); ...
    linspace(0,3.75,38); ...
    linspace(0,3.75,38)];
f3 = figure('Name','ILP included zeros');
```

```
for i=1:4
    subplot(1,4,i);
    histogram(ilp_est(:,i),ed(i,:),'Normalization','probability
       ');
end
f4 = figure('Name','ILP excluded zeros');
for i=1:4
    subplot(1,4,i);
    histogram(ilp_exzero(:,i),edge(i,:),'Normalization','
       probability');
end
```

mc_chain.m

```
%% MC chain
hold on
plot(theta_lte(:,4,50))
% for i=1:4
%     subplot(4,1,i)
%     plot(theta_lte((end-999):end,i,50));
% end

figure
[b1,b2] = meshgrid(linspace(2.95,3.05,100),linspace
   (2.95,3.05,100));

simpost = zeros(100,100);
for i = 1:length(b1)
    for j = 1:length(b2)
        bb = [-6;b1(i,j);b2(i,j);3];
        simpost(i,j) = obj(y(:,50),x(:,:,50),bb,0.5);
    end;
end;
surf(b1,b2,simpost)
xlabel('\theta_2')
ylabel('\theta_3')
zlabel('objective function')
```

mcmc_run.m

```
function [theta accept] = mcmc_run(y,x,initval,tau,burnin,keep,
   acc_rate,seed)

rng(seed);
theta = zeros(keep,4);
ehat = y - x*initval;
```

```
n = length(y);
promu = zeros(4,1);
prosig = ones(4,1);
scale = ones(4,1);
% cov = scale*ehat_sq^2*(x'*x)^(-1);
% cov_chol = chol(cov)';

% post = @(t) obj(y,x,t); %+ sum(log(unifpdf(t,prmin,prmax)));
curr_pi = obj(y,x,initval,tau); % posterior kernel Ln(theta)
curr_theta = initval;
propose = curr_theta;
b = 2;
accept = zeros(4,1);
adjust = accept;

%% MCMC run

while b<= burnin
    if mod(b,200)==0
        for i=1:4
            scale(i) = scale(i)*(1+(accept(i)-adjust(i))/100-
               acc_rate);
            adjust(i) = accept(i);
        end
    end
    e = normrnd(promu,prosig);
    e = scale.*e;

    for i = 1:4
        propose(i) = curr_theta(i) + e(i);
        prop_pi = obj(y,x,propose,tau);
        if prop_pi == NaN;
            prop_pi = -1000000;
        end
        u = unifrnd(0,1);
        u = log(u);
        if u < prop_pi - curr_pi;
            curr_theta = propose;
            curr_pi = prop_pi;
            accept(i) = accept(i)+1;
        end
    end

    b = b+1;
end
```

```
50
51 theta(1,:) = curr_theta';
52 k = 2;
53 while k<= keep
54     if mod(k,200)==0
55         scale(i) = scale(i)*(1+(accept(i)-adjust(i))/100-
              acc_rate);
56         adjust(i) = accept(i);
57     end
58     e = normrnd(promu,prosig);
59     e = scale(i)*e;
60     for i=1:4
61         propose(i) = curr_theta(i)' + e(i);
62         prop_pi = obj(y,x,propose,tau);
63         if prop_pi == NaN;
64             prop_pi = -999999;
65         end
66         u = unifrnd(0,1);
67         u = log(u);
68         if u < prop_pi - curr_pi;
69             accept(i) = accept(i) + 1;
70             curr_theta = propose;
71             curr_pi = prop_pi;
72         end
73     end
74     theta(k,:) = curr_theta';
75
76     k = k+1;
77 end
78 accept = accept';
79
80 end
```

obj.m

```
1 function L = obj(y,x,b,tau)
2
3 n = length(y);
4 % L= -1/n*sum((tau-(y<=max(0,x*b))).*(y-max(0,x*b))); % compute
      tau quantile
5 check = abs(y-max(0,x*b));
6 L= -sum(check); % LAD
7
8
9 end
```

rq.m

```
% Programmed by Roger Koenker.

% input: X is an n x k matrix of exogenous regressors,
%        y is an n x 1 vector of outcome variables
%        p \in (0,1) is the quantile of interest
% Output: p^{th} regression quantiles.




function b = rq_fnm(X, y, p)
% Construct the dual problem of quantile regression
% Solve it with lp_fnm
%
%
[m n] = size(X);
u = ones(m, 1);
a = (1 - p) .* u;
b = -lp_fnm(X', -y', X' * a, u, a)';

function y = lp_fnm(A, c, b, u, x)
% Solve a linear program by the interior point method:
% min(c * u), s.t. A * x = b and 0 < x < u
% An initial feasible solution has to be provided as x
%
% Function lp_fnm of Daniel Morillo & Roger Koenker
% Translated from Ox to Matlab by Paul Eilers 1999
% Modified by Roger Koenker 2000--
% More changes by Paul Eilers 2004


% Set some constants
beta = 0.9995;
small = 1e-5;
max_it = 50;
[m n] = size(A);

% Generate inital feasible point
s = u - x;
y = (A' \  c')';
r = c - y * A;
r = r + 0.001 * (r == 0);     % PE 2004
z = r .* (r > 0);
```

```
w = z - r;
gap = c * x - y * b + w * u;

% Start iterations
it = 0;
while (gap) > small & it < max_it
    it = it + 1;

    %   Compute affine step
    q = 1 ./ (z' ./ x + w' ./ s);
    r = z - w;
    Q = spdiags(sqrt(q), 0, n, n);
    AQ = A * Q;           % PE 2004
    rhs = Q * r';         % "
    dy = (AQ' \ rhs)';    % "
    dx = q .* (dy * A - r)';
    ds = -dx;
    dz = -z .* (1 + dx ./ x)';
    dw = -w .* (1 + ds ./ s)';

    % Compute maximum allowable step lengths
    fx = bound(x, dx);
    fs = bound(s, ds);
    fw = bound(w, dw);
    fz = bound(z, dz);
    fp = min(fx, fs);
    fd = min(fw, fz);
    fp = min(min(beta * fp), 1);
    fd = min(min(beta * fd), 1);

    % If full step is feasible, take it. Otherwise modify it
    if min(fp, fd) < 1

        % Update mu
        mu = z * x + w * s;
        g = (z + fd * dz) * (x + fp * dx) + (w + fd * dw) * (s
           + fp * ds);
        mu = mu * (g / mu) ^3 / ( 2* n);

        % Compute modified step
        dxdz = dx .* dz';
        dsdw = ds .* dw';
        xinv = 1 ./ x;
        sinv = 1 ./ s;
        xi = mu * (xinv - sinv);
```

```
        rhs = rhs + Q * (dxdz - dsdw - xi);
        dy = (AQ' \ rhs)';
        dx = q .* (A' * dy' + xi - r' -dxdz + dsdw);
        ds = -dx;
        dz = mu * xinv' - z - xinv' .* z .* dx' - dxdz';
        dw = mu * sinv' - w - sinv' .* w .* ds' - dsdw';

        % Compute maximum allowable step lengths
        fx = bound(x, dx);
        fs = bound(s, ds);
        fw = bound(w, dw);
        fz = bound(z, dz);
        fp = min(fx, fs);
        fd = min(fw, fz);
        fp = min(min(beta * fp), 1);
        fd = min(min(beta * fd), 1);

    end

    % Take the step
    x = x + fp * dx;
    s = s + fp * ds;
    y = y + fd * dy;
    w = w + fd * dw;
    z = z + fd * dz;
    gap = c * x - y * b + w * u;
    %disp(gap);
end

function b = bound(x, dx)
% Fill vector with allowed step lengths
% Support function for lp_fnm
b = 1e20 + 0 * x;
f = find(dx < 0);
b(f) = -x(f) ./ dx(f);
```