% ========================================================================

%

% Method of moments estimation of a first order MA model.

%

% ========================================================================

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',123) );

t = 250; % Sample size

theta = 0.5; % MA1 Population parameter

% Generate the data for a MA(1) model

u = randn(t,1);

y = u(2:end) - theta\*u(1:end-1);

% Estimate the first order autocorrelation coefficient

y = y - mean(y);

rho = y(2:end)\y(1:end-1);

% Estimate theta using the method of moments estimator

b\_mom = ( -1 + sqrt(1 - 4\*rho^2) ) / (2\*rho);

disp(' ')

disp(['Sample size = ', num2str(t) ]);

disp(['True population parameter (theta) = ', num2str(theta) ]);

disp(['Method of moment estimate = ', num2str(b\_mom) ]);

disp(['True population parameter (AR(1)) = ', num2str(-theta/(1+theta^2)) ]);

disp(['First order AR(1) = ', num2str(rho) ]);

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%\*\*

%\*\* Generate graph

%\*\*

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Switch off TeX interpreter and clear figure

set(0,'defaulttextinterpreter','none');

figure(1);

clf;

plot(1:1:249,y,'-k');

xlabel('$t$');

ylabel('$y\_t$');

axis tight;

box off;

set(gca,'XTick',0:50:250);

set(gca,'YTick',-4:2:4);

xlim([0,250]);

ylim([-4,4]);

%laprint(1,'ma1','options','factory');

% ========================================================================

%

% Estiamte AR(1) coefficient by simulation of a MA(1) model

%

% ========================================================================

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',123) );

theta = 0.5;

t = 250;

h = 1;

n = t\*h;

% Simulate the data for a MA(1) model

u = randn(n,1);

ys = trimr(u,1,0) - theta\*trimr(u,0,1);

% Estimate the first order autocorrelation coefficient

ys = ys - mean(ys);

rhos = trimr(ys,0,1)\trimr(ys,1,0);

disp(' ')

disp(['Sample size = ', num2str(t) ]);

disp(['h = ', num2str(h) ]);

disp(['True population parameter (theta) = ', num2str(theta) ]);

disp(' ')

disp(['True population parameter (rho) = ', num2str(-theta/(1+theta^2)) ]);

disp(['Simulation estimate (rho) = ', num2str(rhos) ]);

%=========================================================================

%

% Monte Carlo analysis to investigate the sampling properties

% of the indirect estimator of a first order MA model.

%

% Gourieroux et. al. (1993) J of Appl. Eco.

%

%=========================================================================

function sim\_ma1indirect( )

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',123) );

% Parameters of MA(1) process

t = 250;

theta = 0.5;

lag = 3;

% Simulation settings

opt = optimset('LargeScale','off','Display','off');

nreps = 1000;

b = zeros(nreps,1);

% Main DO LOOP to generate sampling disribution

for j = 1:nreps

% Generate the actual data for the MA(1) process

u = randn(t,1);

y = trimr(u,1,0) - theta\*trimr(u,0,1);

% Estimate the the auxiliary model using actual data

y = y - mean(y);

if lag == 1

bhat = (trimr(y,0,1))\trimr(y,1,0);

elseif lag == 2

bhat = [trimr(y,1,1),trimr(y,0,2)]\trimr(y,2,0);

elseif lag == 3

bhat = [trimr(y,2,1),trimr(y,1,2),trimr(y,0,3)]\trimr(y,3,0);

end

% Compute indirect estimator (could use a line search algorithm)

e = randn(t,1);

b(j) = fminsearch(@(b) q(b,e,lag,bhat),0.5,opt);

end

% Generate statistics on the sampling distribution

b = tanh(b);

m = mean(b);

stdev = std(b);

rmse = sqrt(mean(b-theta)^2);

disp(' ');

disp(['Number of replications = ', num2str(nreps) ]);

disp(['Sample size = ', num2str(t) ]);

disp(' True Mean Std.err. RMSE ')

disp([theta m stdev rmse]);

end

%

%------------------------- Functions -------------------------------------%

%

%-------------------------------------------------------------------------%

% Objective function to compute the indirect estimator

%-------------------------------------------------------------------------%

function val = q(b,e,lag,bhat)

% Simulate data making sure that b(1) is in the unit circle

ys = trimr(e,1,0) - tanh(b)\*trimr(e,0,1);

ys = ys - mean(ys);

if lag == 1

bhats = (trimr(ys,0,1))\trimr(ys,1,0);

elseif lag == 2

bhats = [trimr(ys,1,1),trimr(ys,0,2)]\trimr(ys,2,0);

elseif lag == 3

bhats = [trimr(ys,2,1),trimr(ys,1,2),trimr(ys,0,3)]\trimr(ys,3,0);

end

val = (bhat - bhats)'\*(bhat - bhats);

end

%=========================================================================

%

% Monte Carlo analysis to investigate the sampling properties

% of the EMM estimator of a MA(1) model.

%

% Gourieroux et. al. (1993) J of Appl. Eco.

%

%=========================================================================

function sim\_ma1emm( )

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',123) );

% Parameters of MA(1) process

t = 250; % Sample size

theta = 0.5; % MA1 Population parameter

lag = 1; % Choose AR lag for auxiliary model

p = 0; % Used to construct weighting matrix

% Simulation settings

opt = optimset('LargeScale','off','Display','off');

nreps = 1000;

b = zeros(nreps,1);

% Main DO LOOP to generate sampling disribution

for j = 1:nreps

% Generate the actual data for the MA(1) process

u = randn(t,1);

y = trimr(u,1,0) - theta\*trimr(u,0,1);

% Estimate the the auxiliary model using actual data

y = y - mean(y);

if lag == 1

x = trimr(y,0,1);

bhat = x\trimr(y,1,0);

ehat = trimr(y,1,0) - x\*bhat;

g = ehat.\*x;

elseif lag == 2

x = [trimr(y,1,1) , trimr(y,0,2)];

bhat = x\trimr(y,2,0);

ehat = trimr(y,2,0) - x\*bhat;

g = repmat(ehat,1,2).\*x;

elseif lag == 3

x = [trimr(y,2,1) , trimr(y,1,2) , trimr(y,0,3)];

bhat = x\trimr(y,3,0);

ehat = trimr(y,3,0) - x\*bhat;

g = repmat(ehat,1,3).\*x;

end

% Compute the optimal weighting matrix

i = g'\*g;

l = 1;

while l <= p

gam = g((l+1):size(g,1),:)'\*g(1:(size(g,1)-l),:);

i = i + (1.0 - l/(p+1))\*(gam + gam');

l = l + 1;

end

i = i/length(g);

iinv = inv(i);

% Compute EMM estimator (could use a line search algorithm)

e = randn(t,1);

b(j) = fminsearch(@(b) q(b,e,lag,bhat,iinv),0.5,opt);

end

% Generate statistics on the sampling distribution

b = tanh(b);

m = mean(b);

stdev = std(b);

rmse = sqrt(mean(b-theta)^2);

disp(' ');

disp(['Number of replications = ', num2str(nreps) ]);

disp(['Sample size = ', num2str(t) ]);

disp(' True Mean Std.err. RMSE ')

disp([theta m stdev rmse]);

end

%

%------------------------- Functions -------------------------------------%

%

%-------------------------------------------------------------------------%

% Objective function to compute the EMM estimator

%-------------------------------------------------------------------------%

function retp = q(b,e,lag,bhat,iinv)

ys = trimr(e,1,0) - tanh(b)\*trimr(e,0,1);

ys = ys - mean(ys);

if lag == 1

xs = trimr(ys,0,1);

ehats = trimr(ys,1,0) - xs\*bhat;

gs = mean( ehats.\*xs )';

elseif lag == 2

xs = [trimr(ys,1,1) , trimr(ys,0,2)];

ehats = trimr(ys,2,0) - xs\*bhat;

gs = mean( repmat(ehats,1,2).\*xs )';

elseif lag == 3

xs = [trimr(ys,2,1) , trimr(ys,1,2) , trimr(ys,0,3)];

ehats = trimr(ys,3,0) - xs\*bhat;

gs = mean( repmat(ehats,1,3).\*xs )';

end

retp = gs'\*iinv\*gs ;

end

%=========================================================================

%

% Over-identification test of the MA(1) model using the EMM estimates

%

%=========================================================================

function sim\_ma1overid( )

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',12345) );

% Parameters of MA(1) process

t = 250; % Sample size

theta = 0.5; % MA1 Population parameter

lag = 3; % Choose AR lag for auxiliary model

p = 0; % Used to construct weighting matrix

% Simulation settings

opt = optimset('LargeScale','off','Display','off');

% Generate the actual data for the MA(1) process

u = randn(t,1);

y = trimr(u,1,0) - theta\*trimr(u,0,1);

% Estimate the the auxiliary model using actual data

y = y - mean(y);

if lag == 1

x = trimr(y,0,1);

bhat = x\trimr(y,1,0);

ehat = trimr(y,1,0) - x\*bhat;

g = ehat.\*x;

elseif lag == 2

x = [trimr(y,1,1) , trimr(y,0,2)];

bhat = x\trimr(y,2,0);

ehat = trimr(y,2,0) - x\*bhat;

g = repmat(ehat,1,2).\*x;

elseif lag == 3

x = [trimr(y,2,1) , trimr(y,1,2) , trimr(y,0,3)];

bhat = x\trimr(y,3,0);

ehat = trimr(y,3,0) - x\*bhat;

g = repmat(ehat,1,3).\*x;

end

% Compute the optimal weighting matrix

i = g'\*g;

if p > 0

for l = 1:p

gam = g((l+1):size(g,1),:)'\*g(1:(size(g,1)-l),:);

i = i + (1.0 - l/(p+1))\*(gam + gam');

end

end

i = i/length(g);

iinv = inv(i);

% Compute EMM estimator (could use a line search algorithm)

e = randn(t,1);

[b,qmin] = fminsearch(@(b) q(b,e,lag,bhat,iinv),0.5,opt);

% Number of restrictions

r = length(bhat) - length(b);

disp(' ');

disp(['Sample size = ', num2str(t) ]);

disp(['True parameter value = ', num2str(theta) ]);

disp(['Lags in auxiliary model = ', num2str(lag) ]);

disp(['Over-identification test = ', num2str(t\*qmin) ]);

disp(['Number of restrictions = ', num2str(r) ]);

if r > 0

disp(['p value = ', num2str(1-cdf('chi2',t\*qmin,r)) ]);

end

end

%

%------------------------- Functions -------------------------------------%

%

%-------------------------------------------------------------------------%

% Objective function to compute the EMM estimator

%-------------------------------------------------------------------------%

function retp = q(b,e,lag,bhat,iinv)

ys = trimr(e,1,0) - tanh(b)\*trimr(e,0,1);

ys = ys - mean(ys);

if lag == 1

xs = trimr(ys,0,1);

ehats = trimr(ys,1,0) - xs\*bhat;

gs = mean( ehats.\*xs )';

elseif lag == 2

xs = [trimr(ys,1,1) , trimr(ys,0,2)];

ehats = trimr(ys,2,0) - xs\*bhat;

gs = mean( repmat(ehats,1,2).\*xs )';

elseif lag == 3

xs = [trimr(ys,2,1) , trimr(ys,1,2) , trimr(ys,0,3)];

ehats = trimr(ys,3,0) - xs\*bhat;

gs = mean( repmat(ehats,1,3).\*xs )';

end

retp = gs'\*iinv\*gs ;

end

% ========================================================================

%

% Simulate Brownian motion and compare continuous and discrete data

%

% ========================================================================

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',12345) );

dt = 1/60; % Minute data

n = 240; % 10 days = 240hrs

t = n/dt; % Sample size

% Parameters

mu = 0.0; % Mean

sig2 = 1.0; % Variance

% Wiener process

dw = sqrt(dt)\*randn(t,1);

% Generate continous data (minutes)

y(1) = 0.0;

for i = 1:t-1

y(i+1)=y(i)+mu\*dt + sqrt(sig2)\*dw(i);

end

y10 = y(10:10:t); % Generate 10 minute data by choosing every 10th observation

yhr = y(10:60:t); % Generate hourly data by choosing every 60th observation

ydy = y(1440:1440:t); % Generate daily data by choosing every 1440th observation \*\*/

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%\*\*

%\*\* Generate graph

%\*\*

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Switch off TeX interpreter and clear figure

set(0,'defaulttextinterpreter','none');

figure(1);

clf;

%--------------------------------------------------------%

% Panel (a)

ind = (1:1:t)/1440;

subplot(2,2,1);

plot(ind,y,'-k');

title('(a) Minute Data');

ylabel('$y\_t$');

xlabel('$t$ days');

set(gca,'XTick',0:1:10);

set(gca,'YTick',-10:10:30);

ylim([-10,30]);

xlim([0,10]);

box off;

%--------------------------------------------------------%

% Panel (b)

subplot(2,2,2);

ind = (10:10:t)/1440;

plot(ind,y10,'-k')

title('(b) Ten Minute Data');

ylabel('$y\_t$');

xlabel('$t$ days');

set(gca,'XTick',0:1:10);

set(gca,'YTick',-10:10:30);

ylim([-10,30]);

xlim([0,10]);

box off;

%--------------------------------------------------------%

% Panel (c)

ind = (10:60:t)/1440;

subplot(2,2,3);

plot(ind,yhr,'-k')

title('(c) Hourly Data');

ylabel('$y\_t$');

xlabel('$t$ days');

set(gca,'XTick',0:1:10);

set(gca,'YTick',-10:10:30);

ylim([-10,30]);

xlim([0,10]);

box off;

%--------------------------------------------------------%

% Panel (d)

ind = (1440:1440:t)/1440;

subplot(2,2,4);

plot(ind,ydy,'-k')

title('(c) Daily Data');

ylabel('$y\_t$');

xlabel('$t$ days');

set(gca,'XTick',0:1:10);

set(gca,'YTick',-10:10:30);

ylim([-10,30]);

xlim([0,10]);

box off;

%laprint(1,'brown','options','factory');

% ========================================================================

%

% This program performs a Monte Carlo analysis to investigate the

% sampling properties of the Indirect estimator of Brownian motion with drift.

%

% Gourieroux et. al. (1993) J of Appl. Eco.

% ========================================================================

function sim\_brownind( )

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',12345) );

t = 500; % Sample size

mu = 0.5;

sig2 = 0.25;

theta = [mu;sig2]; % Parameters of Brownian motion

y0 = 1; % Initial value of Brownian process

dt = 0.1; % Continuous time step interval

h = 1/dt; % Scalar to control the simulation run

n = t\*h; % Length of the simulated series

% Main DO LOOP to generate sampling disribution

ndraws = 100;

theta\_ind = zeros(ndraws,2);

for j = 1:ndraws

disp(['Replication number ', num2str(j) ]);

% Simulate sample data using an exact discretisation

u = randn(t,1);

y = recserar( mu + sqrt(sig2)\*u , y0 , 1.0 );

% Estimate the auxiliary model using actual data

dy = trimr(y,1,0) - trimr(y,0,1);

bhat = [mean(dy) ; mean((dy - mean(dy)).^2)];

% Generate errors to be used to compute the emm estimator.

% Note that these errors need to be generated outside of the procedure.

v = sqrt(dt)\*randn(n,1);

% Estimate model

bstart = [mu ; sig2];

options = optimset('LargeScale','off','Display','off');

b = fminunc(@(b) q(b,dt,v,y0,bhat,n),bstart,options);

theta\_ind(j,:) = b';

end

% Generate statistics on the sampling distribution

mean\_ = mean(theta\_ind);

stdev = sqrt(mean((theta\_ind - repmat(mean\_,size(theta\_ind,1),1)).^2));

rmse = sqrt(mean((theta\_ind - repmat(theta',size(theta\_ind,1),1)).^2));

disp(' ');

disp(['Number of replications = ', num2str(ndraws) ]);

disp(['Sample size = ', num2str(t) ]);

disp(' ');

disp(['True population parameter = ', num2str(theta')]);

disp(['Mean of estimates = ', num2str(mean\_) ]);

disp(['Standard deviation of estimates = ', num2str(stdev) ]);

disp(['RMSE of Theta = ', num2str(rmse) ]);

end

%

% ------------------------ Functions ------------------------------------%

%

%-------------------------------------------------------------------------

% The objective function to compute the indirect estimator

%-------------------------------------------------------------------------

function val = q(b,dt,v,y0,bhat,n)

b(2)= abs(b(2));

% Generate continuous time data and choose every t-th observation

ys = recserar( b(1)\*dt + sqrt(b(2))\*v , y0 , 1 );

nys = zeros(n\*dt,1);

for i = 1:n\*dt

temp = i\*(1/dt);

nys(i) = ys(temp);

end

ys = nys;

dys = trimr(ys,1,0) - trimr(ys,0,1);

bhats = [mean(dys) ; mean((dys - mean(dys)).^2)];

val = (bhat - bhats)'\*(bhat - bhats) ;

end

% ========================================================================

%

% This program performs a Monte Carlo analysis to investigate the

% sampling properties of the EMM estimator of Brownian motion with drift.

%

% Gourieroux et. al. (1993) J of Appl. Eco.

%

% ========================================================================

function sim\_brownemm( )

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',12345) );

t = 500; % Sample size

mu = 0.5;

sig2 = 0.25;

theta = [mu;sig2]; % Parameters of Brownian motion

y0 = 1; % Initial value of Brownian process

dt = 0.1; % Continuous time step interval

h = 1/dt; % Scalar to control the simulation run

n = t\*h; % Length of the simulated series

% Main DO LOOP to generate sampling disribution

ndraws = 100;

theta\_emm = zeros(ndraws,2);

for j = 1:ndraws

disp(['Replication number ', num2str(j) ]);

% Simulate Brownian motion using an exact discretisation

u = randn(t,1);

y = recserar( mu + sqrt(sig2)\*u , y0 , 1.0 );

% Estimate the auxiliary model using actual data

dy = trimr(y,1,0) - trimr(y,0,1);

bhat = [mean(dy) ; mean((dy - mean(dy)).^2)];

g1 = (dy - bhat(1))/bhat(2);

g2 = ((dy - bhat(1)).^2/bhat(2) - 1)\*0.5/bhat(2);

g = [g1 , g2];

% Compute the optimal weighting matrix

i = g'\*g;

p = size(g,1)-1;

l = 1;

while l < p

gam = g((l+1):size(g,1),:)'\*g(1:(size(g,1)-l),:);

i = i + (1.0 - l/(p+1))\*(gam + gam');

l = l + 1;

end

i = i/size(g,1);

iinv = inv(i);

% Generate errors to be used to compute the emm estimator.

% Note that these errors need to be generated outside of the procedure.

v = sqrt(dt)\*randn(n,1);

% Estimate model

bstart = [mu ; sig2];

options = optimset('LargeScale','off','Display','off');

b = fminunc(@(b) q(b,dt,v,y0,bhat,n,iinv),bstart,options);

theta\_emm(j,:) = b';

end

% Generate statistics on the sampling distribution

mean\_ = mean(theta\_emm);

stdev = sqrt(mean((theta\_emm - repmat(mean\_,size(theta\_emm,1),1)).^2));

rmse = sqrt(mean((theta\_emm - repmat(theta',size(theta\_emm,1),1)).^2));

disp(' ');

disp(['Number of replications = ', num2str(ndraws) ]);

disp(['Sample size = ', num2str(t) ]);

disp(' ');

disp(['True population parameter = ', num2str(theta')]);

disp(['Mean of estimates = ', num2str(mean\_) ]);

disp(['Standard deviation of estimates = ', num2str(stdev) ]);

disp(['RMSE of Theta = ', num2str(rmse) ]);

end

%

% ------------------------ Functions ------------------------------------%

%

%-------------------------------------------------------------------------

% The objective function to compute the emm estimator

%-------------------------------------------------------------------------

function val = q(b,dt,v,y0,bhat,n,iinv)

b(2)= abs(b(2));

% Generate continuous time data and choose every t-th observation

ys = recserar( b(1)\*dt + sqrt(b(2))\*v , y0 , 1 );

nys = zeros(n\*dt,1);

for i = 1:n\*dt

temp= i\*(1/dt);

nys(i) = ys(temp);

end

dys = trimr(nys,1,0) - trimr(nys,0,1);

g1s = (dys - bhat(1))/bhat(2);

g2s = ((dys - bhat(1)).^2/bhat(2) - 1)\*0.5/bhat(2);

gs = mean([ g1s , g2s])';

val = gs'\*iinv\*gs ;

end

% ========================================================================

%

% Monte Carlo analysis to investigate the sampling properties

% of the indirect estimator of geometric Brownian motion.

%

% Gourieroux et. al. (1993) J of Appl. Eco.

% ========================================================================

function sim\_geobrind( )

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',12345) );

t = 100; % Sample size

h = 1;

mu = 1.0;

sig = 0.5;

theta = [mu;sig];

y0 = 1;

dt = 0.01;

% Main DO LOOP to generate sampling disribution

ndraws = 100;

bmsm = zeros(ndraws,2);

for j = 1:ndraws

disp(['Replication number ', num2str(j) ]);

% Generate the actual data

yact = exp(recserar(mu - 0.5\*sig^2 + sig\*randn(t,1),log(y0),1.0));

% Estimate the auxiliary model using actual data

bhat = zeros(2,1);

bhat(1) = mean(trimr(yact,1,0)./trimr(yact,0,1)) - 1;

bhat(2) = sqrt(mean((trimr(yact,1,0)./trimr(yact,0,1) - 1 - bhat(1)).^2));

% Generate errors to simulate the true model

dw = randn(t/dt,h)\*sqrt(dt);

% Estimate the indirect model using simulated data

b0 = [mu;sig];

opt = optimset('LargeScale','off','Display','off');

b = fminunc(@(b) fobj(b,bhat,h,dt,dw,y0,t),b0,opt);

bmsm(j,:) = b';

end

% Generate statistics on the sampling distribution

m = mean(bmsm);

stdev = sqrt(mean((bmsm - repmat(m,size(bmsm,1),1)).^2));

rmse = sqrt(mean((bmsm - repmat(theta',size(bmsm,1),1)).^2));

disp(' ');

disp(['Number of replications = ', num2str(ndraws) ]);

disp(['Sample size = ', num2str(t) ]);

disp(' ');

disp(['True population parameter = ', num2str(theta')]);

disp(['Mean of estimates = ', num2str(m) ]);

disp(['Standard deviation of estimates = ', num2str(stdev) ]);

disp(['RMSE of Theta = ', num2str(rmse) ]);

end

%

% ------------------------ Functions ------------------------------------%

%

%-------------------------------------------------------------------------

% The objective function to compute the indirect estimator

%-------------------------------------------------------------------------

function retp = fobj(b,bhat,h,dt,dw,y0,t)

btilda = zeros(size(bhat,1),h);

ysim = exp(recserar( (b(1) - 0.5\*b(2)^2)\*dt + b(2)\*dw,log(y0)\*ones(1,h),ones(1,h)) );

% Generate discrete time data by choosing every t-th observation

nys = zeros(t,1);

for i = 1:t

temp = i\*(1/dt);

nys(i) = ysim(temp);

end

yt = nys;

btilda(1,:) = (mean(trimr(yt,1,0)./trimr(yt,0,1)) - 1)';

btilda(2,:) = sqrt(mean((trimr(yt,1,0)./trimr(yt,0,1) - 1 - btilda(1,:)).^2))';

w = eye(size(bhat,1));

retp = (bhat-mean(btilda')')'\*w\*(bhat - mean(btilda')');

end

%=========================================================================

%

% Monte Carlo analysis to investigate the sampling properties

% of the indirect estimator of Ornstein-Uhlenbech process.

%

% Gourieroux et. al. (1993) J of Appl. Eco.

%

%=========================================================================

function sim\_ouind( )

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',123) );

% Parameters of Ornstein-Uhlenbech process

t = 250;

kappa = 0.8;

alpha = 0.1;

sig2 = 0.06^2;

y0 = 0.1; % Initial value of Ornstein-Uhlenbech process

dt = 0.1; % Continuous time step interval

h = 10/dt; % Scalar to control the simulation run

n = t\*h; % Length of the simulated series

% Simulation settings

opt = optimset('LargeScale','off','Display','off');

nreps = 200;

theta = zeros(nreps,3);

% Main DO LOOP to generate sampling disribution

for j = 1:nreps

disp(['Replication number ', num2str(j) ]);

% Generate the actual data for the Ornstein-Uhlenbech process

u = randn(t,1);

y = recserar(alpha\*(1-exp(-kappa)) + ...

sqrt(sig2)\*sqrt((1-exp(-2\*kappa))/(2\*kappa))\*u,y0,exp(-kappa));

% Estimate the auxiliary model using actual data

x = [ones(size(y,1)-1,1),trimr(y,0,1)];

y = trimr(y,1,0);

b = x\y;

bhat = zeros(3,1);

bhat(1) = b(1)/(1 - b(2));

bhat(2) = 1 - b(2);

bhat(3) = mean((y - x\*b).^2);

% Compute the indirect estimator

v = sqrt(dt)\*randn(n,1);

b0 = [alpha ; kappa ; sig2];

b = fminsearch(@(b) q(b,dt,v,y0,bhat,n),b0,opt);

b(3) = abs(b(3));

theta(j,:) = b';

end

% Generate statistics on the sampling distribution

m = mean(theta);

stdev = std(theta);

rmse = sqrt(mean(bsxfun(@minus,theta,[alpha kappa sig2]).^2));

disp(' ');

disp(['Number of replications = ', num2str(nreps) ]);

disp(['Sample size = ', num2str(t) ]);

disp(' True Mean Std.err. RMSE ')

disp([[alpha kappa sig2]' m' stdev' rmse']);

end

%

%------------------------- Functions -------------------------------------%

%

%-------------------------------------------------------------------------%

% Objective function to compute the indirect estimator

%-------------------------------------------------------------------------%

function retp = q(b,dt,v,y0,bhat,n)

b(3) = abs(b(3));

ys = recserar(b(1)\*b(2)\*dt + sqrt(b(3))\*v,y0,1-b(2)\*dt);

% Generate discrete time data by choosing every t-th observation

nys = zeros(n\*dt,1);

for i = 1:n\*dt

temp = i\*(1/dt);

nys(i) = ys(temp);

end

ys = nys;

xs = [ones(size(ys,1)-1,1),trimr(ys,0,1)];

ys = trimr(ys,1,0);

b = xs\ys;

bhats = zeros(3,1);

bhats(1) = b(1)/(1 - b(2));

bhats(2) = 1 - b(2);

bhats(3) = mean((ys - xs\*b).^2);

retp = ( (bhat - bhats)'\*(bhat - bhats) );

end

% ========================================================================

%

% Estimate a multifactor model of the Stock-Watson business cycle model

% using indirect estimation based on Gallant and Tauchen.

% Auxiliary model is a VAR.

%

% ========================================================================

function sim\_bcycle( )

clear all

clc

RandStream.setDefaultStream( RandStream('mt19937ar','seed',12345) );

% Parameter values

t = 600;

lam = [ 1 ; 0.8 ; 0.7 ; 0.5 ; -0.5 ; -1 ];

sig = 0.2\*seqa(1,1,6)';

phi = 0.8;

% Generate data

eta = randn(t,1);

z = randn(t,6);

s = recserar( eta, 0.0, phi);

y = bsxfun(@times,lam',s) + bsxfun(@times,sig',z);

% Estimate the auxiliary model by a sequence of ar(1) regressions

g = [ ];

bhat = [ ];

for i = 1:6;

yvar = trimr(y(:,i),2,0);

xvar = [ones(length(yvar),1) trimr(y(:,i),1,1) trimr(y(:,i),0,2) ];

b = xvar\yvar;

vhat = yvar - xvar\*b;

s2 = mean(vhat.^2);

bhat = [ bhat [b ; s2] ];

% Moment conditions based on the data

g = [ g [bsxfun(@times,vhat,xvar)/s2 (0.5\*vhat.^2/s2^2 - 0.5/s2)] ];

end

iinv = inv(g'\*g/length(g));

% Simulation Estimation

n = 10\*t; % Length of simulation run

etas = randn(n,1); % Fix simulated disturbances

zs = randn(n,6);

% Call optimiser

opt = optimset('LargeScale','off','Display','off');

theta0 = [ lam' sig' phi] ;

[theta,qmin] = fminunc(@(b) q(b,bhat,etas,zs,iinv),theta0,opt);

dof = length(bhat(:))-length(theta);

jstat = t\*qmin;

disp('Parameter estimates');

disp(' True Estimated ')

disp([theta0' theta'])

disp(['Value of the objective function (Q) = ' num2str(qmin) ]);

disp(['Value of the J-test (TQ) = ' num2str(jstat) ]);

disp(['Degrees of freedom = ' num2str(dof) ]);

disp(['P-value = ' num2str(1-chi2cdf(jstat,dof)) ]);

end

%

%------------------------- Functions -------------------------------------%

%

%-------------------------------------------------------------------------%

% Objective function to compute the EMM estimator

%-------------------------------------------------------------------------%

function ret = q(b,bhat,etas,zs,iinv)

lam = b(1:6);

sig = b(7:12);

phi = b(13);

ys = bsxfun(@times,lam,recserar( etas, 0.0, phi)) + bsxfun(@times,sig,zs);

gs = [ ];

for i = 1:6

% Estimate auxiliary model evaluated at bhat

yvar = trimr(ys(:,i),2,0);

xvar = [ones(length(yvar),1) trimr(ys(:,i),1,1) trimr(ys(:,i),0,2)];

vhats = yvar - xvar\*bhat([1 2 3],i);

gs = [gs bsxfun(@times,vhats,xvar)/bhat(4,i) (0.5\*vhats.^2/bhat(4,i)^2 - 0.5/bhat(4,i)) ];

end

ret = mean(gs)\*iinv\*mean(gs)';

end

% ========================================================================

%

% Estimate a multifactor model of the Stock-Watson business cycle model

% using indirect estimation based on Gallant and Tauchen.

% Auxiliary model is a VAR.

%

% ========================================================================

function sim\_stockwatson( )

clear all

clc

RandStream.setDefaultStream( RandStream('mt19937ar','seed',12345) );

% Load monthly data September 1959 to September 2009 for Australia

load('bcycle.mat')

employed = data(:,1); % Employed

gdp = data(:,2); % GDP

hincome = data(:,3); % Household income

indprod = data(:,4); % Industrial production

retail = data(:,5); % Retail sales

unemp = data(:,6); % Unemployment rate

index = data(:,7); % Coincident index

y = log( [employed gdp hincome indprod retail 1./unemp] );

y = 100\*(trimr(y,3,0) - trimr(y,0,3));

y = bsxfun(@minus,y,mean(y));

y = bsxfun(@rdivide,y,std(y));

t = length(y);

% corrx((trimr(ln(index),3,0)-trimr(ln(index),0,3))~y);

% Estimate the auxiliary model by a sequence of ar(1) regressions

g = [ ];

bhat = [ ];

for i = 1:6;

yvar = trimr(y(:,i),2,0);

xvar = [trimr(y(:,i),1,1) trimr(y(:,i),0,2) ];

b = xvar\yvar;

uhat = yvar - xvar\*b;

s2 = mean(uhat.^2);

bhat = [ bhat [b ; s2] ];

% Moment conditions based on the data

g = [ g [bsxfun(@times,uhat,xvar)/s2 (0.5\*uhat.^2/s2^2 - 0.5/s2)] ];

end

iinv = inv(g'\*g/length(g));

% Simulation Estimation

n = 30\*t; % Length of simulation run

etas = randn(n,1); % Fix simulated disturbances

zs = randn(n,6);

% Call optimiser

opt = optimset('LargeScale','off','Display','iter');

theta0 = [rand(1,12) 1];

[theta,qmin] = fminunc(@(b) q(b,bhat,etas,zs,iinv),theta0,opt);

theta(end) = tanh(theta(end));

dof = length(bhat(:))-length(theta);

jstat = t\*qmin;

disp('Parameter estimates');

disp('Parameter Estimates ')

disp(theta')

disp(['Value of the objective function (Q) = ' num2str(qmin) ]);

disp(['Value of the J-test (TQ) = ' num2str(jstat) ]);

disp(['Degrees of freedom = ' num2str(dof) ]);

disp(['P-value = ' num2str(1-chi2cdf(jstat,dof)) ]);

end

%

%------------------------- Functions -------------------------------------%

%

%-------------------------------------------------------------------------%

% Objective function to compute the EMM estimator

%-------------------------------------------------------------------------%

function ret = q(b,bhat,etas,zs,iinv)

lam = b(1:6);

sig = b(7:12);

phi = tanh(b(13));

ys = bsxfun(@times,lam,recserar( etas, 0.0, phi)) + bsxfun(@times,sig,zs);

ys = bsxfun(@minus,ys,mean(ys));

gs = [ ];

for i = 1:6

% Estimate auxiliary model evaluated at bhat

yvar = trimr(ys(:,i),2,0);

xvar = [ trimr(ys(:,i),1,1) trimr(ys(:,i),0,2)];

vhats = yvar - xvar\*bhat([1 2],i);

gs = [gs bsxfun(@times,vhats,xvar)/bhat(3,i) (0.5\*vhats.^2/bhat(3,i)^2 - 0.5/bhat(3,i)) ];

end

ret = mean(gs)\*iinv\*mean(gs)';

end