```
/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@
Simulated Density Paper
Jun 2005

Emperical section: Testing the two specifications on the Euro-dollar rate
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@*/
new; cls;
library cml, pgraph;
clearg Tg, ee_gmm, ee_sim;

output file = c:\emp.out reset;
outwidth 255; output on; format /MA1 /LD 6,4;

load xt[] = c:\ed1_w.txt;
xt=xt/100;
/*uncomment the following line for the smaller data set considered in the paper*/
/*xt=xt[992:rows(xt),.];*/
Tg=Rows(xt);                                          /*Rows of Xt enters as a global
variable usedin some of the
                                                       procedures: CHANGING THIS WILL
CAUSE ERRORS*/
tao=1|2|4|12;                                         /*Sets the value of tao*/
bs=100;                                                 /*number of bootstrap
replications*/
/*Changing the following specifications will generate the complete tables 6 and 7*/


S=10*(rows(xt)-tao[rows(tao)]);                       /*selects the number of
simulation paths
                                                      other choices include
10*(rows(xt)-tao[rows(tao)]);
                                                      and
30*(rows(xt)-tao[rows(tao)]);*/
mod_ind=2;                                            /*Model indicator 1 for CIR 2
for OU*/
bl=20;                                                /*block length for the
bootstrap*/
u_bar=(meanc(xt)-1*stdc(xt))|(meanc(xt)+1*stdc(xt));/*the interval u_bar*/
/*SELECTION OF DIFFERENT COMBINATIONS ENDS HERE*/


"SELECTIONS";
"bl = " bl;
if mod_ind==1;
    "NULL: CIR";
elseif mod_ind==2;
    "NULL: OU";
endif;
        if s <= 10*(rows(xt)-tao[rows(tao)]); "S =
10*T";s=10*(rows(xt)-tao[rows(tao)]);
        elseif s <= 20*(rows(xt)-tao[rows(tao)]); "S =
20*T";s=20*(rows(xt)-tao[rows(tao)]);
        else; "S = 30*T";s=30*(rows(xt)-tao[rows(tao)]); endif;
        ee_sim=sqrt(1/TG)*rndn(tao[rows(tao)]*TG,S);
        {vt,sup_vt}=BCS_stat(xt,tao,s,mod_ind,xt,1/Tg,u_bar);
        {b_sup_vt,cv95,cv90,cv80}=BCS_boot(xt,tao,s,mod_ind,bL,bs,vt,u_bar);

"STATISTICS AND CVs FOR DIFFERENT VALUES OF TAO";
        "Statistics     " sup_vt;
        "95%            " cv95;
        "90%            " cv90;
```

```
        "80%             " cv80;
end;

/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
* boot1: Generates the block bootstrap sample
           *
************************************************************************************
*************
* Inputs:       dat1    -   time series
               *
*                   lval    -   block boot length
                   *
* Output:   xb1     -   bootstrap sample
           *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@*/
proc (1) = boot1(dat1,lval);
local N,num_uns,undraw1,xbl,ib;
    N=rows(dat1);
    num_uns=N/lval;
    /* draw uniforms U[0,T-l+1] */
    undraw1=round((N-lval)*rndu(num_uns,1));
    xbl={};
    ib=1;
    do while ib<=num_uns;
    xbl=xbl|dat1[undraw1[ib]+1:undraw1[ib]+lval,.];
    ib=ib+1;
    endo;
retp(xbl);
endp;

/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
*   est_OU: returns the SGMM estimaes and standard errors for OU process
           *
************************************************************************************
*************
* Inputs:         x1      -   time series
               *
* Output:     b_OU   -   estimates
           *
*             se_OU  -   GMM standard errors
           *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@*/
proc (2)= est_OU(x);
local b_OU,f,g,cov,retcode,se_OU;
    b_OU=start_ou(x);
    cmlset;
    __output=0;_cml_Algorithm = 1;_cml_LineSearch = 2;_cml_GradMethod =
1;_cml_DirTol = 1e-5;_cml_DFTol = 1e-5;
    _cml_C = {  1 0 0,
                0 0 1};
    _cml_D = {  0,
                0};
    ee_gmm=sqrt(1/Tg)*rndn(10*Tg*Tg,1);
    {b_ou,f,g,cov,retcode} = CML(x,0,&SGMM_OU,b_OU);
    SE_OU=OU_SE(b_ou,x);
    SE_OU=SQRT(DIAG(SE_OU));
retp(b_ou,se_ou);
endp;
```

```
/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
*   est_cir: returns the SGMM estimaes and standard errors for CIR process
           *
***************************************************************************************
*************
* Inputs:         x1      -   time series s
               *
* Output:     b_cir   -   estimates
           *
*             se_cir  -   GMM standard errors
           *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@*/
proc (2)= est_cir(x);
local b_cir,f,g,cov,retcode,se_cir;
    b_cir=start_cir(x);
    cmlset;
    __output=0;_cml_Algorithm = 1;_cml_LineSearch = 2;_cml_GradMethod =
1;_cml_DirTol = 1e-5;_cml_DFTol = 1e-5;
    _cml_C = {  1 0 0,
                0 1 0,
                0 0 1};
    _cml_D = {  0,
                0,
                0};
    _cml_IneqProc = &ineq_CIR;
    ee_gmm=sqrt(1/Tg)*rndn(10*Tg*Tg,1);
    {b_CIR,f,g,cov,retcode} = CML(X,0,&SGMM_CIR,b_CIR);
    SE_CIR=CIR_SE(b_CIR,X);
    SE_CIR=SQRT(DIAG(SE_CIR));
retp(b_cir,se_cir);
endp;

/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
*   SGMM_OU: Returns the obj func
           *
*   dp(t)=phi*(p_bar-p(t))*dt+(sig2)*dW(t)
           *
***************************************************************************************
*************
* Inputs:       b1      -   starting values
               *
*                 x1      -   time series
                 *
* Output:           -   the objective function to be minimised
           *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@*/
proc SGMM_OU(b1,x1);
local XS, g_prime, g, q, meanf, f, w0, invW, v, W, auto,s;
        s=rows(x1)*10;
    xs=dgp_OUS(s,1/Tg,b1[1],b1[2],b1[3],b1[2]);
    g_prime= meanc(X1) - meanc(Xs);
        g_prime=
g_prime~(((meanc((x1[1:rows(x1)-1]-meanc(x1)).*(x1[2:rows(x1)]-meanc(x1))))) -
((meanc((xs[1:rows(xs)-1]-meanc(xs)).*(xs[2:rows(xs)]-meanc(xs))))));
        g_prime= g_prime~(vcx(x1) - vcx(xs));
    g=g_prime';
    q=int(rows(X1)^(1/6));

f=(X1[2:rows(x1)])~(((x1[1:rows(x1)-1]-meanc(x1)).*(x1[2:rows(x1)]-meanc(x1))))~((x1
```

```
[2:rows(x1)]-meanc(x1))^2);
        f=f';
        meanf=meanc(f');
        w0=(1/rows(X1))*((f-meanf)*(f-meanf)');
        invW=w0;
        if q > 0;
                v=0;
                do while v < q; v=v+1;

auto=(1/rows(x1))*(((f[.,v+1:rows(f')]-meanf)*(f[.,1:rows(f')-v]-meanf)')+((f[.,1:ro
ws(f')-v]-meanf)*(f[.,v+1:rows(f')]-meanf)'));
                        invW=invW+(1-(v/(q+1)))*auto;
                endo;
        endif;
        W=inv(invW);
retp(-g_prime*W*g);
endp;

/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
*   SGMM_CIR: Returns the obj func
           *
*   dp(t)=phi*(p_bar-p(t))*dt+(sig1*sqrt(p(t)))*dW(t)
           *
**************************************************************************************
*************
* Inputs:       b1      -   starting values
               *
*                   x1      -   time series
                   *
* Output:               -   the objective function to be minimised
           *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@*/
proc SGMM_CIR(b1,x1);
local XS, g_prime, g, q, meanf, f, w0, invW, v, W, auto,s;
        s=rows(x1)*10;
    xs=dgp_cirS(s,1/Tg,b1[1],b1[2],b1[3],b1[2]);
    g_prime= meanc(X1) - meanc(Xs);
        g_prime=
g_prime~(((meanc((x1[1:rows(x1)-1]-meanc(x1)).*(x1[2:rows(x1)]-meanc(x1))))) -
((meanc((xs[1:rows(xs)-1]-meanc(xs)).*(xs[2:rows(xs)]-meanc(xs))))));
        g_prime= g_prime~(vcx(x1) - vcx(xs));
    g=g_prime';
    q=int(rows(X1)^(1/6));

f=(X1[2:rows(x1)])~(((x1[1:rows(x1)-1]-meanc(x1)).*(x1[2:rows(x1)]-meanc(x1))))~((x1
[2:rows(x1)]-meanc(x1))^2);
        f=f';
        meanf=meanc(f');
        w0=(1/rows(X1))*((f-meanf)*(f-meanf)');
        invW=w0;
        if q > 0;
                v=0;
                do while v < q; v=v+1;

auto=(1/rows(x1))*(((f[.,v+1:rows(f')]-meanf)*(f[.,1:rows(f')-v]-meanf)')+((f[.,1:ro
ws(f')-v]-meanf)*(f[.,v+1:rows(f')]-meanf)'));
                        invW=invW+(1-(v/(q+1)))*auto;
                endo;
        endif;
        W=inv(invW);
retp(-g_prime*W*g);
```

```
endp;

/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
* BCS_boot: Calculates The Statistics
           *
**********************************************************************************
*************
* Inputs:       xt      -   time series
               *
*                   tao     -   a vector indicating the different step ahead con.
int. to be examined   *
*                   s       -   number of sample paths to be simulated
                   *
*                   mod_ind -   model specification 1 for CIR and 2 for log ou
                   *
*           vt      -   pre sup pre sum vt to use in bootstrap will have
rows(tao)*18 cols      *
*           bL      -   block boot lengt
           *
*           bs      -   number of bootstrap replication
           *
*                   u_bar   -   confidence interval
                       *
* Output:   b_sup_vt-   boot strap distribution, sorted 100 rows and rows(tao)*3
cols           *
*           cv95    -   95% critical value
           *
*           cv90    -   90% critical value
           *
*           cv80    -   80% critical value
           *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@*/
proc (4)=BCS_boot(xt,tao,s,mod_ind,bL,bs,vt,u_bar);
local xb,ii,b_vt,sup_vt,t_tao,t_tao_b,vt_store,b_sup_vt,jj,cv95,cv90,cv80;
    t_tao=rows(vt);
    vt=(1/sqrt(t_tao))*sumc(vt);
    ii=0;b_sup_vt={};
    do while ii < bs;ii=ii+1;
"boot rep" ii;
        xb=boot1(xt,bL);
        {b_vt,sup_vt}=BCS_stat(xb,tao,s,mod_ind,xt,1/Tg,u_bar);
        t_tao_b=rows(b_vt);
        b_vt=(1/sqrt(t_tao_b))*sumc(b_vt);
        b_vt=abs(b_vt-vt);
        jj=0;vt_store={};
        do while jj < rows(tao);
            vt_store=vt_store~maxc(b_vt[jj*6+1:jj*6+6]);
            jj=jj+1;
        endo;
        b_sup_vt=b_sup_vt|vt_store;
    endo;
    jj=0;
    do while jj< cols(b_sup_vt);
        jj=jj+1;
        b_sup_vt[.,jj]=sortc(b_sup_vt[.,jj],1);
    endo;
    cv95=b_sup_vt[0.95*rows(b_sup_vt),.];
    cv90=b_sup_vt[0.90*rows(b_sup_vt),.];
    cv80=b_sup_vt[0.80*rows(b_sup_vt),.];
retp(b_sup_vt,cv95,cv90,cv80);
endp;
```

```
/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
* BCS_stat: Calculates The Test Statistics
                                    *
************************************************************************************
*****************************
* Inputs:       xt      -       time series
                                *
*                   tao     -           a vector indicating the different step ahead
con. int. to be examined                *
*                   s       -           number of sample paths to be simulated
                                               *
*                   mod_ind -           model specification 1 for CIR and 2 for log
ou                                      *
*               xt_v    -       time series for calculating v
                                        *
*                   h       -           discretization interval
                                                *
*                   u_bar   -           confidence interval
                                            *
* Output:       vt      -       pre sup pre sum vt to use in bootstrap procedure
                                            *
*               sup_vt  -       Test statistics will have rows(tao) results
                            *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@*/
proc (2)= BCS_stat(xt,tao,s,mod_ind,xt_v,h,u_bar);
local
v,p_true,b,ii,x_sim,p_sim,jj,v_t,vt,sup_vt,v_ind,t_tao,bse,kk,p_sim1,p_sim2,tao3,se,
count;

    v=sortc(xt_v,1);

v=v[0.1*rows(v)]|v[0.3*rows(v)]|v[0.5*rows(v)]|v[0.7*rows(v)]|v[0.9*rows(v)]|v[rows(
v)];
    p_true=(xt .> u_bar[1,1]).*(xt .< u_bar[2,1]);
    /*estimate the model here*/h=1/Tg;
    if mod_ind==1;
        {b,se}=est_cir(xt);
    elseif mod_ind==2;
        {b,se}=est_ou(ln(xt));
    endif;
    p_sim=zeros(rows(p_true)-tao[rows(tao)],rows(tao));
    ii=0;count=0;
    do while ii<S;ii=ii+1;
        ee_gmm=ee_sim[.,ii];
        x_sim={};
        if mod_ind==1;
                x_sim=dgp_cirS(tao[rows(tao)],h,b[1],b[2],b[3],xt[ii]);
        elseif mod_ind==2;
                x_sim=exp(dgp_ouS(tao[rows(tao)],h,b[1],b[2],b[3],ln(xt[ii])));

        endif;
        X_sim=x_sim.*ones(1,rows(xt)-tao[rows(tao)]);
        jj=0;
        do while jj < rows(tao);
            p_sim[.,1+jj]=p_sim[.,1+jj]
                    +((x_sim[tao[1+jj],.] .> u_bar[1,1]).*(x_sim[tao[1+jj],.] .<
u_bar[2,1]))';
            jj=jj+1;
        endo;
        if ii==rows(xt)-tao[rows(tao)];
```

```
            count=count+ii;ii=0;
        endif;
        if count==S;
            ii=count;
        endif;
    endo;
    p_sim=p_sim./s;
    v_t=zeros(rows(p_true)-tao[rows(tao)],rows(tao));
    t_tao=rows(p_sim);
        v_ind=(xt[1:rows(xt)-tao[rows(tao)]] .< v[1])~(xt[1:rows(xt)-tao[rows(tao)]]
.< v[2]);
        v_ind=v_ind~(xt[1:rows(xt)-tao[rows(tao)]] .<
v[3])~(xt[1:rows(xt)-tao[rows(tao)]] .< v[4]);
        v_ind=v_ind~(xt[1:rows(xt)-tao[rows(tao)]] .<
v[5])~(xt[1:rows(xt)-tao[rows(tao)]] .< v[6]);
    sup_vt={};
    jj=0;vt={};
    do while jj < rows(tao);

v_t[.,1+jj]=p_sim[.,1+jj]-p_true[tao[jj+1]+1:rows(xt)-tao[rows(tao)]+tao[jj+1],.];
        jj=jj+1;
        vt=vt~(v_t[.,jj].*v_ind);
        sup_vt=sup_vt~(maxc   (   abs(  (1/sqrt(t_tao))*sumc(v_t[.,jj].*v_ind)  )
)   );
    endo;
retp(vt,sup_vt);
endp;

/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
* ineq_CIR: Single Factor Square Root Process Nonlinear Inequality constraints
           *
**************************************************************************************
*************
*   dp(t)=phi*(p_bar-p(t))*dt+(sig1*sqrt(p(t)))*dW(t)
           *
*   Nonlinear Inequality constraints
           *
*   2*phi*p_bar >= sig1^2
           *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@*/
proc ineq_CIR(b);
retp(2*b[1]*b[2]-b[3]^2);
endp;


/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
*   OU_SE: Returns the gmm var-cov matrix for OU Process
           *
**************************************************************************************
*************
* Inputs:       b1      -   estimated values
               *
*                   x1      -   time series
                   *
* Output:           -   gmm var-cov matrix
           *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@*/

proc OU_SE(b1,x1);
```

```
local se, XS, g_prime, g, q, meanf, f, w0, invW, v, W, auto,s;
    W=SE_W(b1,x1);
    q=GRADp(&grad_f21,b1);
    g=meanc(q);
    q=GRADp(&grad_f22,b1);
    g=g~meanc(q);
    q=GRADp(&grad_f23,b1);
    g=g~meanc(q);
    se=inv(g*W*g');
retp(se/rows(x1));
endp;
proc grad_f21(b);
local g,s,xs;
        s=Tg*10;
        Xs=dgp_OUS(s,1/TG,b[1],b[2],b[3],b[2]);
    g= (Xs);
retp(g);
endp;
proc grad_f22(b);
local g,s,xs;
        s=Tg*10;
        Xs=dgp_OUS(s,1/TG,b[1],b[2],b[3],b[2]);
        g= (xs[1:rows(xs)-1]-meanc(xs)).*(xs[2:rows(xs)]-meanc(xs));
retp(g);
endp;
proc grad_f23(b);
local g,s,xs;
        s=Tg*10;
        Xs=dgp_OUS(s,1/TG,b[1],b[2],b[3],b[2]);
        g= (xs-meanc(xs))^2;
retp(g);
endp;

/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
*   CIR_SE: Returns the gmm var-cov matrix for CIR Process
              *
*************************************************************************************
*************
* Inputs:       b1      -   estimated values
               *
*                   x1      -   time series
                   *
* Output:               -   gmm var-cov matrix
              *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@*/

proc CIR_SE(b1,x1);
local se, XS, g_prime, g, q, meanf, f, w0, invW, v, W, auto,s;
    W=SE_W(b1,x1);
    q=GRADp(&grad_f11,b1);
    g=meanc(q);
    q=GRADp(&grad_f12,b1);
    g=g~meanc(q);
    q=GRADp(&grad_f13,b1);
    g=g~meanc(q);
    se=inv(g*W*g');
retp(se/rows(x1));
endp;
proc SE_W(b1,x1);
local se, XS, g_prime, g, q, meanf, f, w0, invW, v, W, auto,s;
    q=int(rows(X1)^(1/6));
```

```
f=(X1[2:rows(x1)])~(((x1[1:rows(x1)-1]-meanc(x1)).*(x1[2:rows(x1)]-meanc(x1))))~((x1
[2:rows(x1)]-meanc(x1))^2);
        f=f';
        meanf=meanc(f');
        w0=(1/rows(X1))*((f-meanf)*(f-meanf)');
        invW=w0;
        if q > 0;
                v=0;
                do while v < q; v=v+1;

auto=(1/rows(x1))*(((f[.,v+1:rows(f')]-meanf)*(f[.,1:rows(f')-v]-meanf)')+((f[.,1:ro
ws(f')-v]-meanf)*(f[.,v+1:rows(f')]-meanf)'));
                        invW=invW+(1-(v/(q+1)))*auto;
                endo;
        endif;
        W=inv(invW);
retp(W);
endp;
proc grad_f11(b);
local g,s,xs;
        s=Tg*10;
        Xs=dgp_cirS(s,1/TG,b[1],b[2],b[3],b[2]);
    g= (Xs);
retp(g);
endp;
proc grad_f12(b);
local g,s,xs;
        s=Tg*10;
        Xs=dgp_cirS(s,1/TG,b[1],b[2],b[3],b[2]);
        g= (xs[1:rows(xs)-1]-meanc(xs)).*(xs[2:rows(xs)]-meanc(xs));
retp(g);
endp;
proc grad_f13(b);
local g,s,xs;
        s=Tg*10;
        Xs=dgp_cirS(s,1/TG,b[1],b[2],b[3],b[2]);
        g= (xs-meanc(xs))^2;
retp(g);
endp;

/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
* dgp_ou:   Data Generation as OU process
           *
*           dp(t)=phi*(p_bar-p(t))*dt+sig*dW(t)
           *
************************************************************************************
*************
* Inputs:       T       -   Length of time series
               *
*                   h       -   discretization interval
                   *
*                   phi     -   mean reversion parameter Process
                   *
*                   p_bar   -   mean level
                   *
*                   sig1    -   variance term
                   *
*                   start   -   starting value of the process
                   *
* Output:   dat     -   time series
           *
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@*/
proc dgp_ou(T,h,phi,p_bar,sig1,start);
local TN,dat,Pt,i,X1,ee;
        TN=round(T/h);
    ee=sqrt(h)*rndn(T/h,1);
        dat={};
        Pt=start;
        i=0;
        do while i < TN;
                i=i+1;
                X1=Pt+phi*(p_bar-Pt)*h+sig1*ee[i];
                if i%(h^-1)==0;
                        dat=dat|X1;
                endif;
                Pt=X1;
        endo;
retp(dat);
endp;
/*same as above but for sgmm with errors that do not change over optimization*/
proc dgp_ous(T,h,phi,p_bar,sig1,start);
local TN,dat,Pt,i,X1,ee;
        TN=round(T/h);
    ee=ee_gmm;
        dat={};
        Pt=start;
        i=0;
        do while i < TN;
                i=i+1;
                X1=Pt+phi*(p_bar-Pt)*h+sig1*ee[i];
                if i%(h^-1)==0;
                        dat=dat|X1;
                endif;
                Pt=X1;
        endo;
retp(dat);
endp;

/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
* start_ou: Returns the starting values for the OU process
           *
*          dp(t)=phi*(p_bar-p(t))*dt+sig*dW(t)
           *
************************************************************************************
*************
* Inputs:   y   -   time series
           *
* Output:   b   -   1x3 vector of starting values for phi p_bar and sig
           *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@*/
proc start_ou(y);
local mu,rho,sig,phi,test;
    mu=meanc(y);
    rho=meanc((y[1:rows(y)-1]-meanc(y)).*(y[2:rows(y)]-meanc(y)))/vcx(y);
    phi=-ln(abs(rho));
    sig=(vcx(y)*2*phi)^.5;

retp(phi~mu~sig);
endp;
```

```
/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
* dgp_cir:   Data Generation as CIR process
           *
*           dp(t)=phi*(p_bar-p(t))*dt+sig*sqrt(p(t)*dW(t)
           *
************************************************************************************
*************
* Inputs:       T       -   Length of time series
               *
*                   h       -   discretization interval
                   *
*                   phi     -   mean reversion parameter Process
                   *
*                   p_bar   -   mean level
                   *
*                   sig1    -   variance term
                   *
*                   start   -   starting value of the process
                   *
* Output:   dat     -   time series
           *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@*/
proc dgp_cir(T,h,phi,p_bar,sig1,start);
local TN,dat,Pt,i,X1,ee;
        TN=round(T/h);
    ee=sqrt(h)*rndn(T/h,1);
        dat={};
        Pt=start;
        i=0;
        do while i < TN;
                i=i+1;
                X1=Pt+phi*(p_bar-Pt)*h+sig1*sqrt(pt)*ee[i]
            -0.5*(sig1*sqrt(pt))*(0.5*sig1/sqrt(pt))*h
            +0.5*(sig1*sqrt(pt))*(0.5*sig1/sqrt(pt))*(ee[i]^2);
        /*Theory implies that this condition will never be reached as when process
approaches
        zero the volatility is switched off, that result depends on h going to zero
here h is
        very small indeed I am including the condition as a final safe guard, here I
am switching
        off the volatility manually something that should happens assymptotically*/
        if X1 < 0;
            X1=Pt+phi*(p_bar-Pt)*h;
        endif;
                if i%(h^-1)==0;
                        dat=dat|X1;
                endif;
                Pt=X1;
        endo;
retp(dat);
endp;
/*same as above but for sgmm with errors that do not change over optimization*/
proc dgp_cirS(T,h,phi,p_bar,sig1,start);
local TN,dat,Pt,i,X1,ee;
        TN=round(T/h);
    ee=EE_GMM;
        dat={};
        Pt=start;
        i=0;
        do while i < TN;
                i=i+1;
```

```
                X1=Pt+phi*(p_bar-Pt)*h+sig1*sqrt(pt)*ee[i]
            -0.5*(sig1*sqrt(pt))*(0.5*sig1/sqrt(pt))*h
            +0.5*(sig1*sqrt(pt))*(0.5*sig1/sqrt(pt))*(ee[i]^2);
        if X1 < 0;
            X1=Pt+phi*(p_bar-Pt)*h;
        endif;
                if i%(h^-1)==0;
                        dat=dat|X1;
                endif;
                Pt=X1;
        endo;
retp(dat);
endp;

/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
* start_cir: Returns the starting values for the CIR process
           *
*           dp(t)=phi*(p_bar-p(t))*dt+sig*sqrt(p(t)*dW(t)
           *
**********************************************************************************
*************
* Inputs:   y   -   time series
           *
* Output:   b   -   1x3 vector of starting values for phi p_bar and sig
           *
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@*/
proc start_cir(y);
local b,mu,rho,sig,phi,test;
    mu=meanc(y);
    rho=meanc((y[1:rows(y)-1]-meanc(y)).*(y[2:rows(y)]-meanc(y)))/vcx(y);
    phi=-ln(abs(rho));
    sig=(vcx(y)*2*phi/mu)^.5;
    b=phi~mu~sig;
retp(b);
endp;
```