```
/* this program constructs test statistics both accounting for */
/* parameter estimation error and assuming that parameter estimation */
/* error vanishes */

/* the program is calibrated for the empirical example in */

/* Predictive Density and Conditional Confidence Interval Accuracy Tests */

/* by Valentina Corradi and Norman R. Swanson */

/* Density test paper - revised version empirical             */
/* general program                                            */
/* empirical                                                  */
/* rolling or recursive estimation, linear models             */
/* densn1.prg  -            aug 2004                          */
/*                                                            */
/*                                                            */

format /MA1/LD 12,8;
outwidth 255;
output file = c:\densn1.out reset;
output on;

clearg  UU,Unum,Uval,smpl,ALLDAT,ALLDAT1;

/* Parameters to Set Before Carrying out analysis             */

/* block lengths to try in all boot critical value constructions*/

ls1={3,5,10,15,20,25};

/* maximum number of lags in AR models used - the X variables */

p_ar=0; /* set = 0 as lags already in inputed dataset */

/* number of additional exogenous variables - the Z variables */
/* NOTE: the highest lag taken when forming any Z variable must be <= p_ar */

p_ex=0;

/* number of boot replications for CV calc */

Bootnum=100;

/* estimation rolling == 1, recursive otherwise */

roll_yn=2;

/* input dataset */

/* data are from 1954:1-2003:12 (600 obs) */
/* vars are inf, inf_t-1 inf_t-2, inf_t-3, unem_t-1 */


load datin[600,5]=c:\infun1.dat;

/* total sample after lags */

/* smpl=600-p_ar-1; */ /* assume differences tekn in this program, and not
beforehand */
 smpl=600-p_ar;

/* define prediction period */
```

```
P=smpl/2;

/* Define u range to use */
/* Problem is that U range for the indictor function part of Z */
/* should be much wider than U range used for the conditional distribution */
/* part of Z !! */

UU={};
Unum=100; /* number of u's to average over */
Umult=2; /* how far a CI around mean to make Umin and Umax */

Udat=datin[.,1];
/*Udat=trimr(ln(Udat)-lagn(ln(Udat),1),1,0);*/
/*
Umin=(meanc(Udat)-(Umult*stdc(Udat)));
Umax=(meanc(Udat)+(Umult*stdc(Udat)));
print "mean - used in Umin,Umax";
print meanc(Udat);
print "stdc - used in Umin,Umax";
print stdc(Udat);
*/
Umin=minc(Udat);
Umax=maxc(Udat);

i=1;
do while i<=Unum;
  UU=UU|(Umin+((i-1)*((Umax-Umin)/Unum)));
  i=i+1;
endo;


/* ---------------------------------------------------------- */
/* ^^^^^^^^^^^^^^^^^^^ PROCEDURES ^^^^^^^^^^^^^^^^^^^^^^^^^^ */
/* ---------------------------------------------------------- */

PROC ilogistic(p,a,b); /* quantile */
  RETP(a-ln(1/p-1)/b);
ENDP;

PROC rlogistic(r,c,a,b); /* generate a random number */
  RETP(ilogistic(rndu(r,c),a,b));
ENDP;

PROC istudent(p,v);
  RETP(CDFTCI(1-p,v));
ENDP;

PROC rstudent(r,c,v);
  RETP(istudent(RNDU(r,c),v));
ENDP;

proc rsum(y,x,beta) ;
  local resid ;
  resid = y - x*beta ;
  retp(resid'resid) ;
endp ;
/*
proc lagn(x,n);
  local y;
  y = shiftr(x', n, (miss(0, 0))');
  retp(y');
endp;
```

```
*/
proc regr(y,x);
  retp(y/x);
endp;

proc diff(x,k) ;
  if ( k == 0) ;
     retp(x) ;
  endif ;
  retp(trimr(x,k,0)-trimr(lagn(x,k),k,0)) ;
endp ;

/* ------------------------------------------------------------ */
/* PROCEDURE - estimation and prediction -- least squares       */
/* ------------------------------------------------------------ */

proc (9) = pred(YY,XX,P,roll_yn);
local
ii,NN,aa,bb,bet,fcst,fcsterr,yhat,ehat,s2hat,fY,fe,Yh,allb,eh,beth,s2hath,alldat,adj
A,adjAA;

NN=rows(YY);

fY={};
fe={};
Yh={};
eh={};
allb={};
adjAA={};

ii=1;
do while ii<=P;

   if ii==P;
      /* note, even if rolling estimation, do one estimation */
      /* with entire sample to get thetahat_T and associated residuals, etc. */
      /* for use in Hessian, score, etc. */
      /* could have also used thetahat_P, say, but then scores would need */
      /* to be averaged over all T obervations rather than only P */
      /* observations, as is done when using thetahat_T */
      aa=1; bb=NN-P-1+ii;
      beth=inv(XX[aa:bb,.]'*XX[aa:bb,.])*XX[aa:bb,.]'*YY[aa:bb,.];
      yh=XX[aa:bb,.]*beth;
      eh=YY[aa:bb,.]-yh;
      s2hath=vcx(eh);
      alldat=YY[aa:bb,.]~XX[aa:bb,.];
   endif;

   if roll_yn==1;
      aa=ii; bb=NN-P-1+ii;
   else;
      aa=1; bb=NN-P-1+ii;
   endif;

   bet=inv(XX[aa:bb,.]'*XX[aa:bb,.])*XX[aa:bb,.]'*YY[aa:bb,.];

   adjA = (meanc(XX.*(YY-XX*bet)))';
   adjAA=adjAA|adjA;

   fcst=XX[bb+1,.]*bet;
   fcsterr=YY[bb+1,.]-fcst;

   yhat=XX[aa:bb,.]*bet;
```

```
   ehat=YY[aa:bb,.]-yhat;

   s2hat=vcx(ehat);

   /* collect things together */

   fY=fY|fcst;    /* will be a Px1 column */
   fe=fe|fcsterr; /* will be a Px1 column */

   allb=allb|(bet'~s2hat); /* this is a P x cols(XX) matrix of parm est */

   ii=ii+1;
endo;

retp(fY,fe,Yh,eh,beth,s2hath,alldat,allb,adjAA);
endp;

/* ------------------------------------------------------------------------ */
/* PROCEDURE - estimation and prediction bootstrap -- least squares         */
/* ------------------------------------------------------------------------ */

proc (8) = predb(YY,XX,P,roll_yn,recterm);
local
ii,NN,aa,bb,bet,fcst,fcsterr,yhat,ehat,s2hat,fY,fe,Yh,allb,eh,beth,s2hath,allX;

NN=rows(YY);

fY={};
fe={};
Yh={};
eh={};
allb={};

ii=1;
do while ii<=P;

   if ii==P;
      /* note, even if rolling estimation, do one estimation */
      /* with entire sample to get thetahat_T and associated residuals, etc. */
      /* for use in Hessian, score, etc. */
      /* could have also used thetahat_P, say, but then scores would need */
      /* to be averaged over all T obervations rather than only P */
      /* observations, as is done when using thetahat_T */
      aa=1; bb=NN-P-1+ii;
      beth=inv(XX[aa:bb,.]'*XX[aa:bb,.])*XX[aa:bb,.]'*YY[aa:bb,.];
      yh=XX[aa:bb,.]*beth;
      eh=YY[aa:bb,.]-yh;
      s2hath=vcx(eh);
      allX=XX[aa:bb,.];
   endif;

   if roll_yn==1;
      aa=ii; bb=NN-P-1+ii;
   else;
      aa=1; bb=NN-P-1+ii;
   endif;


bet=inv(XX[aa:bb,.]'*XX[aa:bb,.])*((XX[aa:bb,.]'*YY[aa:bb,.])-((bb-aa+1)*recterm[ii,
.]'));

   fcst=XX[bb+1,.]*bet;
   fcsterr=YY[bb+1,.]-fcst;
```

```
   yhat=XX[aa:bb,.]*bet;
   ehat=YY[aa:bb,.]-yhat;

   s2hat=vcx(ehat);

   /* collect things together */

   fY=fY|fcst;    /* will be a Px1 column */
   fe=fe|fcsterr; /* will be a Px1 column */

   allb=allb|(bet'~s2hat); /* this is a P x cols(XX) matrix of parm est */

   ii=ii+1;
endo;

retp(fY,fe,Yh,eh,beth,s2hath,allX,allb);
endp;

/* ------------------------------------------------------------         */
/* PROCEDURE - Calculate Z stats for each u in range of U - normal    */
/* ------------------------------------------------------------         */

proc (1) = Z_norm(pred_act,pred,sigg);
local ii,NN,ZZ;

NN=rows(pred_act);
ZZ={};
ii=1;
do while ii <= Unum;
  ZZ=ZZ|( (1/sqrt(NN))*sumc( ( (pred_act .le UU[ii]) - (cdfn((UU[ii]-pred)/sigg))
)^2 ) );
  ii=ii+1;
endo;

/* so one big row (Unum x 1) is outputed here with the Z piece for each individual u
element */

retp(ZZ);
endp;

/* ------------------------------------------------------------              */
/* PROCEDURE - Calculate Z stats for each u in range of U - Student's t    */
/* ------------------------------------------------------------              */

proc (1) = Z_studt(pred_act,pred,sigg);
local ii,NN,ZZ;

NN=rows(pred_act);
ZZ={};
ii=1;
do while ii <= Unum;
  ZZ=ZZ|( (1/sqrt(NN))*sumc( ( (pred_act .le UU[ii]) -
(1-(cdftc(((UU[ii]-pred)/sigg),3))) )^2 ) );
  ii=ii+1;
endo;

/* so one big row (Unum x 1) is outputed here with the Z piece for each individual u
element */

retp(ZZ);
endp;
```

```
/* ------------------------------------------------------------                */
/* PROCEDURE - Calculate Z stats for each u in range of U - normal  - bootstrap  */
/* ------------------------------------------------------------                */

proc (1) = Z_normb(pred_act,pred,sigg,alld,predactb,predb,siggb);
local ii,NN,ZZ,TT;

NN=rows(pred_act);
TT=rows(alld);
ZZ={};
ii=1;
do while ii <= Unum;
  ZZ=ZZ|( (1/sqrt(NN))*sumc( ( (predactb .le UU[ii]) - (cdfn((UU[ii]-predb)/siggb))
)^2
 - ( (1/TT)*sumc(( (alld[.,1] .le UU[ii]) - (cdfn((UU[ii]-pred)/sigg)) )^2) ) ) );
  ii=ii+1;
endo;

/* so one big row (Unum x 1) is outputed here with the Z piece for each individual u
element */

retp(ZZ);
endp;

/* ------------------------------------------------------------
 */
/* PROCEDURE - Calculate Z stats for each u in range of U - Student's t  - bootstrap
 */
/* ------------------------------------------------------------
 */

proc (1) = Z_studtb(pred_act,pred,sigg,alld,predactb,predb,siggb);
local ii,NN,ZZ,TT;

NN=rows(pred_act);
TT=rows(alld);
ZZ={};
ii=1;
do while ii <= Unum;
  ZZ=ZZ|( (1/sqrt(NN))*sumc( ( (predactb .le UU[ii]) -
(1-(cdftc(((UU[ii]-predb)/siggb),3))) )^2
 - ( (1/TT)*sumc(( (alld[.,1] .le UU[ii]) - (1-(cdftc(((UU[ii]-pred)/sigg),3))) )^2)
) ) );
  ii=ii+1;
endo;

/* so one big row (Unum x 1) is outputed here with the Z piece for each individual u
element */

retp(ZZ);
endp;


/* ------------------------------------------------------------                */
/* PROCEDURE - Bootstrap - Block - Full Sample - bootstrap                     */
/* ------------------------------------------------------------                */

/* bootstrap data sets */
/* assume that all lval nums divide evenly in samples of T obs */
/* used below */

proc (1) = bb_full(dat1,lval);
local N,num_uns,undraw1,x1,ib;
```

```
N=rows(dat1);
num_uns=N/lval;

/* draw uniforms U[0,T-l+1] */

undraw1=round((N-lval)*rndu(num_uns,1));

x1={};
ib=1;
do while ib<=num_uns;
  x1=x1|dat1[undraw1[ib]+1:undraw1[ib]+lval,.];
  ib=ib+1;
endo;

retp(x1);
endp;


/* ---------------------------------------------------------- */
/* ---------------------------------------------------------- */
/* ^^^^^^^^^^^^^^^^^ MAIN PROGRAM ^^^^^^^^^^^^^^^^^^^^^^^^^^ */
/* ---------------------------------------------------------- */
/* ---------------------------------------------------------- */




/* ---------------------------------------------------------- */
/* ---------------------------------------------------------- */

/* DEFINE AND ORGANIZE DATA */

/* define dependent variable */

Y = datin[.,1];

/*
Y=trimr(ln(Y)-lagn(ln(Y),1),1,0);
*/

print "Y Data with " rows(Y) " Obs";
print Y;

/* create AR `p_ar' lags of Y */

/*
X={};
k=1;
do while k<=p_ar;
  X=X~lagn(Y,k);
  k=k+1;
endo;
X=trimr(X,p_ar,0);
Y=trimr(Y,p_ar,0);
*/

X=datin[.,2:cols(datin)];

/* there are 3 lags and then lagged unem in the X variable above */

print "Y,X Data";
print Y~X;
```

```
print "Final smpl is " rows(Y) "obs.";
print "*********************************************";
print "*********************************************";

/* put together exogenous variables */
/* NOTE: the highest lag taken when forming any Z variable must be <= p_ar */

/*

Zin1=datin[.,2];

Z1={};
Z1=lagn(Zin1,1);
Z1=trimr(Z1,p_ar,0);

Z=Z1;

*/

Z={};

/* make intercept */

inter=ones(rows(Y),1);

/* put together all explanatory data */

X=inter~X~Z;

/* ------------------------------------------------------------ */
/* ------------------------------------------------------------ */

/* ESTIMATION AND PREDICTION AND STATS */

/* carry out estimation and prediction, 1-step ahead */


/* I. First, put together data that are predicted */

pred_Y=Y[rows(Y)-P+1:rows(Y),.]; /* predictions made of these */

/* II. Second, est mods and construct P 1-step ahead preds for each mod */

/* model 1 */

{fY1,fe1,Yh1,eh1,bet1,s2hat1,alldat1,allb1,adjAA1}=pred(Y,X[.,1],P,roll_yn);

/* model 2 */

{fY2,fe2,Yh2,eh2,bet2,s2hat2,alldat2,allb2,adjAA2}=pred(Y,X[.,1 4],P,roll_yn);


/*  end prediction */

/* III. Calculate actual Z statistic pieces (indicator-F) for each U and each model
*/

{ZZ_n_m1}=Z_norm(pred_Y,fY1,sqrt(allb1[.,cols(allb1)]));

{ZZ_n_m2}=Z_norm(pred_Y,fY2,sqrt(allb2[.,cols(allb2)]));

{ZZ_t3_m1}=Z_studt(pred_Y,fY1,sqrt(allb1[.,cols(allb1)]));
```

```
{ZZ_t3_m2}=Z_studt(pred_Y,fY2,sqrt(allb2[.,cols(allb2)]));

print "means of Z components over U - mod n1,n2,t3 1, t3 2";
print meanc(ZZ_n_m1)~meanc(ZZ_n_m2)~meanc(ZZ_t3_m1)~meanc(ZZ_t3_m2);

/* IV. form actual stats, which are the benchmark - alternative combos */

ZP1=meanc(ZZ_n_m1 - ZZ_n_m2);
ZP2=meanc(ZZ_n_m1 - ZZ_t3_m1);
ZP3=meanc(ZZ_n_m1 - ZZ_t3_m2);

print "The Actual stats over which to choose the max i.e. the Z stat";
print (ZP1|ZP2|ZP3)';

ZP=maxc(ZP1|ZP2|ZP3);

/* end Z stat construction */

/* ------------------------------------------------------------ */
/* ------------------------------------------------------------ */

/* BOOTSTRAP CRITICAL VALUES */

ZPb={};
ZPb2={};

critout1={}; critout2={};

/* do a loop for different block lengths */

lvali=1;

do while lvali<=rows(ls1);

/* define lval */

lval=ls1[lvali];

/* now do a bunch of bootstrap simulations for this block length */

booti=1;
do while booti<=Bootnum;

/* I. Bootstrap the data Y,X */

{tempd}=bb_full((Y~X),lval);

bY=tempd[.,1];
bX=tempd[.,2:cols(tempd)];

/* Put together data that are predicted */

pred_Yb=bY[rows(bY)-P+1:rows(bY),.]; /* predictions made of these */

/* II. Second, est mods and construct P 1-step ahead preds for each mod */

/* Bootstrap estimator adjustment and boot statistic recentering */

/* model 1 */
{fY1b,fe1b,Yh1b,eh1b,bet1b,s2hat1b,alldat1b,allb1b}=predb(bY,bX[.,1],P,roll_yn,adjAA
1);
/* model 2 */
{fY2b,fe2b,Yh2b,eh2b,bet2b,s2hat2b,alldat2b,allb2b}=predb(bY,bX[.,1
```

```
4],P,roll_yn,adjAA2);

/* As above, but assume that R->oo faster than P, and so no adjustment, but have
recentering */

/* model 1 */
{fY1b2,fe1b2,Yh1b2,eh1b2,bet1b2,s2hat1b2,alldat1b2,allb1b2,adjAA1b}=pred(bY,bX[.,1],
P,roll_yn);
/* model 2 */
{fY2b2,fe2b2,Yh2b2,eh2b2,bet2b2,s2hat2b2,alldat2b2,allb2b2,adjAA2b}=pred(bY,bX[.,1
4],P,roll_yn);

/* III. Calculate boot Z statistic pieces (indicator-F) for each U and each model */

/* Bootstrap estimator adjustment and boot statistic recentering */


{ZZnm1b}=Z_normb(pred_Y,fY1,sqrt(allb1[.,cols(allb1)]),alldat1,pred_Yb,fY1b,sqrt(all
b1b[.,cols(allb1b)]));

{ZZnm2b}=Z_normb(pred_Y,fY2,sqrt(allb2[.,cols(allb2)]),alldat2,pred_Yb,fY2b,sqrt(all
b2b[.,cols(allb2b)]));

{ZZt3m1b}=Z_studtb(pred_Y,fY1,sqrt(allb1[.,cols(allb1)]),alldat1,pred_Yb,fY1b,sqrt(a
llb1b[.,cols(allb1b)]));

{ZZt3m2b}=Z_studtb(pred_Y,fY2,sqrt(allb2[.,cols(allb2)]),alldat2,pred_Yb,fY2b,sqrt(a
llb2b[.,cols(allb2b)]));

/* As above, but assume that R->oo faster than P, and so no adjustment, but have
recentering */


{ZZnm1b2}=Z_normb(pred_Y,fY1,sqrt(allb1[.,cols(allb1)]),alldat1,pred_Yb,fY1b2,sqrt(a
llb1b2[.,cols(allb1b2)]));

{ZZnm2b2}=Z_normb(pred_Y,fY2,sqrt(allb2[.,cols(allb2)]),alldat2,pred_Yb,fY2b2,sqrt(a
llb2b2[.,cols(allb2b2)]));

{ZZt3m1b2}=Z_studtb(pred_Y,fY1,sqrt(allb1[.,cols(allb1)]),alldat1,pred_Yb,fY1b2,sqrt
(allb1b2[.,cols(allb1b2)]));

{ZZt3m2b2}=Z_studtb(pred_Y,fY2,sqrt(allb2[.,cols(allb2)]),alldat2,pred_Yb,fY2b2,sqrt
(allb2b2[.,cols(allb2b2)]));

/* IV. form actual bootstrap stats, which are the benchmark - alternative combos */

/* Bootstrap estimator adjustment and boot statistic recentering */

ZP1b=meanc(ZZnm1b - ZZnm2b);
ZP2b=meanc(ZZnm1b - ZZt3m1b);
ZP3b=meanc(ZZnm1b - ZZt3m2b);

ZPb=ZPb|maxc(ZP1b|ZP2b|ZP3b);

/* As above, but assume that R->oo faster than P, and so no adjustment, but have
recentering */

ZP1b2=meanc(ZZnm1b2 - ZZnm2b2);
ZP2b2=meanc(ZZnm1b2 - ZZt3m1b2);
ZP3b2=meanc(ZZnm1b2 - ZZt3m2b2);

ZPb2=ZPb2|maxc(ZP1b2|ZP2b2|ZP3b2);
```

```
booti=booti+1;
endo;

/* end Z stat construction */

/* Construct Critical Values */

/* Bootstrap estimator adjustment and boot statistic recentering */

temp=sortc(ZPb,1);
cv50=temp[trunc(0.5*rows(temp))];
cv60=temp[trunc(0.6*rows(temp))];
cv70=temp[trunc(0.7*rows(temp))];
cv80=temp[trunc(0.8*rows(temp))];
cv90=temp[trunc(0.9*rows(temp))];
cr_ZPb=cv50|cv60|cv70|cv80|cv90;

/* As above, but assume that R->oo faster than P, and so no adjustment or
recentering */

temp=sortc(ZPb2,1);
cv50=temp[trunc(0.5*rows(temp))];
cv60=temp[trunc(0.6*rows(temp))];
cv70=temp[trunc(0.7*rows(temp))];
cv80=temp[trunc(0.8*rows(temp))];
cv90=temp[trunc(0.9*rows(temp))];
cr_ZPb2=cv50|cv60|cv70|cv80|cv90;

/* V. output */

print "============================";
print "Ex ante period = " P;
if roll_yn==1;
  print "Rolling Estimation";
else;
  print "Recursive Estimation";
endif;
print "Block length is " lval;
print "============================";

print "Z stat = " ZP;
print "Crit val, recentered:";
print cr_ZPb;

critout1=critout1~cr_ZPb;

print "Crit val, no adj or recentering:";
print cr_ZPb2;

critout2=critout2~cr_ZPb2;

/* end the crit val construction for a given block length */

lvali=lvali+1;
endo;

print "============================";
print "============================";
print "============================";

print "actual~for_m1~for_m2~for_m1_err~for_m2_err";
print pred_Y~fy1~fy2~fe1~fe2;
```

```
print "MEAN Sum Squared For Error";
print meanc(fe1^2~fe2^2)';
print "Actual Squared Fes";
print fe1^2~fe2^2~(fe1^2 .> fe2^2);

print "all crit vals for tables";
print critout1~critout2;

output off;
```