```
/* fcst.prc -- Forecasting Procs -- */

@ Version Date 12/18/2003 @

@ -- Functions -- @
fn flamcal(x,f) = (x'f)*(invpd(f'f));       @ LS lam given f @
fn ffcal(x,lam) = (x*lam)*invpd(lam'lam);   @ LS f given lam @

 @ Random Integer between 1 and n @
fn rndint(n) = 1 + floor(n*rndu(1,1));

@ -- Procedures -- @
proc(1) = transx(x,tcode);

/* Transform x
   Return Series with same dimension and corresponding dates
   Missing values where not calculated
   -- Tcodes:
            1 Level
            2 First Difference
            3 Second Difference
            4 Log-Level
            5 Log-First-Difference
            6 Log-Second-Difference
            7 Detrend Log Using 1-sided HP detrending for Monthly data
            8 Detrend Log Using 1-sided HP detrending for Quarterly data
            9 Detrend Level Using 1-sided HP detrending for Monthly data
            10 Detrend Level Using 1-sided HP detrending for Quarterly data

           16 Log-Second-Difference
           17 (1-L)(1-L^12)
           18 (1-L^3)(1-L^3) to logs
*/
local y, n, small, t1, relvarm, relvarq;
small=1.0e-06;
relvarm=.00000075;
relvarq=.000625;                 @ HP parameter
                                 .00000075 for monthly data
```

```
                                   .000625 for quarterly data, see Harvey/Jeager (1993), page 234 @
n=rows(x);
y=miss(zeros(n,1),0);           @ Y is now a series of missing values @

 if tcode .== 1;
  y=x;
 elseif tcode .== 2;
  y[2:n]=x[2:n]-x[1:n-1];
 elseif tcode .== 3;
  y[3:n]=x[3:n]-2*x[2:n-1]+x[1:n-2];
 elseif tcode .== 4;
  if minc(x) .< small; retp(miss(0,0)); endif;
  x=ln(x);
  y=x;
 elseif tcode .== 5;
  if minc(x) .< small;  retp(miss(0,0)); endif;
  x=ln(x);
  y[2:n]=x[2:n]-x[1:n-1];
 elseif tcode .== 6;
  if minc(x) .< small;  retp(miss(0,0)); endif;
  x=ln(x);
  y[3:n]=x[3:n]-2*x[2:n-1]+x[1:n-2];
 elseif tcode .== 7;
  if minc(x) .< small; retp(miss(0,0)); endif;
  x=ln(x);
  {y,t1}=detrend1(x,relvarm);
 elseif tcode .== 8;
  if minc(x) .< small; retp(miss(0,0)); endif;
  x=ln(x);
  {y,t1}=detrend1(x,relvarq);
 elseif tcode .== 9;
  {y,t1}=detrend1(x,relvarm);
 elseif tcode .== 10;
  {y,t1}=detrend1(x,relvarq);
 elseif tcode .== 16;
  if minc(x) .< small;  retp(miss(0,0)); endif;
  x=ln(x);
  y[3:n]=x[3:n]-2*x[2:n-1]+x[1:n-2];
```

```
 elseif tcode .== 17;
  if minc(x) .< small;  retp(miss(0,0)); endif;
  x=ln(x);
  y[14:n]=x[14:n]-x[13:n-1]-x[2:n-12]+x[1:n-13];
 elseif tcode .== 18;
  if minc(x) .< small;  retp(miss(0,0)); endif;
  x=ln(x);
  y[7:n]=x[7:n]-2*x[4:n-3]+x[1:n-6];
 else;
  retp(miss(0,0));
 endif;

retp(y);
endp;

@ ------------------------------------------------------- @
proc(1) = transxin(y,x,tcode);

/* Transform y series back into x

   -- Tcodes:
            1 Level
            2 First Difference
            3 Second Difference
            4 Log-Level
            5 Log-First-Difference
            6 Log-Second-Difference
*/
local xs, i, n;
n=rows(x);
xs=miss(zeros(n,1),0);           @ xs is now a series of missing values @

 if tcode .== 1;
  xs=y;
 elseif tcode .== 2;
  xs[1]=x[1];
  i=2; do while i<=n;
   if ismiss(xs[i-1]) .== 1;
```

```
   xs[i]=x[i];
  else;
   xs[i]=y[i]+xs[i-1];
  endif;
 i=i+1; endo;
elseif tcode .== 3;
 xs[1]=x[1];
 xs[2]=x[2];
 i=3; do while i<=n;
  if ismiss(xs[i-2:i-1]) .== 1;
   xs[i]=x[i];
   xs[i-1]=x[i-1];
  else;
   xs[i]=y[i]+2*xs[i-1]-xs[i-2];
  endif;
 i=i+1; endo;
elseif tcode .== 4;
 y=exp(y);
 xs=y;
elseif tcode .== 5;
 xs[1]=x[1];
 i=2; do while i<=n;
  if ismiss(xs[i-1]) .== 1;
   xs[i]=x[i];
  else;
   xs[i]=xs[i-1]*exp(y[i]);
  endif;
 i=i+1; endo;
elseif tcode .== 6;
 xs[1]=x[1];
 xs[2]=x[2];
 i=3; do while i<=n;
  if ismiss(xs[i-2:i-1]) .== 1;
   xs[i]=x[i];
   xs[i-1]=x[i-1];
  else;
   xs[i]=(xs[i-1]^2)*exp(y[i])/xs[i-2];
  endif;
```

```
  i=i+1; endo;
 endif;

retp(xs);
endp;

@ ------------------------------------------------------- @
proc(1) = exfac_for(xx,k);

/* Extract k factors from the TxT matrix XX
   This uses a Fortran subroutine for the eigenvector
   calculation.  The DLL library for this is given
   in the main program
*/

local t, xeval, xevec, f;

t=rows(xx);
xeval=zeros(k,1);
xevec=zeros(k,t);
dllcall mevesf(t,k,xx,xeval,xevec);
f=xevec';
retp(f);
endp;
@ ------------------------------------------------------- @
proc(1) = exfac_gss(xx,k);

/* Extract k factors from the TxT matrix XX
   This uses gauss to do the eigen calculations.
*/

local va,ve, f;

{va,ve}=eighv(xx);
ve=(rev(ve'))';
f=ve[.,1:k];            @ Factors  @

retp(f);
```

```
endp;

@ ------------------------------------------------------- @
proc(4) = pc_factor(x,k);

/* Compute principal components estimates of factor model

   Input:
    x = txn matrix
    k = number of factors

    Model

    x(it) = lam(i)'f(t) + u(it)

    or

    x = f*lam' + u


    Output:

    f = txk matrix of factors
    lam = n*k matrix of factor loadings
    eigval = kx1 vector of eigenvalues of x*x' (or x'x) ...
    ssr = sum of squared u's

    Normalization:
     f is normalized so that each column has std dev = 1, thus F'F = t*I(k)

    Calculation note:
     Calculations are speeded by using the smaller of x*x' or x'x
*/

local t, n, xx, va, ve, lam, fac, sfac, ssr, eigval;

t=rows(x);
n=cols(x);
```

```
if k .> 0;
if n .< t;
 xx=x'x;
 {va,ve}=eighv(xx);
 ve=(rev(ve'))';
 va=rev(va);
 eigval=va[1:k];
 lam=ve[.,1:k];
 fac=x*lam;
else;
 xx=x*x';
 {va,ve}=eighv(xx);
 ve=(rev(ve'))';
 va=rev(va);
 eigval=va[1:k];
 fac=ve[.,1:k];
 lam=x'fac;
endif;

@ Normalize @
 sfac=sqrt(meanc(fac.^2));
 fac=fac./sfac';
 lam=(x'fac)/t;  @ Note fac'fac = t @
 ssr=sumc(va)-sumc(eigval);
else;
 ssr=sumc(sumc(x.^2));
 lam=miss(0,0);
 eigval=miss(0,0);
 fac=miss(0,0);
endif;

retp(fac,lam,eigval,ssr);
endp;


@ --------------------------------------------------------------------- @
proc(1) = yfcst(x,tcode,nph);
```

```
@ -- Transform series to form series to be forecast --

     Input:
     x == raw series
     tcode == transformation code
     nph == forecast horizon

  -- Tcodes:
            1 Level
            2 First Difference
            3 Second Difference
            4 Log-Level
            5 Log-First-Difference
            6 Log-Second-Difference
           16 Log(1-L)(1-L^12) forecast deviation
           17 Log(1-L)(1-L^12) forecast deviation (same as 16)
           18 Log (1-L)(1-L^4)

 -- Important Note -- This produces series that is shifted forward nph
    ahead

@

local yf, t, y, n, small;
small=1.0e-06;
n=rows(x);
yf=miss(zeros(n,1),0);          @ Y is now a series of missing values @


@ -- Logs or Levels as appropriate -- @
if tcode .<= 3;
   y=x;
elseif (tcode .>= 4) .and (tcode .<= 6);
   if minc(x) .< small; retp(miss(0,0)); endif;
   y=ln(x);
elseif (tcode .>= 16) .and (tcode .<= 18);
   if minc(x) .< small; retp(miss(0,0)); endif;
```

```
   y=ln(x);
else;
 "Invalid Transformation Code in yfcst";
 "Tcode = ";;tcode;
 "Processing Stops";stop;
endif;

@ -- Transform and Shift Forward -- @
if (tcode .== 1) .or (tcode .== 4);
  yf[1:rows(y)-nph]=y[1+nph:rows(y)];
elseif (tcode .== 2) .or (tcode .== 5);
  yf[1:rows(y)-nph]=y[1+nph:rows(y)]-y[1:rows(y)-nph];
elseif (tcode .== 3) .or (tcode .== 6);
  yf[2:rows(y)-nph]=(y[2+nph:rows(y)]-y[2:rows(y)-nph])
                   - nph*(y[2:rows(y)-nph]-y[1:rows(y)-nph-1]);
elseif (tcode .== 16) .or (tcode .== 17);
  yf[13:rows(y)-nph]=(y[13+nph:rows(y)]-y[13+nph-12:rows(y)-12])
                   - (y[13:rows(y)-nph]-y[13-12:rows(y)-nph-12]);
elseif (tcode .== 18);
  yf[5:rows(y)-nph]=(y[5+nph:rows(y)]-y[5+nph-4:rows(y)-4])
                   - (y[5:rows(y)-nph]-y[1:rows(y)-nph-4]);
endif;

retp(yf);
endp;

@ --------------------------------------------------------------------- @
proc(1) = yfcsta(y,tcode,nph);

@ -- Transform series to form series to be forecast
     Note input is transformed series instead of raw series
     This is useful when transformed series contains missing
     values because outlier ajustments --

     Input:
     y == transformed series
     tcode == transformation code
     nph == forecast horizon
```

```
  -- Tcodes:
            1 Level
            2 First Difference
            3 Second Difference
            4 Log-Level
            5 Log-First-Difference
            6 Log-Second-Difference

 -- Important Note -- This produces series that is shifted forward nph
    ahead

@
local yf, t, n, small;
small=1.0e-06;
n=rows(y);
yf=miss(zeros(n,1),0);          @ Y is now a series of missing values @


@ -- Transform and Shift Forward -- @
if (tcode .== 1) .or (tcode .== 4);
  yf[1:rows(y)-nph]=y[1+nph:rows(y)];
elseif (tcode .== 2) .or (tcode .== 5);
  for t (1, rows(y)-nph,1);
   yf[t]=sumc(y[t+1:t+nph]);
  endfor;
elseif (tcode .== 3) .or (tcode .== 6);
  for t (1, rows(y)-nph,1);
   yf[t]=sumc(cumsumc(y[t+1:t+nph]));
  endfor;
else;
"Invalid Transformation Code in yfcst";
 "Tcode = ";;tcode;
 "Processing Stops";stop;
endif;

retp(yf);
endp;
```

```
@ ------------------------------------------------------------------ @
proc(2)=icmod(y,x1,lmeth);

/* -- Choose IC Model order over columns of
      X1 for linear regression of y onto x1
      Orders are from 0-Cols(x1)

      lmeth=1:  AIC
      lmeth=2:  BIC

      Return is:
       nox1:    Order of X1
       crit:    Value of Criterion at maximum

*/
local big, biccrit, yy, i, x, xxi, ee, ir, x1o, pfac, crit;
big=999999999;
biccrit=big*ones(cols(x1)+1,1);
yy=y'y;
if lmeth .== 1; pfac=2/rows(y); endif;
if lmeth .== 2; pfac=ln(rows(y))/rows(y); endif;

ee=yy;
biccrit[1,1]=ln(det(ee));

i=1; do while i <= cols(x1);
  x=x1[.,1:i];
  xxi=safe_xpx(x);
  if ismiss(xxi) .==1;
   ee=yy;
  else;
   ee=yy-y'x*xxi*x'y;
  endif;
  biccrit[i+1,1]=ln(det(ee)) + cols(y)*cols(x)*pfac;
i=i+1; endo;
ir=minindc(biccrit);
ir=ir-1;
```

```
crit=minc(biccrit);
x1o=ir;

retp(x1o,crit);
endp;


@ -------------------------------------------------------------------- @
proc(2)=icmod0(y,x1,lmeth);

/* -- Choose IC Model order over columns of
      X1 for linear regression of y onto z, and x1
      Orders are from 1-Cols(x1)

      lmeth=1:  AIC
      lmeth=2:  BIC

      Return is:
       nox1:    Order of X1
       crit:    Value of Criterion at maximum

*/
local big, biccrit, yy, i, x, xxi, ee, ir, x1o, pfac, crit;
big=999999999;
biccrit=big*ones(cols(x1),1);
yy=y'y;
if lmeth .== 1; pfac=2/rows(y); endif;
if lmeth .== 2; pfac=ln(rows(y))/rows(y); endif;

i=1; do while i <= cols(x1);
  x=x1[.,1:i];
  xxi=safe_xpx(x);
  if ismiss(xxi) .==1;
   ee=yy;
  else;
   ee=yy-y'x*xxi*x'y;
  endif;
  biccrit[i,1]=ln(det(ee)) + cols(y)*cols(x)*pfac;
```

```
i=i+1; endo;
ir=minindc(biccrit);
crit=minc(biccrit);
x1o=ir;

retp(x1o,crit);
endp;

@ -------------------------------------------------------------------- @
proc(2)=icmod1(y,z,x1,lmeth);

/* -- Choose IC Model order over columns of
      X1 for linear regression of y onto z, and x1
      Orders are from 0-Cols(x1)

      lmeth=1:  AIC
      lmeth=2:  BIC

      Return is:
       nox1:    Order of X1
       crit:    Value of Criterion at maximum

       if x is a scalar, then crit value for regression of y onto z
       is returned
*/
local big, biccrit, yy, i, x, xxi, ee, ir, x1o, pfac, crit;
big=999999999;
biccrit=big*ones(cols(x1)+1,1);
yy=y'y;
if lmeth .== 1; pfac=2/rows(y); endif;
if lmeth .== 2; pfac=ln(rows(y))/rows(y); endif;

i=0; do while i <= cols(x1);
  x=z;
  if (i .> 0) .and (rows(x1) .> 1); x=x~x1[.,1:i]; endif;
  xxi=safe_xpx(x);
  if ismiss(xxi) .==1;
   ee=yy;
```

```
  else;
   ee=yy-y'x*xxi*x'y;
  endif;
  biccrit[i+1,1]=ln(det(ee)) + cols(y)*cols(x)*pfac;
i=i+1; endo;
ir=minindc(biccrit);
crit=minc(biccrit);
x1o=ir-1;

retp(x1o,crit);
endp;

@ --------------------------------------------------------------------- @
proc(3)=icmod2(y,z,x1,x2,lmeth);

/* -- Choose IC Model order over columns of
      X1 and X2 for linear regression of y onto z, x1 and X2
      Choice done separately for each column of X1 and X2
      Orders are from 0-Cols(x1) and 0-Cols(x2)

      lmeth=1:  AIC
      lmeth=2:  BIC

      Return is:
       nox1:    Order of X1
       nox2:    Order of X2
       crit:    Best Value of Criterion
*/
local big, biccrit, yy, i, j, x, xxi, ee, ir,xr, jr, x1o, x2o, pfac, crit;
big=999999999;
biccrit=big*ones(cols(x1)+1,cols(x2)+1);
yy=y'y;
if lmeth .== 1; pfac=2/rows(y); endif;
if lmeth .== 2; pfac=ln(rows(y))/rows(y); endif;

i=0; do while i <= cols(x1);
 j=0; do while j <= cols(x2);
  x=z;
```

```
  if i .> 0; x=x~x1[.,1:i]; endif;
  if j .> 0; x=x~x2[.,1:j]; endif;
  xxi=safe_xpx(x);
  if ismiss(xxi) .==1;
   ee=yy;
  else;
   ee=yy-y'x*xxi*x'y;
  endif;
  biccrit[i+1,j+1]=ln(det(ee)) + cols(y)*cols(x)*pfac;
 j=j+1; endo;
i=i+1; endo;
ir=minindc(biccrit);
xr=minc(biccrit);
jr=minindc(xr);
x1o=ir[jr]-1;
x2o=jr-1;

crit=minc(minc(biccrit));

retp(x1o,x2o,crit);

endp;
@ -------------------------------------------------------------------- @
proc(2)=icmod3(y,z,x1,x2,lmeth);

/* -- Choose IC Model order over columns of
      X1 and X2 for linear regression of y onto z, X1 and X2
      Choice done jointly each column of X1 and X2
      Thus X1 and X2 must have same number of columns

      lmeth=1:  AIC
      lmeth=2:  BIC

      Return is:
       nox1:    Order of X1
       crit:    Best Value of Criterion
*/
local big, biccrit, yy, i, x, xxi, ee, ir, x1o, pfac, crit;
```

```
big=999999999;
biccrit=big*ones(cols(x1)+1,1);
yy=y'y;
if lmeth .== 1; pfac=2/rows(y); endif;
if lmeth .== 2; pfac=ln(rows(y))/rows(y); endif;

if cols(x1) ./= cols(x2);
 "In ICMOD3, cols(x1) ./= cols(x2), processing stops";
endif;


i=0; do while i <= cols(x1);
  x=z;
  if (i .> 0) .and (rows(x1) .> 1); x=x~x1[.,1:i]~x2[.,1:i]; endif;
  xxi=safe_xpx(x);
  if ismiss(xxi) .==1;
   ee=yy;
  else;
   ee=yy-y'x*xxi*x'y;
  endif;
  biccrit[i+1,1]=ln(det(ee)) + cols(y)*cols(x)*pfac;
i=i+1; endo;
ir=minindc(biccrit);
crit=minc(biccrit);
x1o=ir-1;

retp(x1o,crit);

endp;
@ -------------------------------------------------------------------- @
proc(2)=icvar1(y,z,x1,lmeth);

/* -- Choose IC Lags in VAR order over columns of
      X1 for linear regression of y onto z, and x1
      in usual applications
      z contains the constant term
      x1 contains lags (ordered by lags)
```

```
      lmeth=1:  AIC
      lmeth=2:  BIC

      Return is:
       nox1:    Order of X1
       crit:    Value of Criterion at maximum

       if x is a scalar, then crit value for regression of y onto z
       is returned
*/
local big, biccrit, yy, i, x, xxi, ee, ir, x1o, pfac, crit, nlagmax, nreg;
big=999999999;
biccrit=big*ones(cols(x1)+1,1);
yy=y'y;
if lmeth .== 1; pfac=2/rows(y); endif;
if lmeth .== 2; pfac=ln(rows(y))/rows(y); endif;

nlagmax=cols(x1)/cols(y);

i=0; do while i <= nlagmax;
  nreg=i*cols(y);
  x=z;
  if (i .> 0) .and (rows(x1) .> 1);
    x=x~x1[.,1:nreg];
  endif;
  xxi=safe_xpx(x);
  if ismiss(xxi) .==1;
   ee=yy;
  else;
   ee=yy-y'x*xxi*x'y;
  endif;
  biccrit[i+1,1]=ln(det(ee)) + cols(y)*cols(x)*pfac;
i=i+1; endo;
ir=minindc(biccrit);
crit=minc(biccrit);
x1o=ir-1;

retp(x1o,crit);
```

```
endp;

@ ---------------------------------------------------------------  @
proc(1)=bicmod(y,x,nxmin,nxmax);

/* -- Choose BIC Model order over columns of X for linear regression
      of y onto x
      Orders chosen from nxmin to nxmax
*/
local biccrit, yy, i, x1, ee, bico, big, oldval, xxi;
big=999999999;
biccrit=big*ones(cols(x),1);
yy=y'y;
oldval=trapchk(1);
trap 1,1;
i=nxmin; do while i <= nxmax;
 x1=x[.,1:i];
 xxi=invpd(x1'x1);
 if scalerr(xxi);
  biccrit[i]=big;
  "Inversion Problem in BIC - i";;i;
 else;
  ee=yy-y'x1*xxi*x1'y;
  biccrit[i]=ln(det(ee)) + cols(y)*i*ln(rows(y))/rows(y);
 endif;
i=i+1; endo;
trap oldval,1;

bico=minindc(biccrit);

retp(bico);
endp;

@ ---------------------------------------------------------- @
proc(1) = safe_xpx(x);

/* -- Safe Inverse of X'X
      INVPD unless error -- then pinv
```

```
*/
local xxi, oldval, xx;

xx=x'x;
oldval=trapchk(1);
trap 1,1;
 xxi=invpd(xx);
 if scalerr(xxi);
  xxi=pinv(xx);
  @ "Inversion Problem Using Pinv"; @
 endif;
trap oldval,1;
retp(xxi);
endp;

@ ----------------------------------------------------------- @
proc(2)=gname(s);
@ -- Extract name and a vector of scalar codes from string
     Everything must be separated by spaces (arbitrary number)
     examples

     (1)  name      (tcode is returned as missing value)
     (2)  name 1 2 3  (three codes returned)

-- @
local sname, slen, s1, y, tcodest, tcode, tmp;

 @ Eliminate leading blanks @
 slen=strlen(s);
 y=strindx(s," ",1);
 do while y .== 1;
  slen=slen-1;
  s=strsect(s,2,slen);
  y=strindx(s," ",1);
 endo;

 @ Extract Name @
 s1=slen;
```

```
 if y./= 0; s1=y-1; endif;
 sname=strsect(s,1,s1);

 @ Find Transformation Code @
 tcode=miss(0,0);
 s1=strlen(sname);
 slen=slen-s1;
 if slen .== 0; retp(sname,tcode); endif;
 do while slen .> 0;
   s1=s1+1;
   s=strsect(s,s1,slen);
   @ Eliminate leading blanks @
   slen=strlen(s);
   y=strindx(s," ",1);
   do while y .== 1;
     slen=slen-1;
     if slen .== 0; retp(sname,tcode); endif;
     s=strsect(s,2,slen);
     y=strindx(s," ",1);
    endo;
    @ Extract code @
    s1=slen;
    if y./= 0; s1=y-1; endif;
    tcodest=strsect(s,1,s1);
    tmp=stof(tcodest);           @ Transformation Code @
    if ismiss(tcode) .== 1;
     tcode=tmp;
    else;
     tcode=tcode|tmp;
    endif;
    slen=slen-s1;
    if slen .== 0; retp(sname,tcode); endif;
  endo;

retp(sname,tcode);
endp;

@ ----------------------------------------------------------- @
```

```
proc(2) = estep(y,f);

@ -- Regress non-missing values of y onto f
     Replace Missing Values with fitted value

-- @
local missc, y1, z, f1, lam, e, ssr, yhat, iy, iny;

missc=1e+32;  @ -- A missing value indicator -- @

@ -- Non-Missing Values  -- @
z=packr(y~f);
y1=z[.,1];
f1=z[.,2:cols(z)];

@ -- Estimate Lambda and compute SSR -- @
lam=flamcal(y1,f1);
e=y1-(f1*lam');
ssr=e'e;


@ -- Indicators for Missing and non-missing values -- @
y=missrv(y,missc);
iy = (y .== missc);
iny = (y ./= missc);


@ -- Replace Missing Values with Fitted Values -- @
yhat=f*lam';
y=(iny.*y) + (iy.*yhat);

retp(y,ssr);
endp;

@ -------------------------------------------------- @
proc(1) = standmv(y);
```

```
@ -- Standardize a series that may contain missing values -- @
local y1, m1, s1;

@ -- Standardize y -- @
y1=packr(y);
m1=meanc(y1);
s1=stdc(y1);
y=(y-m1')./s1;

retp(y);
endp;

@ --------------------------------------------------------------------- @
proc(2)=cicmod1(y,z,x1,n,omega);

/* -- Choose CIC Model order over columns of
      X1 for linear regression of y onto z, and x1
      Orders are from 0-Cols(x1)

      n == number of cross sections used to estimate factor
      omega == scale factor in CIC

      Return is:
       nox1:    Order of X1
       crit:    Value of Criterion at maximum

       if x is a scalar, then crit value for regression of y onto z
       is returned
*/
local big, biccrit, yy, i, x, xxi, ee, ir, x1o, pfac, crit,
      eps, d1, d2, dnt, t;
big=999999999;
biccrit=big*ones(cols(x1)+1,1);
yy=y'y;

t=rows(y);
eps=.01;
d1=sqrt(n)/(t^(1+eps));
```

```
d2=t^(1-eps);
dnt=minc(d1|d2);
pfac=omega*ln(t)/dnt;

i=0; do while i <= cols(x1);
  x=z;
  if (i .> 0) .and (rows(x1) .> 1); x=x~x1[.,1:i]; endif;
  xxi=safe_xpx(x);
  if ismiss(xxi) .==1;
   ee=yy;
  else;
   ee=yy-y'x*xxi*x'y;
  endif;
  biccrit[i+1,1]=ln(ee) + cols(x)*pfac;
i=i+1; endo;
ir=minindc(biccrit);
crit=minc(biccrit);
x1o=ir-1;

retp(x1o,crit);
endp;

@ ---------------------------------------------------------------------- @
proc(1) = sadet(yy,tsize);

/* -- Tests for seasonality -- monthly

     Input:
      y = Data series
      tsize = size of test

     Output:
      san = 0 no rejection
            1 rejection
            2 -- not enough obs

*/
local cvarsa, idm, ii, sdm, temp, san, yas, bsa, err, s2hat, vb, wstat, pv;
```

```
cvarsa=ones(rows(yy),1);
idm=floor((rows(yy)/12))+1;
ii=1; do while ii<=idm;
  if ii==1; sdm = eye(12);
   else; sdm = sdm | eye (12);
  endif;
ii=ii+1; endo;
sdm=cvarsa[1:rows(yy)]~sdm[1:rows(yy),2:12]; @ matrix of regressors @
temp=packr(yy~sdm);
if rows(temp) .< 48;
 san=2;
 retp(san);
endif;
yas=temp[.,1];
sdm=temp[.,2:cols(temp)];
bsa=invpd(sdm'sdm)*(sdm'yas);
err=yas-sdm*bsa;
s2hat=sumc(err.^2)/(rows(yas)-cols(sdm));
vb=s2hat*invpd(sdm'sdm);
wstat= bsa[2:12,.]'*invpd(vb[2:12,2:12])*bsa[2:12,.];
pv=cdfchic(wstat,11);
san = (pv <= tsize);

retp(san);
endp;
@ --------------------------------------------------------------------- @
proc(1) = sadetq(yy,tsize);

/* -- Tests for seasonality -- quarterly

     Input:
      y = Data series
      tsize = size of test

     Output:
      san = 0 no rejection
            1 rejection
```

```
            2 -- not enough obs

*/
local cvarsa, idm, ii, sdm, temp, san, yas, bsa, err, s2hat, vb, wstat, pv;

cvarsa=ones(rows(yy),1);
idm=floor((rows(yy)/4))+1;
ii=1; do while ii<=idm;
  if ii==1; sdm = eye(4);
   else; sdm = sdm | eye (4);
  endif;
ii=ii+1; endo;
sdm=cvarsa[1:rows(yy)]~sdm[1:rows(yy),2:4]; @ matrix of regressors @
temp=packr(yy~sdm);
if rows(temp) .< 20;
 san=2;
 retp(san);
endif;
yas=temp[.,1];
sdm=temp[.,2:cols(temp)];
bsa=invpd(sdm'sdm)*(sdm'yas);
err=yas-sdm*bsa;
s2hat=sumc(err.^2)/(rows(yas)-cols(sdm));
vb=s2hat*invpd(sdm'sdm);
wstat= bsa[2:4,.]'*invpd(vb[2:4,2:4])*bsa[2:4,.];
pv=cdfchic(wstat,3);
san = (pv <= tsize);

retp(san);
endp;
@ -------------------------------------------------------------------- @
proc(1) = seasqreg(y);

/* -- Seasonal Adjust Using Dummy Variable Regression

*/
local cvarsa, idm, ii, sdm, ysa, b;
```

```
ysa=miss(zeros(rows(y),1),0);
idm=floor((rows(y)/4))+1;
ii=1; do while ii<=idm;
  if ii==1; sdm = eye(4);
   else; sdm = sdm | eye (4);
  endif;
ii=ii+1; endo;
sdm=sdm[1:rows(y),.]; @ matrix of regressors @

b=invpd(sdm'sdm)*(sdm'y);
ysa=y-sdm*b;
ysa=ysa+(meanc(b))*ones(rows(y),1);

retp(ysa);
endp;
@ -------------------------------------------------------------------@
proc(1)=padar(y,n,arvec);

/* -- Pad Data series y out using AR Forecasts and Backcasts
      y -- series to be padded
      n -- number of terms to pad forward and backward
      arvec -- vector of AR lags
              if lags 1,3 and 6 are needed, then ARVEC=1|3|6, etc.
*/
local w, x, i, beta, bols, nar, v, forc, ypad;

nar=maxc(arvec);

@ Pad out future @
w=y[nar+1:rows(y)];
x=ones(rows(w),1);
i=1; do while i<=rows(arvec);
 x=x~y[nar+1-arvec[i]:rows(y)-arvec[i]];
i=i+1; endo;
bols=invpd(x'x)*(x'w);
beta=zeros(1+nar,1);
beta[1]=bols[1];
i=1; do while i<=rows(arvec);
```

```
 beta[1+arvec[i]]=bols[i+1];
i=i+1; endo;
v=rev(y[rows(y)-nar+1:rows(y)]);
forc=zeros(n,1);
i=1; do while i <= n;
 forc[i]=beta'(1|v);
 v[2:rows(v)]=v[1:rows(v)-1];
 v[1]=forc[i];
i=i+1; endo;
ypad=y|forc;


@ Pad out past, by reversing series @
y=rev(y);
w=y[nar+1:rows(y)];
x=ones(rows(w),1);
i=1; do while i<=rows(arvec);
 x=x~y[nar+1-arvec[i]:rows(y)-arvec[i]];
i=i+1; endo;
bols=invpd(x'x)*(x'w);
beta=zeros(1+nar,1);
beta[1]=bols[1];
i=1; do while i<=rows(arvec);
 beta[1+arvec[i]]=bols[i+1];
i=i+1; endo;
v=rev(y[rows(y)-nar+1:rows(y)]);
forc=zeros(n,1);
i=1; do while i <= n;
 forc[i]=beta'(1|v);
 v[2:rows(v)]=v[1:rows(v)-1];
 v[1]=forc[i];
i=i+1; endo;
forc=rev(forc);
ypad=forc|ypad;

retp(ypad);
endp;
@ -----------------------------------------------------------------@
```

```
proc(1)=x11arq(y);

/* X11ARq.prc, mww, 6/29/00
   Carry out seasonal adjustment using QUARTERLY X11 AR
   This applies the linear version of X11 to the
   series y, after it has been padded out with forecasts
   and backcasts constructed from an estimated AR model
   The current AR model is an AR(5).
   This can be changed by the vector ARVEC below.

 */

local n, arvec, ypad, x11, ysa;

if ismiss(y);
 "Y contains missing values in X11ARQ";
 "Proc not implemented for missing values";
 "Vector of missing values is returned";
 ysa=miss(zeros(rows(y),1),0);
 retp(ysa);
endif;

@ -- Pad Series -- @
n=28;
arvec=1|2|3|4|5;
ypad=padar(y,n,arvec);

@ -- X11 Filter -- @
x11=x11filtq;

@ -- Construct Seasonally Adjusted Series -- @
ysa=zeros(rows(y),1);
for i (1,rows(y),1);
 ysa[i]=ypad[i:i+2*n]'x11;
endfor;

retp(ysa);
endp;
```

```
@ --------------------------------------------------------------- @
proc(1)=x11filtq;

/*
   x11filtq.prc, mww and jpl 6/26/00
   Compute X11 Filter for quarterly data

   Follow the steps in

      Larocque, Guy "Analyse d'une methode de desaisonnalisation:
          le programme X11 du US Bureau of Census, version trimestrielle",
          Annale de l'INSEE, n.28, 1977


  This program is based on a Mark Watson's program for monthly data.
  The latter included the following notice

  -- Follow 8 Steps in Watson's JBES discussion of Ghysels, Granger,
Siklos (Which is taken from Wallis's 1974 JASA paper

*/

local a1,a2,s1,temp,a3,a4,a5,a6,h,temp1,mid,x11,s3,a7,a8;

@ Step 1: TC1 = a1(L)x(t) -- @
  a1=zeros(5,1);
  a1[1]=1/8;
  a1[5]=1/8;
  a1[2:4]=1/4*ones(3,1);

@ Step 2: SI1=x-TC1=a2(L)x @
  a2=-a1;
  a2[3]=1+a2[3];

@ Step 3: S1=S1(L)SI1=a3(L)x @
  s1=zeros(17,1);
  s1[1]=1/9;
  s1[5]=2/9;
```

```
  s1[9]=3/9;
  s1[13]=2/9;
  s1[17]=1/9;

  temp=rows(s1)-rows(a2);
  temp=temp/2;
  temp=zeros(temp,1);
  temp=temp|a2|temp;

  a3=polymult(s1,temp);
  temp=a3.==0;
  a3=delif(a3,temp);


@ Step 4: S2=a2(L)S1=a4(L)x @
  temp=rows(a3)-rows(a2);
  temp=temp/2;
  temp=zeros(temp,1);
  temp=temp|a2|temp;
  a4=polymult(a3,temp);
  temp=a4.==0;
  a4=delif(a4,temp);

@ Step 5: TC2=H(L)(x-S2)=a5(L)x @
  h=zeros(5,1);
  h[1]=-.073;
  h[2]=.294;
  h[3]=.558;
  h[4:5]=rev(h[1:2]);

  temp1=-a4;
  mid=rows(temp1)-1;
  mid=mid/2;
  mid=mid+1;
  temp1[mid]=1+temp1[mid];

  temp=rows(temp1)-rows(h);
  temp=temp/2;
```

```
  temp=zeros(temp,1);
  temp=temp|h|temp;
  a5=polymult(temp1,temp);
  temp=a5.==0;
  a5=delif(a5,temp);


@ Step 6: S3=S3(L)(x-TC2)=a6(L)x @
  temp1=-a5;
  mid=rows(temp1)-1;
  mid=mid/2;
  mid=mid+1;
  temp1[mid]=1+temp1[mid];
  s3=zeros(25,1);
  s3[1]=1/15;
  s3[5]=2/15;
  s3[9]=3/15;
  s3[13]=3/15;
  s3[17]=3/15;
  s3[21]=2/15;
  s3[25]=1/15;
  temp=rows(temp1)-rows(s3);
  temp=temp/2;
  temp=zeros(temp,1);
  temp=temp|s3|temp;
  a6=polymult(temp1,temp);
  temp=a6.==0;
  a6=delif(a6,temp);

@ Step 7: S4=a2(L)S3=a7(L)x @
  temp=rows(a6)-rows(a2);
  temp=temp/2;
  temp=zeros(temp,1);
  temp=temp|a2|temp;
  a7=polymult(a6,temp);
  temp=a7.==0;
  a7=delif(a7,temp);
```

```
@ Step 8: XSA=x-S4=a8x @
  a8=-a7;
  mid=rows(a8)-1;
  mid=mid/2;
  mid=mid+1;
  a8[mid]=1+a8[mid];
  temp=a8.==0;
  x11=delif(a8,temp);

retp(x11);
endp;

@ --------------------------------------------------------------  @

proc(1)=x11ar(y);

/* X11AR.prc, mww, 3/29/00
   Carry out seasonal adjustment using X11 AR
   This applies the linear version of X11 to the
   series y, after it has been padded out with forecasts
   and backcasts constructed from an estimated AR model
   The current AR model is an AR(3) with lags 12 and 13
   added.  This can be changed by the vector ARVEC below.

 */

local n, arvec, ypad, x11, ysa;

if ismiss(y);
 "Y contains missing values in X11AR";
 "Proc not implemented for missing values";
 "Vector of missing values is returned";
 ysa=miss(zeros(rows(y),1),0);
 retp(ysa);
endif;

@ -- Pad Series -- @
n=84;
```

```
arvec=1|2|3|12|13;
ypad=padar(y,n,arvec);

@ -- X11 Filter -- @
x11=x11filt;

@ -- Construct Seasonally Adjusted Series -- @
ysa=zeros(rows(y),1);
for i (1,rows(y),1);
 ysa[i]=ypad[i:i+2*n]'x11;
endfor;

retp(ysa);
endp;
@ -----------------------------------------------------------------@

proc(1)=x11filt;

local a1,a2,s1,temp,a3,a4,a5,a6,h,temp1,mid,x11,s3,a7,a8;

/* x11filt.prc, mww 3/29/00
   Compute X11 Filter
*/

@ -- Follow 8 Steps in Watson's JBES discussion of Ghysels, Granger, Siklos
     (Which is taken from Wallis's 1974 JASA paper -- @

@ Step 1: TC1 = a1(L)x(t) -- @
  a1=zeros(13,1);
  a1[1]=1/24;
  a1[13]=1/24;
  a1[2:12]=1/12*ones(11,1);

@ Step 2: SI1=x-TC1=a2(L)x @
  a2=-a1;
  a2[7]=1+a2[7];

@ Step 3: S1=S1(L)SI1=a3(L)x @
```

```
  s1=zeros(49,1);
  s1[1]=1/9;
  s1[13]=2/9;
  s1[25]=3/9;
  s1[37]=2/9;
  s1[49]=1/9;

  temp=rows(s1)-rows(a2);
  temp=temp/2;
  temp=zeros(temp,1);
  temp=temp|a2|temp;

  a3=polymult(s1,temp);
  temp=a3.==0;
  a3=delif(a3,temp);


@ Step 4: S2=a2(L)S1=a4(L)x @
  temp=rows(a3)-rows(a2);
  temp=temp/2;
  temp=zeros(temp,1);
  temp=temp|a2|temp;
  a4=polymult(a3,temp);
  temp=a4.==0;
  a4=delif(a4,temp);

@ Step 5: TC2=H(L)(x-S2)=a5(L)x @
  h=zeros(13,1);
  h[1]=-.0194;
  h[2]=-.0279;
  h[3]=0;
  h[4]=.0655;
  h[5]=.1474;
  h[6]=.2143;
  h[7]=.2402;
  h[8:13]=rev(h[1:6]);

  temp1=-a4;
```

```
  mid=rows(temp1)-1;
  mid=mid/2;
  mid=mid+1;
  temp1[mid]=1+temp1[mid];

  temp=rows(temp1)-rows(h);
  temp=temp/2;
  temp=zeros(temp,1);
  temp=temp|h|temp;
  a5=polymult(temp1,temp);
  temp=a5.==0;
  a5=delif(a5,temp);

@ Step 6: S3=S3(L)(x-TC2)=a6(L)x @
  temp1=-a5;
  mid=rows(temp1)-1;
  mid=mid/2;
  mid=mid+1;
  temp1[mid]=1+temp1[mid];
  s3=zeros(73,1);
  s3[1]=1/15;
  s3[13]=2/15;
  s3[25]=3/15;
  s3[37]=3/15;
  s3[49]=3/15;
  s3[61]=2/15;
  s3[73]=1/15;
  temp=rows(temp1)-rows(s3);
  temp=temp/2;
  temp=zeros(temp,1);
  temp=temp|s3|temp;
  a6=polymult(temp1,temp);
  temp=a6.==0;
  a6=delif(a6,temp);

@ Step 7: S4=a2(L)S3=a7(L)x @
  temp=rows(a6)-rows(a2);
  temp=temp/2;
```

```
  temp=zeros(temp,1);
  temp=temp|a2|temp;
  a7=polymult(a6,temp);
  temp=a7.==0;
  a7=delif(a7,temp);

@ Step 8: XSA=x-S4=a8x @
  a8=-a7;
  mid=rows(a8)-1;
  mid=mid/2;
  mid=mid+1;
  a8[mid]=1+a8[mid];
  temp=a8.==0;
  x11=delif(a8,temp);

retp(x11);
endp;
@ ---------------------------------------------------------------------- @
/* seasadj.prc, 4/11/00, mww
   Monthly Seasonal Adjustment
   Seasonally adjust a series using a X11 if the series
    fails a "non-seasonal" pretest
    Adjustment fails if
    (i) too few obs for pretest
    (ii) missing values in the "middle" of the series
    When adjustment fails then scalar missing value is returned

    Inputs:

    Y -- Series to be adjusted
    size -- size of pretest

    The returns are:

    YS == seasonally adjusted Y (scalar missing value if proc fails)
    san == 0 (no ajustment necessary based on pretest); YS=Y
           1 (adjustment carried out); YS=X11(Y)
           2 Proc fails, too few obs for pretest, YS=scalar missing value
```

```
           3 Proc fails, missing values in interior of Y, YS = Scal Mis. val.
*/

proc(2) = seasadj(y,size);

local ys, san, t, temp, t1, t2, t2a;

  ys=y;
  san=sadet(ys,size);  @ checks for seasonality and applies x11 as required @
  if san == 0;
   retp(ys,san);
  endif;

  if san == 2;
   ys=miss(0,0);
   retp(ys,san);
  endif;

  if san == 1;
    t=seqa(1,1,rows(ys));
    temp=packr(ys~t);
    ys=temp[.,1];
    t1=temp[1,2];             @ Index of First Non-missing value  @
    t2=temp[rows(temp),2];    @ Index of Last Non-missing value   @
    t2a=t1-1+rows(temp);      @ Val of t2 if no internal Missing  @
    if t2 ./= t2a;
       ys=miss(0,0);
       san=3;        @ Missing Values in interior of series @
       retp(ys,san);
    endif;
    ys=x11ar(ys);
    if t1 ./= 1; ys=miss(zeros(t1-1,1),0)|ys; endif;
    if t2 ./= rows(y); ys=ys|miss(zeros(rows(y)-t2,1),0); endif;

    retp(ys,san);
  endif;

endp;
```

```
@ ---------------------------------------------------------------------- @
/* seasadq.prc, 6/2/00, mww
   Quaterly Seasonal Adjustment
   Seasonally adjust a series using a dummy variable regression
    fails a "non-seasonal" pretest
    Adjustment fails if
    (i) too few obs for pretest
    (ii) missing values in the "middle" of the series
    When adjustment fails then scalar missing value is returned

    Inputs:

    Y -- Series to be adjusted
    size -- size of pretest

    The returns are:

    YS == seasonally adjusted Y (scalar missing value if proc fails)
    san == 0 (no ajustment necessary based on pretest); YS=Y
           1 (adjustment carried out); YS=X11(Y)
           2 Proc fails, too few obs for pretest, YS=scalar missing value
           3 Proc fails, missing values in interior of Y, YS = Scal Mis. val.
*/

proc(2) = seasadq(y,size);

local ys, san, t, temp, t1, t2, t2a;

  ys=y;
  san=sadetq(ys,size);  @ checks for seasonality and applies x11 as required @
  if san == 0;
   retp(ys,san);
  endif;

  if san == 2;
   ys=miss(0,0);
   retp(ys,san);
  endif;
```

```
  if san == 1;
    t=seqa(1,1,rows(ys));
    temp=packr(ys~t);
    ys=temp[.,1];
    t1=temp[1,2];              @ Index of First Non-missing value  @
    t2=temp[rows(temp),2];     @ Index of Last Non-missing value   @
    t2a=t1-1+rows(temp);       @ Val of t2 if no internal Missing  @
    if t2 ./= t2a;
       ys=miss(0,0);
       san=3;                  @ Missing Values in interior of series @
       retp(ys,san);
    endif;
    ys=x11arq(ys);
    if t1 ./= 1; ys=miss(zeros(t1-1,1),0)|ys; endif;
    if t2 ./= rows(y); ys=ys|miss(zeros(rows(y)-t2,1),0); endif;

    retp(ys,san);
  endif;

endp;
@ ------------------------------------------------------------------------ @
proc(1) = adjout(y,thr,tflag);

/* -- Adjust for outliers using fraction of IQR

      y = Data series
      thr = threshold in multiples of IQR
      tflag = 0  == replace with missing value
              1  == replace with maximum value
              2  == replace with median value
              3  == replace with local median (obs + or - 3 on each side)
              4  == replace with one-sided median (5 preceding obs)

*/
local missc, missv, z, zm, iqr, ya, iya, iyb, x, isign, jsign, yt;
local j1, j2, ymvec, i, iwin;
local small;
```

```
small = 1.0e-06;

missc=1e+32;
missv=missc.*ones(rows(y),1);
@ -- Compute IQR -- @
z=packr(y);
z=sortc(z,1);
zm=z[0.5*rows(z)];
iqr=z[.75*rows(z)]-z[.25*rows(z)];

if iqr .< small;
 ya=miss(0,0);
 retp(ya);
endif;

ya=abs(y-zm);

iya = ya .gt (thr*iqr);
iyb = ya .le (thr*iqr);
if tflag .== 0;
  x=(iyb .* y) + (iya .* missv);
  x=miss(x,missc);
elseif tflag .== 1;
 isign = y .> 0;
 jsign = -(y .< 0);
 isign=isign+jsign;
 yt=(zm.*ones(rows(y),1)) + isign .* (thr .* ones(rows(y),1));
 x=(iyb .* y) + (iya .* yt);
elseif tflag .== 2;
  x=(iyb .* y) + (iya .* zm);
elseif tflag .== 3;
  @ Compute rolling median @
  iwin=3;  @ Window on either side @
  ymvec=miss(zeros(rows(y),1),0);
  for i (1,rows(y),1);
   j1=maxc(1|(i-iwin));
   j2=minc(rows(y)|(i+iwin));
```

```
   x=packr(y[j1:j2]);
   x=sortc(x,1);
   ymvec[i]=x[0.5*rows(x)];
  endfor;
  x=(iyb .* y) + (iya .* ymvec);
elseif tflag .== 4;
  @ Compute rolling median @
  iwin=5;  @ Window on ones side @
  ymvec=miss(zeros(rows(y),1),0);
  for i (1,rows(y),1);
   j1=maxc(1|(i-iwin));
   j2=i;
   x=packr(y[j1:j2]);
   x=sortc(x,1);
   ymvec[i]=x[0.5*rows(x)];
  endfor;
  x=(iyb .* y) + (iya .* ymvec);
endif;

retp(x);
endp;
@ ------------------------------------------------------------------------------- @
proc(2)=detrend1(y,q);
/* -- Procedure for producing one-sided detrended versions of a
      series, using a white-noise + I(2) model.  This produces a
      very smooth version of the trend component.  The two-sided
      version of the model produces the HP filter.

      A nice description of what this does is given in
      Harvey and Jaeger, JAE, July-Sep. 1993, pp. 231-248

      Inputs:
      y = series to be detrended
      q = relative variance of I(2) component
          Note: HP quarterly uses q=.000625 (Kydland Prescott)
                for monthly data a value of q=.00000075
                matches quarterly gain at 50%, 80% and 90% periods
*/
```

```
local i, vague, f, x, p, xf, h, e, k, xc;
vague=1e+4; @ Vague Prior on I(2) component @

@ -- Initialize System Matrices -- @
f=zeros(2,2);
f[1,1]=2;
f[1,2]=-1;
f[2,1]=1;
x=zeros(2,1);
p=vague*ones(2,2);
p[1,1]=vague+q;
xf=miss(zeros(rows(y),1),0);

i=1; do while i <= rows(y);
 x=f*x;              @ x(t/t-1) @
 p=f*p*f';
 p[1,1]=p[1,1]+q;    @ p(t/t-1) @
 if ismiss(y[i]) .== 0;
  h=p[1,1]+1;         @ variance of y @
  e=y[i]-x[1];        @ innovation @
  k=p[.,1]/h;         @ kalman gain @
  x=x+k*e;            @ x(t/t) @
  p=p-(k*p[1,.]);     @ p(t/t) @
 endif;
 xf[i]=x[1];
i=i+1; endo;

xc=y-xf;

retp(xc,xf);
endp;
@ ---------------------------------------------------------------------------- @
proc(1) = med(x);

@ -- Return median of columns of X -- @
local n, n1, n2, y, m, i;
m=miss(zeros(cols(x),1),0);
n=rows(x);
```

```
n1=floor(n/2);
n2=ceil(n/2);
if n1 ./= n2;
 n1=n2;
else;
 n2=n1+1;
endif;
i=1; do while i <= cols(x);
 y=sortc(x[.,i],1);
 m[i]=meanc(y[n1:n2]);
i=i+1; endo;

retp(m);
endp;

@ ----------------------------------------------------------- @
proc(1) = tr_mean(x,trimpct);

@ -- Return trimmed mean of columns of X -- @

/*
INPUT:
trimpct = trimming percent (as a fraction, e.g. .02)

Output
trimmean = column vector of trimmed means
*/

local trimmean, n1, n2, ic, tmp;

trimmean=miss(zeros(cols(x),1),0);
n1=1+ceil(trimpct*rows(x));
n2=rows(x)-ceil(trimpct*rows(x));
if n1 .< n2;
 ic=1; do while ic <= cols(x);
   tmp=sortc(x[.,ic],1);
   trimmean[ic]=meanc(tmp[n1:n2]);
 ic=ic+1; endo;
```

```
endif;

retp(trimmean);
endp;
@ -----------------------------------------------------  @
@ PCTILE.PRC @
@   computes percentiles of a column vector @
proc (1)=pctile(x,pct);
 local xpct ;
 x=sortc(x,1);
 pct=ceil(pct*rows(x));
 xpct=x[pct];
retp(xpct);
endp;
@----------------------------------------------------------------------------------------- @

proc(3) = qlra(y,x1,x2,ccut,nma);
@ -- Computes QLR Statistic -- Robust and Non-Robust

   Input:
   y = lhv data
   x1 = rhv data with fixed coefficients under null and alternative
        (input an a scalar [1,1] matrix if all variables
         are allowed to vary)
   x2 = rhv data with fixed coefficients under null and
        time varying coefficients under alternative
   ccut=endpoints for sequential chow regressions

    nma -- number of MA components for HAC Matrix
           (0 => White Hetero Robust)

    Output:

    LM -- QLR Statistic
    LMR -- QLR Robust Statistic
    lsbreak -- least squares estimate of break date
@
 local nobs, n1t, n2t, ktrim, sxy, e0, ss0, i, sxx, syy,
```

```
      sx2y, sxx2, sx2x2, ssb, mxx, mxy,
      qlr, maxobs, sxx2i, x2t, x2ta, kap, mxxi, beta, w, e1, we,
      wewe, vbeta, x, temp;
local k, vbetar, s2, kern, ii, v, r1, r2, lr, lrr, lm, lmr, ssr, ssrvec;
local lsbreak, big;

nobs=rows(y); ktrim=floor(ccut*nobs); n1t=ktrim; n2t=nobs-ktrim;
k=cols(x2);

big=1.0e+15;
lr=-big*ones(nobs,1);
lrr=-big*ones(nobs,1);
ssrvec=big*ones(nobs,1);

@ N-W Kernel @
kern=zeros(nma+1,1);
ii = 0; do while ii <= nma;
 kern[ii+1,1]=1;
 if nma .> 0;
  kern[ii+1,1]=(1-(ii/(nma+1)));
 endif;
ii=ii+1; endo;

x=x2; if rows(x1) ./= 1; x=x1~x2; endif;
@ full sample @
 sxy=x'y; sxx=x'x; syy=y'y;

 sx2y=x2[1:n1t-1,.]'y[1:n1t-1,.];
 sx2x2=x2[1:n1t-1,.]'x2[1:n1t-1,.];
 sxx2=x[1:n1t-1,.]'x2[1:n1t-1,.];

  i=n1t; do until i>n2t;
   sx2y=sx2y+x2[i,.]'y[i,.];
   sx2x2=sx2x2+x2[i,.]'x2[i,.];
   sxx2=sxx2+x[i,.]'x2[i,.];
   mxx=(sxx~sxx2)|(sxx2'~sx2x2);
   mxy=sxy|sx2y;
   mxxi=invpd(mxx);
```

```
    beta=mxxi*mxy;
    w=x~(x2[1:i,.]|zeros(nobs-i,cols(x2)));
    e1=y-w*beta;
    ssr=e1'e1;
    ssrvec[i]=ssr;
    s2=ssr/(rows(e1)-rows(beta));

  @ Form Hetero-Serial Correlation Robust Covariance Matrix @
    we=w.*e1;
    v=zeros(cols(we),cols(we));
    ii = -nma; do while ii <= nma;
     if ii <= 0;
      r1=1;
      r2=rows(we)+ii;
     else;
      r1=1+ii;
      r2=rows(we);
     endif;
     v=v + kern[abs(ii)+1,1]*(we[r1:r2,.]'we[r1-ii:r2-ii,.]);
    ii=ii+1; endo;

    vbetar=mxxi*(v)*mxxi;
    vbeta=s2*mxxi;
    beta=beta[cols(x)+1:rows(beta),1];
    vbetar=vbetar[cols(x)+1:cols(vbetar),cols(x)+1:cols(vbetar)];
    vbeta=vbeta[cols(x)+1:cols(vbeta),cols(x)+1:cols(vbeta)];
    lr[i]=beta'(invpd(vbeta))*beta;
    lrr[i]=beta'(invpd(vbetar))*beta;
   i=i+1; endo;

    lm=maxc(lr);
    lmr=maxc(lrr);
    lsbreak=minindc(ssrvec);
retp(lm,lmr,lsbreak);
endp;

@ --------------------------------------------------------------- @
proc(1) = pval_qlr(x,n,ccut);
```

```
/* pval_qlr.prc, mww, 1/27/97
   Calculate pvalue for QLR tests.
   x = value of test statistic
   n = number of degrees of freedon
   ccut = trimming constant

   Pvalue is computed using the approximation in
   Hansen, 1995, JBES, "Approximate Asymptotic P Values ... "
   pages 60-67.

*/
local pval, parmvec, d, di, dii, p, x1;

@ -- Note, this is now only implemented for ccut = .15 -- @
if ccut ./= .15;
 pval=9999;
 retp(pval);
endif;

let parmvec[24,4] =
1    -0.99 1.02  3.0
2    -1.65 1.06  4.7
3    -2.05 1.13  6.8
4    -2.52 1.11  8.0
5    -3.46 1.07  8.3
6    -4.05 1.08  9.5
7    -4.42 1.10 11.0
8    -5.36 1.08 11.3
9    -5.43 1.10 13.1
10   -6.47 1.06 12.8
11   -6.79 1.04 13.5
12   -7.80 1.02 13.6
13   -7.93 1.07 15.9
14   -8.54 1.05 16.1
15   -9.05 1.05 17.2
16   -9.13 1.09 19.3
17  -10.45 1.05 18.3
18  -10.63 1.05 19.5
```

```
19  -12.14 0.90 14.9
20  -12.14 0.97 18.3
25  -14.16 1.05 25.0
30  -17.06 1.03 27.8
35  -20.09 0.98 28.4
40  -21.65 1.05 36.7;

@ -- Find Appropriate Row of Parmvec -- @
d=abs(parmvec[.,1]-n*ones(rows(parmvec),1));
di=minc(d);
if di .> .0001;
  pval=9999;
  retp(pval);
 else;
  dii= (d .== di);
  p=selif(parmvec,dii);
  x1=p[1,2]+p[1,3]*x;
  if x1 .>= 0;
   pval=cdfchic(x1,p[1,4]);
  else;
   pval=1;
  endif;
  retp(pval);
endif;

endp;
@ -------------------------------------------------------- @
proc(2) = hac(y,x,nma,ikern);
/*
     Modified by MWW, 12-28-96

     Procedure for estimating the regression y = xbeta+ u
     The procedure produces the OLS estimate of b
     and a hetero/autocorrelation consistent estimate of
     the autocorrelation matrix.

Input:
     y = tx1
```

```
     x = txk
     nma=truncation parameter (nma=0, White SEs)
     ikern = kernel indicator
             1 => triangular
             2 => rectangular

Output:
     Beta = OLS estimate of beta (kx1)
    VBeta = Robust estimate of covariance matrix of beta (kxk)
            (Note this is computed used PINV, if X'X is singular

*/
local u, z, ii, xxi, r1, r2, v, kern, beta, vbeta, oldval, xx;
xx=x'x;
beta=x'y/(xx);
u=y-x*beta;

z=x.*u;
v=zeros(cols(x),cols(x));


@ Form Kernel @
kern=zeros(nma+1,1);
ii = 0; do while ii <= nma;
 kern[ii+1,1]=1;
 if nma .> 0;
 if ikern .== 1; kern[ii+1,1]=(1-(ii/(nma+1))); endif;
 endif;
ii=ii+1; endo;

@ Form Hetero-Serial Correlation Robust Covariance Matrix @
ii = -nma; do while ii <= nma;
 if ii <= 0; r1=1; r2=rows(z)+ii; else; r1=1+ii; r2=rows(z); endif;
 v=v + kern[abs(ii)+1,1]*(z[r1:r2,.]'z[r1-ii:r2-ii,.]);
ii=ii+1; endo;
@ -- Use PINV if matrix is not PD -- @
oldval=trapchk(1);
trap 1,1;
```

```
 xxi=invpd(xx);
 if scalerr(xxi);
  xxi=pinv(xx);
 endif;
trap oldval,1;
vbeta=xxi*v*xxi;
retp(beta,vbeta);
endp;
@ -------------------------------------------------------- @
proc(2) = hacm(y,nma,ikern);
/*
     Modified by MWW, 1-28-2004

     Procedure for estimating the mean of the columns of y
     and a HAC estimate of the covariance matrix of the mean

Input:
     y = txm
     nma=truncation parameter (nma=0, no correction for serial correlation)
     ikern = kernel indicator
             1 => triangular
             2 => rectangular

Output:
     ybar = mean of columns of y (m x 1)
    Vybar = Robust estimate of covariance matrix of ybar (m x m)

*/
local ybar, z, ii, v, r1, r2, kern, vybar;
ybar=meanc(y);
z=y-ybar';
v=zeros(cols(z),cols(z));


@ Form Kernel @
kern=zeros(nma+1,1);
ii = 0; do while ii <= nma;
 kern[ii+1,1]=1;
```

```
 if nma .> 0;
 if ikern .== 1; kern[ii+1,1]=(1-(ii/(nma+1))); endif;
 endif;
ii=ii+1; endo;

@ Form Hetero-Serial Correlation Robust Covariance Matrix @
ii = -nma; do while ii <= nma;
 if ii <= 0; r1=1; r2=rows(z)+ii; else; r1=1+ii; r2=rows(z); endif;
 v=v + kern[abs(ii)+1,1]*(z[r1:r2,.]'z[r1-ii:r2-ii,.]);
ii=ii+1; endo;
 v = v/(rows(z)-1);
 vybar=v/rows(z);
retp(ybar,vybar);
endp;
 @ ----------------------------------------------------------- @
 proc(1) = randomint(iL,iU,nr,nc);

 @ Choose a matrix of random integers between iL and iU (inclusive) @
 local z;
 z=il+floor( (iu-il+1)*rndu(nr,nc));
 retp(z);
 endp;

 @ -----------------------------------------------------------  @
 proc(0) = prtmat_comma(x);
 @ Print Matrix in column delimited format
   Note format must be set before this is called @
 local i, j;
 i=1; do while i <=rows(x);
  j=1; do while j <= cols(x);
   x[i,j];;
   if j < cols(x);
    " ,";;
   else;
    "";
   endif;
  j=j+1; endo;
 i=i+1; endo;
```

```
 retp;
 endp;
@ ------------------------------------------------------------  @
 proc(0) = prtmat_char(x,char);
 @ Print Matrix in "character" delimited format
   Note format must be set before this is called

   char is a string character used for the delimitor

   @
 local i, j;
 i=1; do while i <=rows(x);
  j=1; do while j <= cols(x);
   x[i,j];;
   if j < cols(x);
    " ";;char;;" ";;
   else;
    "";
   endif;
  j=j+1; endo;
 i=i+1; endo;
 retp;
 endp;
  @ ------------------------------------------------------------  @
 proc(0) = prtstr_comma(xstr);
 @ Print String in column delimited format
   Note format must be set before this is called @
 local i, j;
 i=1; do while i <=rows(xstr);
  j=1; do while j <= cols(xstr);
   $xstr[i,j];;
   if j < cols(xstr);
    " ,";;
   else;
    "";
   endif;
  j=j+1; endo;
 i=i+1; endo;
```

```
retp;
endp;
@ -----------------------------------------------------------  @
proc(0) = prt_strmat_comma(astr,x);
@ Print string and Matrix in column delimited format
  Note format must be set before this is called @
local i, j;
i=1; do while i <=rows(x);
 j=1; do while j <= cols(astr);
  $astr[i,j];;" ,";;
 j=j+1; endo;
 j=1; do while j <= cols(x);
  x[i,j];;
  if j < cols(x);
   " ,";;
  else;
   "";
  endif;
 j=j+1; endo;
i=i+1; endo;
retp;
endp;
@ -----------------------------------------------------------  @
proc(0) = prtmat_comma_2(x);
@ Print Matrix in column delimited format
  Note format must be set before this is called
  No line break after last entry which is a comman @
local i, j;
i=1; do while i <=rows(x);
 j=1; do while j <= cols(x);
  x[i,j];;
  " ,";;
 j=j+1; endo;
i=i+1; endo;
retp;
endp;
@ -----------------------------------------------------------  @
```

```
proc (4)=kfilt(y,x1,p1,h,f,r,q);

@ Kalman Filter Procedure  -- Hamilton Notation
  y(t) = h'x(t) + w(t)
  x(t) = F x(t-1) + v(t)

  var(w(t)) = r;
  var(v(t)) = q;

  x1 = x(t-1/t-1)  -- on input
  p1 = p(t-1/t-1)  -- on input
@
local x2, p2, e, k, ht;

x2=f*x1;
e=y-h'x2;
p2=f*p1*f'+ q;
ht=h'p2*h + r;
k=p2*h*(invpd(ht));
x1=x2+k*e;
p1=(eye(rows(p1))-k*h')*p2;

@ Correct any small round-off problems @
p1=0.5*(p1+p1');

retp(x1,p1,x2,p2);

endp;

@ ------------------------------------------------------------  @
proc(2)=Ksmooth(x1,x2,x3,p1,p2,p3,f);
@ Kalman Smoother  @
 local p2i, as, oldval;
 oldval = trapchk(1);
 trap 1,1;
 p2i=invpd(p2);
 trap oldval,1;
 if scalerr(p2i);
```

```
   p2i=pinv(p2);
 endif;
 as=p1*f'*p2i;
 p3=p1+as*(p3-p2)*as';
 x3=x1+as*(x3-x2);
retp(x3,p3);
endp;

@ ---------------------------------------------------------- @
proc(1)=bcdate12(nfy,nfm,dnobs);
/* -- Set Up Indicator Vector of Peak and Trough Dates
      -- Monthly
      dnobs == length of vector
      nfy = first year of vector
      nfm = first month of vector
*/
local pk, tr, bcd, i, p1, t1;

/* Set Up Indicators for Peaks and Troughs */
@ -- Monthly Values -- @
let pk[32,2] =
6    1857
10   1860
4    1865
6    1869
10   1873
3    1882
3    1887
7    1890
1    1893
12   1895
6    1899
9    1902
5    1907
1    1910
1    1913
8    1918
1    1920
```

```
5    1923
10   1926
8    1929
5    1937
2    1945
11   1948
7    1953
8    1957
4    1960
12   1969
11   1973
1    1980
7    1981
7    1990
3    2001;

let tr[33,2] =
12    1854
12    1858
6     1861
12    1867
12    1870
3     1879
5     1885
4     1888
5     1891
6     1894
6     1897
12    1900
8     1904
6     1908
1     1912
12    1914
3     1919
7     1921
7     1924
11    1927
3     1933
```

```
6     1938
10    1945
10    1949
5     1954
4     1958
2     1961
11    1970
3     1975
7     1980
11    1982
 3    1991
11    2001;

/* Set up Indicator Array */
bcd=zeros(dnobs,1);

i=1; do while i <= rows(pk);
 p1=12*(pk[i,2]-nfy-1)+pk[i,1]+(13-nfm);  @ - index - @
 if (p1 .>= 1) .and (p1 .<= dnobs);
  bcd[p1]=1;
 endif;
i=i+1; endo;

i=1; do while i <= rows(tr);
 t1=12*(tr[i,2]-nfy-1)+tr[i,1]+(13-nfm);  @ - index - @
 if (t1 .>= 1) .and (t1 .<= dnobs);
  bcd[t1]=1;
 endif;
i=i+1; endo;

retp(bcd);
endp;

@ --------------------------------------------------------- @
proc(1)=bccdate4(nfy,nfq,dnobs);
/* -- Set Up Indicator Vector of Peak and Trough Dates
      -- Quarterly
      dnobs == length of vector
```

```
      nfy = first year of vector
      nfq = first quarter of vector
*/
local pk, tr, bcd, i, p1, t1;

/* Set Up Indicators for Peaks and Troughs */
@ -- Quarterly Values -- @
let pk[9,2] =   4 48
                3 53
                3 57
                2 60
                4 69
                4 73
                1 80
                3 81
                3 90;

let tr[9,2] =   4 49
                2 54
                2 58
                1 61
                4 70
                1 75
                3 80
                4 82
                1 91;

/* Set up Indicator Array */
bcd=zeros(dnobs,1);

i=1; do while i <= rows(pk);
 p1=4*(pk[i,2]-nfy-1)+pk[i,1]+(5-nfq);  @ - index - @
 if (p1 .>= 1) .and (p1 .<= dnobs);
  bcd[p1]=1;
 endif;
i=i+1; endo;

i=1; do while i <= rows(tr);
```

```
 t1=4*(tr[i,2]-nfy-1)+tr[i,1]+(5-nfq);  @ - index - @
 if (t1 .>= 1) .and (t1 .<= dnobs);
  bcd[t1]=1;
 endif;
i=i+1; endo;

retp(bcd);
endp;
@ ---------------------------------------------------------------------------------- @
proc(2)=hp1sd(y,q);
/* -- Procedure for producing one-sided detrended versions of a
      series, using a white-noise + I(2) model.  This produces a
      very smooth version of the trend component.  The two-sided
      version of the model produces the HP filter.

      A nice description of what this does is given in
      Harvey and Jaeger, JAE, July-Sep. 1993, pp. 231-248

      Inputs:
      y = series to be detrended
      q = relative variance of I(2) component
          Note: HP quarterly uses q=.000625 (Kydland Prescott)
                for monthly data a value of q=.00000075
                matches quarterly gain at 50%, 80% and 90% periods
*/
local i, vague, f, x, p, xf, h, e, k, xc;
vague=1e+4; @ Vague Prior on I(2) component @

@ -- Initialize System Matrices -- @
f=zeros(2,2);
f[1,1]=2;
f[1,2]=-1;
f[2,1]=1;
x=zeros(2,1);
p=vague*ones(2,2);
p[1,1]=vague+q;
xf=miss(zeros(rows(y),1),0);
```

```
i=1; do while i <= rows(y);
 x=f*x;              @ x(t/t-1) @
 p=f*p*f';
 p[1,1]=p[1,1]+q;    @ p(t/t-1) @
 if ismiss(y[i]) .== 0;
  h=p[1,1]+1;         @ variance of y @
  e=y[i]-x[1];        @ innovation @
  k=p[.,1]/h;         @ kalman gain @
  x=x+k*e;            @ x(t/t) @
  p=p-(k*p[1,.]);     @ p(t/t) @
 endif;
 xf[i]=x[1];
i=i+1; endo;

xc=y-xf;

retp(xc,xf);
endp;
@ -------------------------------------------------------------------------------- @
proc(2)=hp2sd(y,q);
/* -- Procedure for producing two-sided detrended versions of a
      series, using a white-noise + I(2) model.  This produces a
      very smooth version of the trend component.  The two-sided
      version of the model produces the HP filter.

      A nice description of what this does is given in
      Harvey and Jaeger, JAE, July-Sep. 1993, pp. 231-248

      Inputs:
      y = series to be detrended
      q = relative variance of I(2) component
          Note: HP quarterly uses q=.000625 (Kydland Prescott)
                for monthly data a value of q=.00000075
                matches quarterly gain at 50%, 80% and 90% periods
*/
local i, vague, f, x, p, xf, h, e, k, xc;
local nobs, t, x1, x2, p1, p2, x1t, x2t, p1t, p2t, x3, p3;
vague=1e+4; @ Vague Prior on I(2) component @
```

```
@ -- Initialize System Matrices -- @
if ismiss(y) .== 1;
 "HP2sd not implemented for missing data";
  stop;
endif;
nobs=rows(y);
f=zeros(2,2);
f[1,1]=2;
f[1,2]=-1;
f[2,1]=1;
x=zeros(2,1);
p=vague*ones(2,2);
p[1,1]=vague+q;
xf=miss(zeros(nobs,1),0);

x1t=miss(zeros(nobs,2),0);
x2t=miss(zeros(nobs,2),0);
p1t=miss(zeros(nobs,4),0);
p2t=miss(zeros(nobs,4),0);

i=1; do while i <= rows(y);
 x1t[i,.]=x';
 p1t[i,.]=vec(p)';
 x=f*x;              @ x(t/t-1) @
 p=f*p*f';
 p[1,1]=p[1,1]+q;    @ p(t/t-1) @
 x2t[i,.]=x';
 p2t[i,.]=vec(p)';
  h=p[1,1]+1;         @ variance of y @
  e=y[i]-x[1];        @ innovation @
  k=p[.,1]/h;         @ kalman gain @
  x=x+k*e;            @ x(t/t) @
  p=p-(k*p[1,.]);     @ p(t/t) @
i=i+1; endo;

x3=x;
xf[nobs]=x3[1];
```

```
p3=p;
t=nobs; do while t >= 2;
 x2=x2t[t,.]';
 p2=reshape(p2t[t,.],2,2);
 x1=x1t[t,.]';
 p1=reshape(p1t[t,.],2,2);
 {x3,p3}=ksmooth(x1,x2,x3,p1,p2,p3,f);
 xf[t-1]=x3[1];
t=t-1; endo;

xc=y-xf;

retp(xc,xf);
endp;
@ ---------------------------------------------------------------------------- @
proc(1) = mtoq(mdata,caldsm,caldsq,imeth);

/* -- Converts monthly observations to quarterly observations --

      mdata == monthly data series
      caldsm == Tx2 vector of dates (yr,mth) for monthly obs
      caldsq == Nx2 vector of dates (Yr,qtr) for quarterly obs

      imeth == 0  Average over quarter
               1  First Month of Quarter
               2  Second Month of Quarter
               3  Third month of Quarter
*/
local qdata, i, temp, itemp, iy, iq, im;

qdata=miss(zeros(rows(caldsq),1),0);

@ -- Check to make sure that mdata has correct number of observations -- @
if rows(mdata) ./= rows(caldsm);
 "Error in Procedure MTOQ -- wrong number of elements in mdata";
 "Rows of Mdata";;rows(mdata);
 "Rows of Caldsm";;rows(caldsm);
 retp(qdata);
```

```
endif;

@ -- Check to make imeth is in the correct bounds -- @
if imeth .> 3;
 "Error in Procedure MTOQ -- Invalid Value for Imeth";
 "Imeth";;imeth;
 retp(qdata);
endif;


temp=mdata;

if imeth .== 0; @ Form averages @
 temp=miss(zeros(rows(mdata),1),0);
 i=3; do while i <= rows(mdata);
  temp[i]=meanc(mdata[i-2:i]);
 i=i+1; endo;
 imeth=3;
endif;

i=1; do while i <= rows(caldsq);
 iy=caldsq[i,1]; @ Year @
 iq=caldsq[i,2]; @ Quarter @
 im=(3*(iq-1)) + imeth;    @ Month to use @
 itemp=(caldsm[.,1] .== iy) .and (caldsm[.,2] .== im);
 qdata[i]=selif(temp,itemp);
i=i+1; endo;

retp(qdata);
endp;
@ ------------------------------------------------------ @
```