% MATLAB file MIWLE.m %

```
function [theta,v,err,F0,Ohat,H0]=MIWLE(SS,df,theta0,v0,funn,vfr,pr,bd,thrs,btol,maxiter,varargin)

% Maximum Inverse Wishart Likelihood with bias adjustment
% The inverse of sample size is a parameter
% A two stage method is used.
% Code last updated Dec 2013.

% Electronic Supplementary Material to
% QUANTIFYING ADVENTITIOUS ERROR IN A COVARIANCE STRUCTURE AS A RANDOM EFFECT
% Psychometrika, 80(3), 619-624
% by  Hao Wu (Boston College, hao.wu.5@bc.edu) and Michael W. Browne (The Ohio State University)

%%%%%% List of Input Arguments %%%%%%%%%%%%%%%%%%%%
% df is the degrees of freedom.
%   For unadjusted discrepancy function, use df=p(p+1)/2.
% SS is the input covariance matrix.
% theta0 is the initial value of structure parameters in the model
% v0 is the initial value of v=1/m
% fun is a function handle of the Covariance structure
%   of the form [O,err,R,dOdx,iRdOdx]=fun(theta,...)
%   input and output arguments are
%   theta: a row vector of parameters
%   O: the covariance structure
%   [R,err]=chol(O);
%   dOdx: the derivative of O, a p^2 by q matrix
%   iRdOdx=(R'\otimesR')\dOdx.
% vfree denotes whether m is fixed or not:
%   0 means 1/m is treated as fixed at v0;
%   else means it is not fixed.
% pr prints iteration details:
%   0:  none;
%   1:  only F, Res.Cos, cond.#.H, NPB and NEC;
%   2:  only parameter values;
%   3:  both
% bd is a matrix of length(theta) by 2,
%   giving boundary conditions for parameters
```

```
% thrs is the stopping threshold
% btol is the boundary threshold matrix.
%   It can be input as a scalar, a column vector or a 2-columned matrix.
% maxiter is the maximum iterations allowed.
% varargin passes arguments to fun, the function handle

%%%%%% List of Output Arguments %%%%%%%%%%%%%%%%%%%
% theta is the parameter estimate
% v is the estimate of 1/m
% err>0 means nonconvergence in either of the two stages
% F0 is the Inverted Wishart discrepancy function value
% Ohat is the implied covariance matrix
% H0 is the Hessian of the inverted Wishart discrepancy function
%   w.r.t the covariance structure parameters

%%%%%% List of Screen Outputs %%%%%%%%%%%%%%%%%%%
% #: iteration number
% F: inverted Wishart discrepancy function value (1st stage)
%    inverted Wishart -2 log-likelihood value (2nd stage)
% Res.Cos: residual cosine
% cond.#.H: conditional number of Hessian (unconstrained parameters)
% NPB: number of parameters on boundary
% NEC: number of effective constraints
% parameters: covariance structure parameter values
% v: parameter value of v

global S p detS CS fun other alpha coef

S=SS;
fun=funn;
other=varargin;


if nargin<6, error('Too few arguments.');end
s=length(theta0); % number of parameters in the covariance structure.
if nargin<7 || all(pr~=0:3),    pr=1;end
if nargin<8 || isempty(bd),    bd=kron([-Inf,Inf],ones(s,1));end
if nargin<9 || isempty(thrs),    thrs=.1^7; end
```

```
if nargin<10 || isempty(btol),    btol=.1^8; end
if nargin<11 || isempty(maxiter),    maxiter=50; end

if all(size(btol)==[1,1])
     btol=btol*ones(s,2);
elseif size(btol,2)==1
     btol=btol*[1,1];
end

if df<0
    error('degrees of freedom must be positive.');
end
[p,p1]=size(S); % p is the number of manifest variables.
if p~=p1|| ~all(all(isreal(S)))
    error('The covariance matrix should be real square matrix.');
elseif any([s,2]~=size(bd))|| any([s,2]~=size(btol))|| any(size(theta0)~=[1,s])
    error('The size of one or more input matrices is wrong.');
end

bd=[bd(:,1)+btol(:,1),bd(:,2)-btol(:,2)]; % not including v.
theta=theta0';
v=v0;

%---------------------------------------------------
% Initial screening

if any(any(S~=S'))
    display('Covariance matrix not symmetric. Lower half used.');
    S=tril(S);
    S=S+triu(S',1);
end
[CS,err]=chol(S);
if err>0,    error('Covariance matrix not positive definite.'); end

alpha=2*df/p/(p+1); % The adjustment factor.
detS=prod(diag(CS))^2;
coef=[p*(p+1)/2,p*(2*p^2+3*p-1)/12,(p-1)*p*(p+1)*(p+2)/24,p*(6*p^4+15*p^3-10*p^2-30*p+3)/360];
```

```
if any(v<0) || v*(p-1)>1 || any(theta>bd(:,2)) || any(theta<bd(:,1))
        error('initial value on or out of bounds.');
end
[~,err]=feval(fun,theta',other{:});
if err~=0
    error('initial value yields non-positive definite covariance');
end

switch pr
        case 0
        case 1
            display(' #       F          Res.Cos     cond.#.H    NPB NEC');
        case 2
            display(' #       F         parameters');
        case 3
            display(' #       F          Res.Cos     cond.#.H    NPB NEC     parameters');
end


%---------------------------------
% iterations
iter=-1;
inc=true(s,1);
crit=Inf;
d=0;
f=Inf;

while (crit>thrs && f>thrs && iter<=maxiter &&any(inc))
    iter=iter+1;
    theta1=theta+d;
    ubd1=find(theta1>bd(:,2)); % locations of new effective bounds
    lbd1=find(theta1<bd(:,1));
    bd1=[lbd1;ubd1];
    if ~isempty(bd1)
        bds=[bd(lbd1,1);bd(ubd1,2)];
        if pr~=0
            display('back to the boundary.');
```

```
    end
    [r,ind]=min((bds-theta(bd1))./d(bd1));
    d=d*r;
    theta1=theta+d;
    theta1(bd1(ind))=bds(ind);
end
while 1
    [f1,err,g,H]=FgHtheta(theta1,f);
    if err~=0 && pr==0
        d=d/2;
        theta1=theta+d;
        continue;
    elseif err~=0
        display('step halved.')
        d=d/2;
        theta1=theta+d;
        continue;
    end
    break;
end
theta=theta1;
f=f1;
ubd=(theta==bd(:,2));
lbd=(theta==bd(:,1));
invH=inv(H);
d=-invH*g; % unconstrained search direction
if any(d(ubd)>0) ||any(d(lbd)<0) % out of boundary
    lambda=zeros(s,1);
    effbd=~true(s,1);
    effbd(ubd)=true;
    effbd(lbd)=true;
    lambda(effbd)=invH(effbd,effbd)\d(effbd); % would-be lagrange multiplier
    [rmbd,ind]=max((-lambda.*ubd+lambda.*lbd).*effbd); % removable bounds
    while  rmbd>0 % some effective bounds can be removed
        effbd(ind)=false; % remove inequality constraints;
        lambda(effbd)=invH(effbd,effbd)\d(effbd); % would-be lagrange multiplier
        [rmbd,ind]=max((-lambda.*ubd+lambda.*lbd).*effbd); % removable bounds
    end
```

```
        inc(effbd)=false;
        d(inc)=-H(inc,inc)\g(inc);
        d(~inc)=0;
    end
    condH=cond(H);
    crit=max(g(inc).*g(inc)./diag(H(inc,inc))/f);
    switch pr
        case 0
        case 1
            str1=sprintf('%3d',iter);
            str2=sprintf('%0.5e',f);
            str3=sprintf('%0.5e',crit);
            str4=sprintf('%0.5e',condH);
            str5=sprintf('%2d',sum(ubd)+sum(lbd));
            str6=sprintf('%2d',s-sum(inc));
            display([str1,' ',str2,' ',str3,' ',str4,' ',str5,'  ',str6]);
        case 2
            str1=sprintf('%3d',iter);
            str2=sprintf('%0.5e',f);
            str7=sprintf('%+13.5e',theta);
            display([str1,' ',str2,' ',str7]);
        case 3
            str1=sprintf('%3d',iter);
            str2=sprintf('%0.5e',f);
            str3=sprintf('%0.5e',crit);
            str4=sprintf('%0.5e',condH);
            str5=sprintf('%2d',sum(ubd)+sum(lbd));
            str6=sprintf('%2d',s-sum(inc));
            str7=sprintf('%+13.5e',theta);
            display([str1,' ',str2,' ',str3,' ',str4,' ',str5,'  ',str6,'  ',str7]);
    end
end
F0=f; % IW discrepancy function value.

%---------------------
% output of 1st stage and switch to the 2nd stage.
H0=H;
Ohat=feval(fun,theta',other{:});
```

```
if iter>maxiter
    if pr~=0
        display('maximum number of iterations reached without convergence.');
    end
    err=1;
    v=NaN;
    theta=NaN(s,1);
    F0=NaN;
    H0=nan(s,s);
    Ohat=[];
    return;
elseif vfr==0
    v=v0;
    err=0;
    return;
end

switch pr
        case 0
        otherwise
            display(' #       F          Res.Cos     H       v');
end

%--------------------------------
% iterations
iter=-1;
crit=Inf;
d=0;
f=Inf;
inc=1;
while (crit>thrs && iter<=maxiter && ~isempty(inc))
    iter=iter+1;
    v1=v+d;
    if v1<0.1^7 || v1>1/(p-1)-.1^7
        if pr~=0
            display('back to the boundary.');
        end
        v1=.1^7*(v1<0.1^7)+(1/(p-1)-.1^7)*(v1>1/(p-1)-.1^7);
```

```
            d=v1-v;
        end
        while 1
            [f1,err,g,H]=FgHv(v1,F0,f);
            if err~=0 && pr==0
                d=d/2;
                v1=v+d;
                continue;
            elseif err~=0
                display('step halved.')
                d=d/2;
                v1=v+d;
                continue;
            end
            break;
        end
        v=v1;
        f=f1;
        if v==0.1^7 && g>=0 || v>=(1/(p-1)-.1^7) && g<=0
           inc=[];
        end
        d=-g/H;
        crit=g^2/H;
        switch pr
            case 0
            otherwise
                str1=sprintf('%3d',iter);
                str2=sprintf('%0.5e',f);
                str3=sprintf('%0.5e',crit);
                str4=sprintf('%0.5e',H);
                str5=sprintf('%0.5e',v);
                display([str1,' ',str2,' ',str3,' ',str4,' ',str5]);
        end
    end

    %---------------------
    % output
    if iter<=maxiter
```

```
    err=0;
else
    if pr~=0

        display('maximum iteration reached without convergence.');
    end
    err=1;
    v=NaN;
end

end % of main function

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

function [F,errt,dFdx,d2Fdxdx]=FgHtheta(theta,F0)

global S detS CS p fun other

[O,errt,R]=feval(fun,theta',other{:}); %R'R=O;

if  errt~=0
    [F,dFdx,d2Fdxdx]=deal(NaN);
    errt=1;
    return;
end
detO=prod(diag(R))^2;

dif=CS'\(O-S)/CS;
F=max([0,log(detS/detO)+trace(dif)]);

if F>F0
    errt=3;
    [dFdx,d2Fdxdx]=deal(NaN);
    return;
end

%----------------------------------
    % gradient and Hessian
```

```
if isempty(theta)
    dFdx=[];
    d2Fdxdx=[];
else
    [~,~,R,~,iRdOiR]=feval(fun,theta',other{:}); % a p^2 by s matrix
    A=CS'\R';
    A=A'*A-eye(p);
    dFdx=iRdOiR'*A(:);
    d2Fdxdx=iRdOiR'*iRdOiR;
end
errt=0;

end % end of function FgHtheta

%-------------------------------------------------------
function [f,errv,dfdv,d2fdv2]=FgHv(v,F0,f0)

    % f = -2lnL
    % F0 is inverted Wishart discrepancy function value
    % f0 is F value of the last round

global p alpha coef

if v==0 && F0>0
        f=Inf;
elseif v<0
        errv=2;
        [f,dfdv,d2fdv2]=deal(NaN);
        return;
elseif v==0 && F0==0
        f=-Inf;
elseif (p==1 && v<=0.01)|| (p>1 && (p-1)*v<=.01)
        f=alpha*(coef*[log(v),v,v^2,v^3]'+p*(p-1)/2*log(2*pi)+p*log(4*pi))+F0/v;
else
        i=1:p;
        m=1/v;
        f=alpha*(2*sum(gammaln((m-i+1)/2))-m*p*(log(m/2)-1)+p*(p-1)/2*log(pi))+F0*m;
```

```
end

if f>f0
    errv=3;
    [dfdv,d2fdv2]=deal(NaN);
    return;
end
    %--------------------------------
    % gradient and Hessian

if v==0
    dfdv=NaN;
    d2fdv2=NaN;
elseif  (p==1 && v<=0.01)|| (p>1 && (p-1)*v<=.01)
    dfdv=alpha*coef*[1/v,1,2*v,3*v^2]'-F0/v^2;
    d2fdv2=max([1e-8,alpha*coef*[-1/v^2,0,2,6*v]'+2*F0/v^3]);
else
    i=1:p;
    m=1/v;
    dcdm=sum(psi((m-i+1)/2))-p*log(m/2);
    dfdv=-m^2*(alpha*dcdm+F0);
    d2fdv2=max([1e-8,alpha*(m^4*sum(psi(1,(m-i+1)/2))/2-p*m^3+2*m^3*dcdm)+2*F0/v^3]);
end
errv=0;

end
```