gensysToAMA: A Matlab Implementation of the Anderson-Moore Algorithm Using **gensys** Input and Output Matrices

Gary S. Anderson

April 24, 2007

Abstract

This note describes a Matlab program for solving linear rational expectation problems. The **gensysToAMA** program provides a version of the Anderson-Moore algorithm (AMA) that has a matrix interface exactly the same as the **gensys** program. The code allows the user to invoke the AMA solution code, a copy of their own version of **gensys**, or a copy of **gensys** that was available in early 2007. The code can also verify that the solutions obtained using **gensys** and AMA are equivalent. Timing tests reveal that, except for problems of small dimension, **gensysToAMA** computes solutions much more quickly than **gensys**.

Contents

1 Introduction and Summary

This note describes a Matlab program for solving linear rational expectation problems. The **gensysToAMA** program provides a version of the Anderson-Moore algorithm (AMA) that has a matrix interface exactly the same as the **gensys** program. The code allows the user to invoke the AMA solution code, a copy of their own version of **gensys**, or a copy of **gensys** that was available in early 2007. The code can also verify that the solutions obtained using **gensys** and AMA are equivalent. Timing tests reveal that, except for problems of small dimension, **gensysToAMA** computes solutions much more quickly than **gensys**.

2 Usage

2.1 Installation

1. **unzip the files** into a directory (*someDir*) accessible by Matlab. This will create a directory, **gensysToAMA**Dist, containing the **gensysToAMA** programs and some example .mat input matrix files.

2. start **matlab**

3. place the **gensysToAMA** directory on the Matlab path using

   addpath *someDir*/gensysToAMADist

4. during a matlab session, you can run a quick test of the installation:

   type

   >>**isGensysToAMAOK**

   to verify the **gensysToAMA** program functions correctly. After a few seconds, you should get a **"SUCCESS"** message.

2.2 Examples using gensysToAMA

The installation directory provides a number of ".mat" files that contain input matrices for the gensys (or gensysToAMA) program. These models vary in size and required computation time. For example, the two canada ".mat" files characterize the largest models and require the most computation time. Typing "help gensysToAMA provides information on the gensysToAMA inputs and outputs.

gensysToAMA

```
>> help gensysToAMA
 function [G1,CC,impact,fmat,fwt,ywt,gev,eu]=gensysToAMA(g0,g1,cc,psi,pi,div,
 optionalArg)
  gensys interface to both gensys and the Anderson-Moore algorithm.
  Just as with gensys, system given as
         g0*y(t)=g1*y(t-1)+c+psi*z(t)+pi*eta(t),
  with z an exogenous variable process and eta being endogenously determined
  one-step-ahead expectational errors.  Returned system is
        y(t)=G1*y(t-1)+C+impact*z(t)+ywt*inv(I-fmat*inv(L))*fwt*z(t+1) .
  If z(t) is i.i.d., the last term drops out.
  If div is omitted from argument list, a div>1 is calculated.
  eu(1)=1 for existence, eu(2)=1 for uniqueness.  eu(1)=-1 for
  existence only with not-s.c. z; eu=[-2,-2] for coincident zeros.
  By Christopher A. Sims
  >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
  when called with no optional args,
         program first tries to find gensys on the matlab path
         if that fails, the program runs gensys2007 in the gensysToAMA directory

  an optional string argument may follow the original gensys arguments
  'gensys'   run gensys
         program first tries to find gensys on the matlab path
         if that fails, the program runs gensys2007 in the gensysToAMA directory
  'gensys2007'   run gensys
         the program runs gensys2007 in the gensysToAMA directory

  'ama'    run the anderson-moore algorithm with gensys inputs and outputs

 'both'    run the anderson-moore algorithm and the gensys program
         verify that solutions are equivalent print out execution times
```

2.2.1 Run both algorithms on a small model

gensysToAMA Example

```
>> load inp
>> who

Your variables are:

cc     div    g0     g1     hmat   nlags  pi     psi

>> [ggg,ccc,iii,fff,ffw,yyw,gev,eue]=gensysToAMA(g0,g1,cc,psi,pi,1.0,'both');
gensysToAMA:running both ama and gensys for comparison
problem dimensions: g0:10 x 10, psi:10 x 2, pi:10 x 5
gensysToAMA:running ama
gensysToAMA:converting ama output to gensys format
gensysToAMA:running gensys
gensysToAMA: trying gensys on your matlab path
gensysToAMA: that failed, using gensys2007 in gensysToAMA dir
gensysToAMA:runs complete
no difference in sims and AMA results
AMATime=1.562500e-002  AMAFTime=0   convertToTime=0 convertFromTime=0  genSysTime=0
>>
```

2.2.2 Large models much faster with AMA

gensysToAMA Example

```
>> load canada2lagas1.mat
>> [ggg,ccc,iii,fff,ffw,yyw,gev,eue]=gensysToAMA(g0,g1,cc,psi,pi,1.0,'both');
gensysToAMA:running both ama and gensys for comparison
problem dimensions: g0:222 x 222, psi:222 x 1, pi:222 x 111
gensysToAMA:running ama
gensysToAMA:converting ama output to gensys format
gensysToAMA:running gensys
gensysToAMA: trying gensys on your matlab path
gensysToAMA: that failed, using gensys2007 in gensysToAMA dir
gensysToAMA:runs complete
no difference in sims and AMA results
AMATime=8.593750e-001  AMAFTime=4.687500e-002
convertToTime=3.125000e-002 convertFromTime=9.375000e-002
genSysTime=4.273438e+001
>>
```

3 Problem Statement and Notation

These algorithms compute solutions for models of the form

$$\sum_{i=-\tau}^{\theta} H_i x_{t+i} = \psi z_t, t = 0, \dots, \infty \tag{1}$$

with initial conditions, if any, given by constraints of the form

$$x_i = x_i^{data}, i = -\tau, \dots, -1 \tag{2}$$

where both τ and θ are non-negative, and x_t is an L dimensional vector of endogenous variables with

$$\lim_{t\to\infty} \|x_t\| < \infty \tag{3}$$

and z_t is a k dimensional vector of exogenous variables.

$$ \tag{4} $$

Solutions can be cast in the form

$$(x_t - x^*) = \sum_{i=-\tau}^{-1} B_i (x_{t+i} - x^*) \tag{5}$$

Given any algorithm that computes the B_i, one can easily compute other quantities useful for characterizing the impact of exogenous variables. For models with $\tau = \theta = 1$ the formulae are especially simple.

Let

$$\phi = (H_0 + H_1 B_1)^{-1} \tag{6}$$

$$F = -\phi H_1 B \tag{7}$$

We can write

$$(x_t - x^*) = B_1 (x_{t-1} - x^*) + \sum_{s=0}^{\infty} F^s \phi \psi z_{t+s} \tag{8}$$

and when

$$z_{t+1} = \Upsilon z_t \tag{9}$$

$$vec(\vartheta) = (I - \Upsilon^T \otimes F)^{-1} vec(\phi \psi) \tag{10}$$

$$(x_t - x^*) = B_1 (x_{t-1} - x^*) + \vartheta z_t \tag{11}$$

Consult Anderson [1997] for other useful formulae concerning rational expectations model solutions.

4 Algorithmic Solution Concepts

The following sections present the inputs and outputs for each of the algorithms for the following simple example:

$$V_{t+1} = (1 + R) V_t - D_{t+1} \tag{12}$$

$$D = (1 - \delta) D_{t-1} \tag{13}$$

4.1 Anderson-Moore

Inputs		
$$\sum_{i=-\tau}^{\theta} H_i x_{t+i} = \psi z_t \quad (14)$$ (15)		
Model Variable	Description	Dimensions
$x_{t-\tau}, \ldots, x_t, \ldots, x_{t+\theta}$	Model Variables	$L(\tau+\theta) \times 1$
z_t	Exogenous Variables	$M \times 1$
θ	Longest Lead	1×1
τ	Longest Lag	1×1
H_i	Structural Coefficients Matrix	$(L \times L)(\tau+\theta+1)$
ψ	Exogenous Shock Coefficients Matrix	$L \times M$
Υ	Optional Exogenous VAR Coefficients Matrix($z_{t+1} = \Upsilon z_t$)	$M \times M$

Outputs		
$$x_t = B \begin{bmatrix} x_{t-\tau} \\ \vdots \\ x_{t-1} \end{bmatrix} + \begin{bmatrix} 0 & \ldots 0 & I \end{bmatrix} \sum_{s=0}^{\infty} (F^s \begin{bmatrix} 0 \\ \phi\psi z_{t+s} \end{bmatrix}) \quad (16)$$ (17)		
Model Variable	Description	Dimensions
B	reduced form coefficients matrix	$L \times L(\tau+\theta)$
ϕ	exogenous shock scaling matrix	$L \times L$
F	exogenous shock transfer matrix	$L\theta \times L\theta$
ϑ	autoregressive shock transfer matrix when $z_{t+1} = \Upsilon z_t$ the infinite sum simplifies to give $x_t = B \begin{bmatrix} x_{t-\tau} \\ \vdots \\ x_{t-1} \end{bmatrix} + \vartheta z_t$	$L \times M$

Anderson-Moore input:

AIM Modeling Language Input
Parameter File Input

$$H = \begin{bmatrix} 0 & 0 & -1.1 & 0 & 1 & 1. \\ 0 & -0.7 & 0 & 1 & 0 & 0 \end{bmatrix}, \psi = \begin{bmatrix} 4. & 1. \\ 3. & -2. \end{bmatrix}, \Upsilon \begin{bmatrix} 0.9 & 0.1 \\ 0.05 & 0.2 \end{bmatrix} \quad (18)$$

produces output:

$$B = \begin{bmatrix} 0. & 1.225 \\ 0. & 0.7 \end{bmatrix} F = \begin{bmatrix} 0.909091 & 0.909091 \\ 0. & 0. \end{bmatrix} \phi = \begin{bmatrix} -0.909091 & 1.75 \\ 0. & 1. \end{bmatrix} \quad (19)$$

$$\phi\psi = \begin{bmatrix} 1.61364 & -4.40909 \\ 3. & -2. \end{bmatrix} \vartheta = \begin{bmatrix} 21.0857 & -3.15714 \\ 3. & -2. \end{bmatrix} \quad (20)$$

Usage Notes for Anderson-Moore Algorithm

1. "Align" model variables so that the data history (without applying model equations), completely determines all of x_{t-1}, but none of x_t.
2. Develop a "model file" containing the model equations written in the "AIM modeling language"
3. Apply the model pre-processor to create MATLAB programs for initializing the algorithm's input matrix,(H). Create Ψ and, optionally, Υ matrices.
4. Execute the MATLAB programs to generate B, ϕ, F and optionally ϑ

Users can obtain code for the algorithm and the preprocessor from the author[1]

4.2 Sims

Inputs

$$\Gamma_0 y_t = \Gamma_1 y_{t-1} + C + \psi z_t + \Pi \eta_t \tag{21}$$

(22)

Model Variable	Description	Dimensions
y_t	State Variables	$L \times 1$
z_t	Exogenous Variables	$M_1 \times 1$
η_t	Expectational Error	$M_2 \times 1$
Γ_0	Structural Coefficients Matrix	$L \times L$
Γ_1	Structural Coefficients Matrix	$L \times L$
C	Constants	$L \times 1$
ψ	Structural Exogenous Variables Coefficients Matrix	$L \times M_1$
Π	Structural Expectational Errors Coefficients Matrix	$L \times M_2$

Outputs

$$y_t = \Theta_1 y_{t-1} + \Theta_c + \Theta_0 z_t + \Theta_y \sum_{s=1}^{\infty} \Theta_f^{s-1} \Theta_z E_t z_{t+s} \tag{23}$$

(24)

Model Variable	Description	Dimensions
Θ_1		$L \times L$
Θ_c		$L \times 1$
Θ_0		$L \times M_1$
Θ_y		$L \times M_2$
Θ_f		$M_2 \times M_2$
Θ_z		$M_2 \times M_1$

[1] http://www.bog.frb.fed.us/pubs/oss/oss4/aimindex.html July, 1999.

Sims input:

$$\Gamma_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1.1 & 0 & 1 & 1. \\ 0 & 1 & 0 & 0 \end{bmatrix}, \Gamma_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0.7 & 0 & 0 \end{bmatrix}, C = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{25}$$

$$\psi = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 4. & 1. \\ 3. & -2. \end{bmatrix}, \Pi = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{26}$$

$$\tag{27}$$

produces output:

$$\Theta_c \begin{bmatrix} 0. \\ 0. \\ 0. \\ 0. \end{bmatrix}, \Theta_0 = \begin{bmatrix} 1.61364 & -4.40909 \\ 3. & -2. \\ 3.675 & -2.45 \\ 2.1 & -1.4 \end{bmatrix}, \Theta_f = \begin{bmatrix} 0.909091 & 1.23405 \\ 0. & 0. \end{bmatrix}, \tag{28}$$

$$\Theta_y = \begin{bmatrix} 1.28794 & 1.74832 \\ -2.2204510^{-16} & -1.1102210^{-16} \\ 1.41673 & 0.702491 \\ -1.1102210^{-16} & 1.22066 \end{bmatrix}, \Theta_z = \begin{bmatrix} -0.0796719 & -2.29972 \\ 2.4577 & -1.63846 \end{bmatrix} \tag{29}$$

$$\Theta_y \Theta_z = \begin{bmatrix} 4.19421 & -5.82645 \\ -2.5516810^{-16} & 6.9254610^{-16} \\ 1.61364 & -4.40909 \\ 3. & -2. \end{bmatrix}, \tag{30}$$

5 Comparing Output and Computation Time

Anderson and Moore describe their algorithm in Anderson and Moore [1983]. Sims descibes his algorithm in Sims [1996], Although they attack a the same class of problems their notation is different. Their computer codes reflect these notational differences. Section A.2 presents the code for converting gensys style input into the form expected by the Anderson-Moore algorithm. style input Section A.3 presents the code for converting Anderson-Moore style output into the form generated by the gensys program.

When running the code using the 'both' option, the code computes the two-norm of the difference between the gensys style output of the two programs.[2] In particular, the code verifies that the following two-norms are smaller than 1×10^{-8}.

$$\| G1_{gensys} - G1_{AMA} \| < 1 \times 10^{-8}$$
$$\| CC_{gensys} - CC_{AMA} \| < 1 \times 10^{-8}$$
$$\| impact_{gensys} - impact_{AMA} \| < 1 \times 10^{-8}$$

Comparing $fmat, fwt, ywt$ must account for that fact that applying any similarity transformation to $fmat$ and correspondingly adjusting ywt and fwt produces an equivalent solution. Consequently, the code verifies that

$$fmat_{gensys} \text{ is similar to } fmat_{AMA}$$
$$\| real(ywt_{gensys} fwt_{gensys}) - ywt_{AMA} fwt_{AMA} \| < 1 \times 10^{-8}$$
$$\| real(ywt_{gensys} fmat_{gensys} fwt_{gensys}) - ywt_{AM} fmat_{AMA} ywt_{AMA} \| < 1 \times 10^{-8}$$

[2]The code does not compare the outputs when the rational expectations model does not have a unique saddle point solution.

References

Gary Anderson. A reliable and computationally efficient algorithm for imposing the saddle point property in dynamic models. URL `http://www.federalreserve.gov/pubs/oss/oss4/aimindex.html`. Unpublished Manuscript, Board of Governors of the Federal Reserve System., 1997.

Gary Anderson and George Moore. An efficient procedure for solving linear perfect foresight models. Unpublished Manuscript, Board of Governors of the Federal Reserve System., 1983.

Christopher A. Sims. Solving linear rational expectations models. Seminar paper, 1996.

A Appendices

A.1 An early 2007 gensys implementation

A.1.1 gensys2007

```
function [G1,C,impact,fmat,fwt,ywt,gev,eu]=gensys2007(g0,g1,c,psi,pi,div)
% function [G1,C,impact,fmat,fwt,ywt,gev,eu]=gensys2007(g0,g1,c,psi,pi,div)
% frozen copy of gensys for gensysToAMA
%for use in case gensys not found on user path
% System given as
%         g0*y(t)=g1*y(t-1)+c+psi*z(t)+pi*eta(t),
% with z an exogenous variable process and eta being endogenously determined
% one-step-ahead expectational errors.  Returned system is
%        y(t)=G1*y(t-1)+C+impact*z(t)+ywt*inv(I-fmat*inv(L))*fwt*z(t+1) .
% If z(t) is i.i.d., the last term drops out.
% If div is omitted from argument list, a div>1 is calculated.
% eu(1)=1 for existence, eu(2)=1 for uniqueness.  eu(1)=-1 for
% existence only with not-s.c. z; eu=[-2,-2] for coincident zeros.
% By Christopher A. Sims
% Corrected 10/28/96 by CAS
eu=[0;0];
realsmall=1e-6;
fixdiv=(nargin==6);
n=size(g0,1);
[a b q z v]=qz(g0,g1);
if ~fixdiv, div=1.01; end
nunstab=0;
zxz=0;
for i=1:n
% ------------------div calc------------
   if ~fixdiv
      if abs(a(i,i)) > 0
         divhat=abs(b(i,i))/abs(a(i,i));
         % bug detected by Vasco Curdia and Daria Finocchiaro, 2/25/2004  A root of
         % exactly 1.01 and no root between 1 and 1.02, led to div being stuck at 1.01
         % and the 1.01 root being misclassified as stable.  Changing < to <= below fixes this.
         if 1+realsmall<divhat & divhat<=div
            div=.5*(1+divhat);
         end
      end
   end
% ----------------------------------------
   nunstab=nunstab+(abs(b(i,i))>div*abs(a(i,i)));
   if abs(a(i,i))<realsmall & abs(b(i,i))<realsmall
      zxz=1;
   end
end
div ;
nunstab;
if ~zxz
%alejandro indicates ordqz faster   [a b q z]=qzdiv2007(div,a,b,q,z);
[a b q z]=qzdiv2007(div,a,b,q,z);
%   [a b q z]=ordqz(div,a,b,q,z);
end
```

```
gev=[diag(a) diag(b)];
if zxz
   disp('Coincident zeros.  Indeterminacy and/or nonexistence.')
   eu=[-2;-2];
   % correction added 7/29/2003.  Otherwise the failure to set output
   % arguments leads to an error message and no output (including eu).
   G1=[];C=[];impact=[];fmat=[];fwt=[];ywt=[];gev=[];
   return
end
q1=q(1:n-nunstab,:);
q2=q(n-nunstab+1:n,:);
z1=z(:,1:n-nunstab)';
z2=z(:,n-nunstab+1:n)';
a2=a(n-nunstab+1:n,n-nunstab+1:n);
b2=b(n-nunstab+1:n,n-nunstab+1:n);
etawt=q2*pi;
% zwt=q2*psi;
[ueta,deta,veta]=svd(etawt);
md=min(size(deta));
bigev=find(diag(deta(1:md,1:md))>realsmall);
ueta=ueta(:,bigev);
veta=veta(:,bigev);
deta=deta(bigev,bigev);
% ------ corrected code, 3/10/04
eu(1) = length(bigev)>=nunstab;
% ------ Code below allowed "existence" in cases where the initial lagged state was free to take on v
% ------ inconsistent with existence, so long as the state could w.p.1 remain consistent with a stabl
% ------ if its initial lagged value was consistent with a stable solution.  This is a mistake, thoug
% ------ are situations where we would like to know that this "existence for restricted initial state
%% [uz,dz,vz]=svd(zwt);
%% md=min(size(dz));
%% bigev=find(diag(dz(1:md,1:md))>realsmall);
%% uz=uz(:,bigev);
%% vz=vz(:,bigev);
%% dz=dz(bigev,bigev);
%% if isempty(bigev)
%%      exist=1;
%% else
%%      exist=norm(uz-ueta*ueta'*uz) < realsmall*n;
%% end
%% if ~isempty(bigev)
%%      zwtx0=b2\zwt;
%%      zwtx=zwtx0;
%%      M=b2\a2;
%%      for i=2:nunstab
%%              zwtx=[M*zwtx zwtx0];
%%      end
%%      zwtx=b2*zwtx;
%%      [ux,dx,vx]=svd(zwtx);
%%      md=min(size(dx));
%%      bigev=find(diag(dx(1:md,1:md))>realsmall);
%%      ux=ux(:,bigev);
%%      vx=vx(:,bigev);
%%      dx=dx(bigev,bigev);
```

```
%%      existx=norm(ux-ueta*ueta'*ux) < realsmall*n;
%% else
%%      existx=1;
%% end
% ---------------------------------------------------------
% Note that existence and uniqueness are not just matters of comparing
% numbers of roots and numbers of endogenous errors.  These counts are
% reported below because usually they point to the source of the problem.
% ------------------------------------------------------------
[ueta1,deta1,veta1]=svd(q1*pi);
md=min(size(deta1));
bigev=find(diag(deta1(1:md,1:md))>realsmall);
ueta1=ueta1(:,bigev);
veta1=veta1(:,bigev);
deta1=deta1(bigev,bigev);
%% if existx | nunstab==0
%%    %disp('solution exists');
%%    eu(1)=1;
%% else
%%     if exist
%%         %disp('solution exists for unforecastable z only');
%%         eu(1)=-1;
%%     %else
%%         %fprintf(1,'No solution.  %d unstable roots. %d endog errors.\n',nunstab,size(ueta1,2));
%%     end
%%     %disp('Generalized eigenvalues')
%%    %disp(gev);
%%    %md=abs(diag(a))>realsmall;
%%    %ev=diag(md.*diag(a)+(1-md).*diag(b))\ev;
%%    %disp(ev)
%% %   return;
%% end
if isempty(veta1)
        unique=1;
else
        unique=norm(veta1-veta*veta'*veta1)<realsmall*n;
end
if unique
   %disp('solution unique');
   eu(2)=1;
else
   fprintf(1,'Indeterminacy.  %d loose endog errors.\n',size(veta1,2)-size(veta,2));
   %disp('Generalized eigenvalues')
   %disp(gev);
   %md=abs(diag(a))>realsmall;
   %ev=diag(md.*diag(a)+(1-md).*diag(b))\ev;
   %disp(ev)
%   return;
end
tmat = [eye(n-nunstab) -(ueta*(deta\veta')*veta1*deta1*ueta1')'];
G0= [tmat*a; zeros(nunstab,n-nunstab) eye(nunstab)];
G1= [tmat*b; zeros(nunstab,n)];
% ----------------------
% G0 is always non-singular because by construction there are no zeros on
```

```
% the diagonal of a(1:n-nunstab,1:n-nunstab), which forms G0's ul corner.
% -----------------------
G0I=inv(G0);
G1=G0I*G1;
usix=n-nunstab+1:n;
C=G0I*[tmat*q*c;(a(usix,usix)-b(usix,usix))\q2*c];
impact=G0I*[tmat*q*psi;zeros(nunstab,size(psi,2))];
fmat=b(usix,usix)\a(usix,usix);
fwt=-b(usix,usix)\q2*psi;
ywt=G0I(:,usix);
% -------------------- above are output for system in terms of z'y -------
G1=real(z*G1*z');
C=real(z*C);
impact=real(z*impact);
% Correction 10/28/96:  formerly line below had real(z*ywt) on rhs, an error.
ywt=z*ywt;
```

A.1.2 qzdiv2007

```
function [G1,C,impact,fmat,fwt,ywt,gev,eu]=gensys2007(g0,g1,c,psi,pi,div)
% function [G1,C,impact,fmat,fwt,ywt,gev,eu]=gensys2007(g0,g1,c,psi,pi,div)
% frozen copy of gensys for gensysToAMA
%for use in case gensys not found on user path
% System given as
%        g0*y(t)=g1*y(t-1)+c+psi*z(t)+pi*eta(t),
% with z an exogenous variable process and eta being endogenously determined
% one-step-ahead expectational errors.  Returned system is
%       y(t)=G1*y(t-1)+C+impact*z(t)+ywt*inv(I-fmat*inv(L))*fwt*z(t+1) .
% If z(t) is i.i.d., the last term drops out.
% If div is omitted from argument list, a div>1 is calculated.
% eu(1)=1 for existence, eu(2)=1 for uniqueness.  eu(1)=-1 for
% existence only with not-s.c. z; eu=[-2,-2] for coincident zeros.
% By Christopher A. Sims
% Corrected 10/28/96 by CAS
eu=[0;0];
realsmall=1e-6;
fixdiv=(nargin==6);
n=size(g0,1);
[a b q z v]=qz(g0,g1);
if ~fixdiv, div=1.01; end
nunstab=0;
zxz=0;
for i=1:n
% ------------------div calc------------
   if ~fixdiv
      if abs(a(i,i)) > 0
         divhat=abs(b(i,i))/abs(a(i,i));
         % bug detected by Vasco Curdia and Daria Finocchiaro, 2/25/2004  A root of
         % exactly 1.01 and no root between 1 and 1.02, led to div being stuck at 1.01
         % and the 1.01 root being misclassified as stable.  Changing < to <= below fixes this.
         if 1+realsmall<divhat & divhat<=div
            div=.5*(1+divhat);
         end
      end
```

```
   end
% ---------------------------------------
   nunstab=nunstab+(abs(b(i,i))>div*abs(a(i,i)));
   if abs(a(i,i))<realsmall & abs(b(i,i))<realsmall
      zxz=1;
   end
end
div ;
nunstab;
if ~zxz
%alejandro indicates ordqz faster   [a b q z]=qzdiv2007(div,a,b,q,z);
[a b q z]=qzdiv2007(div,a,b,q,z);
%   [a b q z]=ordqz(div,a,b,q,z);
end
gev=[diag(a) diag(b)];
if zxz
   disp('Coincident zeros.  Indeterminacy and/or nonexistence.')
   eu=[-2;-2];
   % correction added 7/29/2003.  Otherwise the failure to set output
   % arguments leads to an error message and no output (including eu).
   G1=[];C=[];impact=[];fmat=[];fwt=[];ywt=[];gev=[];
   return
end
q1=q(1:n-nunstab,:);
q2=q(n-nunstab+1:n,:);
z1=z(:,1:n-nunstab)';
z2=z(:,n-nunstab+1:n)';
a2=a(n-nunstab+1:n,n-nunstab+1:n);
b2=b(n-nunstab+1:n,n-nunstab+1:n);
etawt=q2*pi;
% zwt=q2*psi;
[ueta,deta,veta]=svd(etawt);
md=min(size(deta));
bigev=find(diag(deta(1:md,1:md))>realsmall);
ueta=ueta(:,bigev);
veta=veta(:,bigev);
deta=deta(bigev,bigev);
% ------ corrected code, 3/10/04
eu(1) = length(bigev)>=nunstab;
% ------ Code below allowed "existence" in cases where the initial lagged state was free to take on v
% ------ inconsistent with existence, so long as the state could w.p.1 remain consistent with a stabl
% ------ if its initial lagged value was consistent with a stable solution.  This is a mistake, thoug
% ------ are situations where we would like to know that this "existence for restricted initial stat
%% [uz,dz,vz]=svd(zwt);
%% md=min(size(dz));
%% bigev=find(diag(dz(1:md,1:md))>realsmall);
%% uz=uz(:,bigev);
%% vz=vz(:,bigev);
%% dz=dz(bigev,bigev);
%% if isempty(bigev)
%%      exist=1;
%% else
%%      exist=norm(uz-ueta*ueta'*uz) < realsmall*n;
%% end
```

```
%% if ~isempty(bigev)
%%      zwtx0=b2\zwt;
%%      zwtx=zwtx0;
%%      M=b2\a2;
%%      for i=2:nunstab
%%              zwtx=[M*zwtx zwtx0];
%%      end
%%      zwtx=b2*zwtx;
%%      [ux,dx,vx]=svd(zwtx);
%%      md=min(size(dx));
%%      bigev=find(diag(dx(1:md,1:md))>realsmall);
%%      ux=ux(:,bigev);
%%      vx=vx(:,bigev);
%%      dx=dx(bigev,bigev);
%%      existx=norm(ux-ueta*ueta'*ux) < realsmall*n;
%% else
%%      existx=1;
%% end
% ----------------------------------------------------
% Note that existence and uniqueness are not just matters of comparing
% numbers of roots and numbers of endogenous errors.  These counts are
% reported below because usually they point to the source of the problem.
% ------------------------------------------------------
[ueta1,deta1,veta1]=svd(q1*pi);
md=min(size(deta1));
bigev=find(diag(deta1(1:md,1:md))>realsmall);
ueta1=ueta1(:,bigev);
veta1=veta1(:,bigev);
deta1=deta1(bigev,bigev);
%% if existx | nunstab==0
%%    %disp('solution exists');
%%    eu(1)=1;
%% else
%%     if exist
%%         %disp('solution exists for unforecastable z only');
%%         eu(1)=-1;
%%     %else
%%         %fprintf(1,'No solution.  %d unstable roots. %d endog errors.\n',nunstab,size(ueta1,2));
%%     end
%%     %disp('Generalized eigenvalues')
%%    %disp(gev);
%%    %md=abs(diag(a))>realsmall;
%%    %ev=diag(md.*diag(a)+(1-md).*diag(b))\ev;
%%    %disp(ev)
%% %   return;
%% end
if isempty(veta1)
        unique=1;
else
        unique=norm(veta1-veta*veta'*veta1)<realsmall*n;
end
if unique
   %disp('solution unique');
   eu(2)=1;
```

```
else
   fprintf(1,'Indeterminacy.  %d loose endog errors.\n',size(veta1,2)-size(veta,2));
   %disp('Generalized eigenvalues')
   %disp(gev);
   %md=abs(diag(a))>realsmall;
   %ev=diag(md.*diag(a)+(1-md).*diag(b))\ev;
   %disp(ev)
%   return;
end
tmat = [eye(n-nunstab) -(ueta*(deta\veta')*veta1*deta1*ueta1')'];
G0= [tmat*a; zeros(nunstab,n-nunstab) eye(nunstab)];
G1= [tmat*b; zeros(nunstab,n)];
% ----------------------
% G0 is always non-singular because by construction there are no zeros on
% the diagonal of a(1:n-nunstab,1:n-nunstab), which forms G0's ul corner.
% -----------------------
G0I=inv(G0);
G1=G0I*G1;
usix=n-nunstab+1:n;
C=G0I*[tmat*q*c;(a(usix,usix)-b(usix,usix))\q2*c];
impact=G0I*[tmat*q*psi;zeros(nunstab,size(psi,2))];
fmat=b(usix,usix)\a(usix,usix);
fwt=-b(usix,usix)\q2*psi;
ywt=G0I(:,usix);
% -------------------- above are output for system in terms of z'y -------
G1=real(z*G1*z');
C=real(z*C);
impact=real(z*impact);
% Correction 10/28/96:  formerly line below had real(z*ywt) on rhs, an error.
ywt=z*ywt;
```

A.1.3 qzswitch2007

```
function [G1,C,impact,fmat,fwt,ywt,gev,eu]=gensys2007(g0,g1,c,psi,pi,div)
% function [G1,C,impact,fmat,fwt,ywt,gev,eu]=gensys2007(g0,g1,c,psi,pi,div)
% frozen copy of gensys for gensysToAMA
%for use in case gensys not found on user path
% System given as
%        g0*y(t)=g1*y(t-1)+c+psi*z(t)+pi*eta(t),
% with z an exogenous variable process and eta being endogenously determined
% one-step-ahead expectational errors.  Returned system is
%       y(t)=G1*y(t-1)+C+impact*z(t)+ywt*inv(I-fmat*inv(L))*fwt*z(t+1) .
% If z(t) is i.i.d., the last term drops out.
% If div is omitted from argument list, a div>1 is calculated.
% eu(1)=1 for existence, eu(2)=1 for uniqueness.  eu(1)=-1 for
% existence only with not-s.c. z; eu=[-2,-2] for coincident zeros.
% By Christopher A. Sims
% Corrected 10/28/96 by CAS
eu=[0;0];
realsmall=1e-6;
fixdiv=(nargin==6);
n=size(g0,1);
[a b q z v]=qz(g0,g1);
if ~fixdiv, div=1.01; end
```

```
nunstab=0;
zxz=0;
for i=1:n
% ------------------div calc-------------
   if ~fixdiv
      if abs(a(i,i)) > 0
         divhat=abs(b(i,i))/abs(a(i,i));
         % bug detected by Vasco Curdia and Daria Finocchiaro, 2/25/2004  A root of
         % exactly 1.01 and no root between 1 and 1.02, led to div being stuck at 1.01
         % and the 1.01 root being misclassified as stable.  Changing < to <= below fixes this.
         if 1+realsmall<divhat & divhat<=div
            div=.5*(1+divhat);
         end
      end
   end
% -----------------------------------------
   nunstab=nunstab+(abs(b(i,i))>div*abs(a(i,i)));
   if abs(a(i,i))<realsmall & abs(b(i,i))<realsmall
      zxz=1;
   end
end
div ;
nunstab;
if ~zxz
%alejandro indicates ordqz faster   [a b q z]=qzdiv2007(div,a,b,q,z);
[a b q z]=qzdiv2007(div,a,b,q,z);
%   [a b q z]=ordqz(div,a,b,q,z);
end
gev=[diag(a) diag(b)];
if zxz
   disp('Coincident zeros.  Indeterminacy and/or nonexistence.')
   eu=[-2;-2];
   % correction added 7/29/2003.  Otherwise the failure to set output
   % arguments leads to an error message and no output (including eu).
   G1=[];C=[];impact=[];fmat=[];fwt=[];ywt=[];gev=[];
   return
end
q1=q(1:n-nunstab,:);
q2=q(n-nunstab+1:n,:);
z1=z(:,1:n-nunstab)';
z2=z(:,n-nunstab+1:n)';
a2=a(n-nunstab+1:n,n-nunstab+1:n);
b2=b(n-nunstab+1:n,n-nunstab+1:n);
etawt=q2*pi;
% zwt=q2*psi;
[ueta,deta,veta]=svd(etawt);
md=min(size(deta));
bigev=find(diag(deta(1:md,1:md))>realsmall);
ueta=ueta(:,bigev);
veta=veta(:,bigev);
deta=deta(bigev,bigev);
% ------ corrected code, 3/10/04
eu(1) = length(bigev)>=nunstab;
% ------ Code below allowed "existence" in cases where the initial lagged state was free to take on
```

```
% ------ inconsistent with existence, so long as the state could w.p.1 remain consistent with a stabl
% ------ if its initial lagged value was consistent with a stable solution.  This is a mistake, thoug
% ------ are situations where we would like to know that this "existence for restricted initial state
%% [uz,dz,vz]=svd(zwt);
%% md=min(size(dz));
%% bigev=find(diag(dz(1:md,1:md))>realsmall);
%% uz=uz(:,bigev);
%% vz=vz(:,bigev);
%% dz=dz(bigev,bigev);
%% if isempty(bigev)
%%      exist=1;
%% else
%%      exist=norm(uz-ueta*ueta'*uz) < realsmall*n;
%% end
%% if ~isempty(bigev)
%%      zwtx0=b2\zwt;
%%      zwtx=zwtx0;
%%      M=b2\a2;
%%      for i=2:nunstab
%%              zwtx=[M*zwtx zwtx0];
%%      end
%%      zwtx=b2*zwtx;
%%      [ux,dx,vx]=svd(zwtx);
%%      md=min(size(dx));
%%      bigev=find(diag(dx(1:md,1:md))>realsmall);
%%      ux=ux(:,bigev);
%%      vx=vx(:,bigev);
%%      dx=dx(bigev,bigev);
%%      existx=norm(ux-ueta*ueta'*ux) < realsmall*n;
%% else
%%      existx=1;
%% end
% ----------------------------------------------------
% Note that existence and uniqueness are not just matters of comparing
% numbers of roots and numbers of endogenous errors.  These counts are
% reported below because usually they point to the source of the problem.
% ------------------------------------------------------
[ueta1,deta1,veta1]=svd(q1*pi);
md=min(size(deta1));
bigev=find(diag(deta1(1:md,1:md))>realsmall);
ueta1=ueta1(:,bigev);
veta1=veta1(:,bigev);
deta1=deta1(bigev,bigev);
%% if existx | nunstab==0
%%    %disp('solution exists');
%%    eu(1)=1;
%% else
%%     if exist
%%         %disp('solution exists for unforecastable z only');
%%         eu(1)=-1;
%%     %else
%%         %fprintf(1,'No solution.  %d unstable roots. %d endog errors.\n',nunstab,size(ueta1,2));
%%     end
%%     %disp('Generalized eigenvalues')
```

```
%%    %disp(gev);
%%    %md=abs(diag(a))>realsmall;
%%    %ev=diag(md.*diag(a)+(1-md).*diag(b))\ev;
%%    %disp(ev)
%% %   return;
%% end
if isempty(veta1)
        unique=1;
else
        unique=norm(veta1-veta*veta'*veta1)<realsmall*n;
end
if unique
   %disp('solution unique');
   eu(2)=1;
else
   fprintf(1,'Indeterminacy.  %d loose endog errors.\n',size(veta1,2)-size(veta,2));
   %disp('Generalized eigenvalues')
   %disp(gev);
   %md=abs(diag(a))>realsmall;
   %ev=diag(md.*diag(a)+(1-md).*diag(b))\ev;
   %disp(ev)
%   return;
end
tmat = [eye(n-nunstab) -(ueta*(deta\veta')*veta1*deta1*ueta1')'];
G0= [tmat*a; zeros(nunstab,n-nunstab) eye(nunstab)];
G1= [tmat*b; zeros(nunstab,n)];
% ----------------------
% G0 is always non-singular because by construction there are no zeros on
% the diagonal of a(1:n-nunstab,1:n-nunstab), which forms G0's ul corner.
% -----------------------
G0I=inv(G0);
G1=G0I*G1;
usix=n-nunstab+1:n;
C=G0I*[tmat*q*c;(a(usix,usix)-b(usix,usix))\q2*c];
impact=G0I*[tmat*q*psi;zeros(nunstab,size(psi,2))];
fmat=b(usix,usix)\a(usix,usix);
fwt=-b(usix,usix)\q2*psi;
ywt=G0I(:,usix);
% -------------------- above are output for system in terms of z'y -------
G1=real(z*G1*z');
C=real(z*C);
impact=real(z*impact);
% Correction 10/28/96:  formerly line below had real(z*ywt) on rhs, an error.
ywt=z*ywt;
```

A.2 convertFromGensysIn

```
function [theHM,theH0,theHP]=convertFromGensysIn(g0,g1,pi)
%function [theHM,theH0,theHP]=convertFromGensysIn(g0,g1,pi)
gDim=size(g0,1);
piCol=size(pi,2);
theHM=sparse([...
-g1,zeros(gDim,piCol)
zeros(piCol,gDim+piCol)]);
```

```
theH0=sparse([...
g0,-pi;...
zeros(piCol,gDim+piCol)]);
theHP=sparse([...
zeros(gDim,gDim+piCol);...
zeros(piCol,gDim),eye(piCol)]);
```

A.3 convertToGensysOut

```
function [CC,G1,impact,ywt,fmat,fwt]=...
convertToGensysOut(bb,phi,theF,cc,g0,g1,psi,ncpi)
%function [CC,G1,impact,ywt,fmat,fwt]=...
%convertToGensysOut(bb,phi,theF,cc,g0,g1,psi,ncpi)

[nr,nc]=size(g1);
[nrpsi,ncpsi]=size(psi);
stateDim=size(bb,2)-ncpi;
G1=bb(1:nr,1:nc);

ststate=(g0-g1)\cc;
CC=(eye(nr)-G1)*ststate;

thePsi=[psi;zeros(ncpi,ncpsi)];
aa=phi*thePsi;
impact=aa(1:nr,:);
%no unique way to represent these components
[ywt,fmat,fwt]=smallF(theF,aa,stateDim);

function [onLeft,inMiddle,onRight]=smallF(anF,bigPhi,nn)
[fRows,fCols]=size(anF);
lilFL=anF(nn+1:fRows,nn+1:fRows);
uu=null(full(lilFL));
theNull=size(uu,2);
eOpts.disp=0;
lilFU=anF(1:nn,nn+1:fRows);
onLeft=lilFU;
onRight=bigPhi(nn+1:fRows,:);
inMiddle=lilFL;
```

A.4 Linear Algebra for Comparisons

Under construction. Check code in **gensysToAMA** .

$$F = (H_0 + H_+ B)^{-1} H_+ = \phi H_+$$

$$H = \begin{bmatrix} g1 & 0 & g0 & -pi & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & I \end{bmatrix}$$

$$F = \begin{bmatrix} g0 & \pi \\ B_{\pi L} & B_{\pi R} \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix} = \begin{bmatrix} 0 & f_U \\ & f_L \end{bmatrix}$$

$$f_L = V^{-1} \Lambda_f V$$

$$F = \begin{bmatrix} I & \\ & V^{-1} \end{bmatrix} \begin{bmatrix} 0 & f_U \\ & \Lambda_f \end{bmatrix} \begin{bmatrix} I & \\ & V \end{bmatrix}$$

$$F^k = \begin{bmatrix} I & \\ & V^{-1} \end{bmatrix} \begin{bmatrix} 0 & f_U \Lambda_f^{k-1} \\ & \Lambda_f^k \end{bmatrix} \begin{bmatrix} I & \\ & V \end{bmatrix}$$

$$F^k \phi \psi = F^k \begin{bmatrix} \phi_U \\ \phi_L \end{bmatrix} \psi$$

$$F^k \phi \psi = \begin{bmatrix} I & \\ & V^{-1} \end{bmatrix} \begin{bmatrix} 0 & f_U \Lambda_f^{k-1} \\ & \Lambda_f^k \end{bmatrix} \begin{bmatrix} I & \\ & V \end{bmatrix} \begin{bmatrix} \phi_U \\ \phi_L \end{bmatrix} \psi$$

$$F^k \phi \psi = F^k \begin{bmatrix} \phi_U \\ \phi_L \end{bmatrix} \psi$$

$$F^k \phi \psi = \begin{bmatrix} f_U \Lambda\_f^{k-1} V \phi_L \\ V^{-1} \Lambda_f^k V \phi_L \end{bmatrix} \psi$$