IRIS State-Space – Essentials (i)

test_sspace.m

by Michal Andrle (IMF)

4 November 2012

Introduction

This simple tutorial demonstrates how to obtain and utilize the state-space form of the model implied by IRIS first-order (log)-linear solution.

A simple transformation of the IRIS state-space form to a `textbook` one is provided and a very simple simulation is contrasted to results from IRIS `simulate` command.

Contents

1 Housekeeping

```
14 clear all; close all; clc;
```

2 Reading the DSGE model

Loading the state-space model, a non-stationary one.

```
19 load('read_model.mat');
```

3 Get the state-space form

To get the state-space form in IRIS, use the model property `sspace`, which returns all the matrices needed. See also a `help/sspace` `Omega` is the covariance matrix of all shocks, note that transition and measurement errors shocks are stacked into one vector. The proper loading is guaranteed by `R` and `H` matrices.

Variable `list` returned from the `sspace()` command is a cell array of measurement variables names, transition names including auxilliary lags and shocks.

```
33 % please, read Mirek's documentation
34 help model/sspace
35
36 % get the sspace without the Schur transform (triangular= false)
37 [T R K Z H D U Omega list] = sspace(m,'triangular',false);
38
39 % check the structure of the transition vector. note lags and logs...
40 list{2}'
```

```
sspace  State-space matrices describing the model solution.

Syntax
=======

    [T,R,K,Z,H,D,U,Omg] = sspace(m,...)

Input arguments
================

* `m` [ model ] - Solved model object.

Output arguments
=================
```

```
* ‘T‘ [ numeric ] - Transition matrix.

* ‘R‘ [ numeric ] - Matrix at the shock vector in transition equations.

* ‘K‘ [ numeric ] - Constant vector in transition equations.

* ‘Z‘ [ numeric ] - Matrix mapping transition variables to measurement
variables.

* ‘H‘ [ numeric ] - Matrix at the shock vector in measurement
equations.

* ‘D‘ [ numeric ] - Constant vector in measurement equations.

* ‘U‘ [ numeric ] - Transformation matrix for predetermined variables.

* ‘Omg‘ [ numeric ] - Covariance matrix of shocks.

Options
========

* ‘’triangular=’‘ [ *‘true‘* | ‘false‘ ] - If true, the state-space form
returned has the transition matrix ‘T‘ quasi triangular and the vector of
predetermined variables transformed accordingly. This is the form used
in IRIS calculations.

Description
============

The state-space representation has the following form:

    [xf;alpha] = T*alpha(-1) + K + R*e

    y = Z*alpha + D + H*e

    xb = U*alpha

    Cov[e] = Omg

where ‘xb‘ is an nb-by-1 vector of predetermined (backward-looking)
transition variables and their auxiliary lags, ‘xf‘ is an nf-by-1 vector
of non-predetermined (forward-looking) variables and their auxiliary
leads, ‘alpha‘ is a transformation of ‘xb‘, ‘e‘ is an ne-by-1 vector of
shocks, and ‘y‘ is an ny-by-1 vector of measurement variables.
Furthermore, we denote the total number of transition variables,
and their auxiliary lags and leads, nx = nb + nf.
```

```
  The transition matrix, ‘T‘, is, in general, rectangular nx-by-nb.
  Furthremore, the transformed state vector alpha is chosen so that the
  lower nb-by-nb part of ‘T‘ is quasi upper triangular.

  You can use the ‘get(m,’xVector’)‘ function to learn about the order of
  appearance of transition variables and their auxiliary lags and leads in
  the vectors ‘xb‘ and ‘xf‘. The first nf names are the vector ‘xf‘, the
  remaining nb names are the vector ‘xb‘.

ans =
    'log(N)'
    'log(Q)'
    'log(H)'
    'log(Pk)'
    'log(Rk)'
    'log(Lambda)'
    'log(d4P)'
    'log(RMC)'
    'log(Y)'
    'log(W)'
    'log(A)'
    'log(P)'
    'log(R)'
    'log(dP)'
    'log(dW)'
    'log(Y{-1})'
    'log(W{-1})'
    'log(A{-1})'
    'log(P{-1})'
    'log(P{-2})'
    'log(P{-3})'
```

4 Transform the IRIS state-space into a textbook form

Let's transform the IRIS state-space form, which is rather an efficient one from the computational point of view, to a standard, textbook for as can be found in Harvey's 1989 book on Kalman filter... It is trivial, once the trinagular form is corrected for, see `help @model/sspace`

```
48 % get dimensions
49 [ny na2] = size(Z);
50 [nx na1] = size(T);
51 [ans  ne ] = size(H);
52
53 if eq(na1,na2)
```

```
    na = na1;
else
    % one will never get here...
    error('Hmmm something is wrong...');
end

nxf = nx - na;

% nex |X| state vector is X := [XF; ALPHA]
% create a new transition matrix...
TT = zeros(nx, nx);
    TT(:,1:nxf) = 0;
    TT(:,nxf+1:end) = T; % TT = [0 T];

% create a new measurement matrix...
ZZ = zeros(ny, nx);
    ZZ(:,nxf+1:end) = Z; % ZZ = [0 Z];
```

5 Simulate a simple impulse response simulation

```
shock2simulate = 'Ea';
shockvalue     = 0.01;
nper           = 40.0;

% simulation
shoxNames = list{3}; % shox names = eList...

% assign the simulation (path, IRF, anything...)
ee = zeros(ne, nper);
    ix = find(strcmpi(shock2simulate,shoxNames));
    ee(ix,1) = shockvalue;
```

6 Simulate a path of shocks

Due to a non-stationarity of the model, we will simulate deviations around its balanced growth path. We will ignore the initial conditions (same for both shock and control simulation) and constants/drifts. For this purpose a variable `dev` is created, taking the value of zero.

```
% allocate the simulation path
yy = zeros(ny, nper);
xx = zeros(nx, nper);

x_init = zeros(nx,1); % initial conditions...
```

```
dev    = 0;

% textbook form... sound, familiar ha?
t = 1;
    xx(:,1) = dev*(K + x_init) + R*ee(:,1);
    yy(:,1) = ZZ*xx(:,1) + dev*D + H*ee(:,1);
for t = 2  : nper
    xx(:,t) = TT*xx(:,t-1) + dev*K + R*ee(:,t);
    yy(:,t) = ZZ*xx(:,t  ) + dev*D + H*ee(:,t);
end
```

7 Simulate the IRIS simulation...

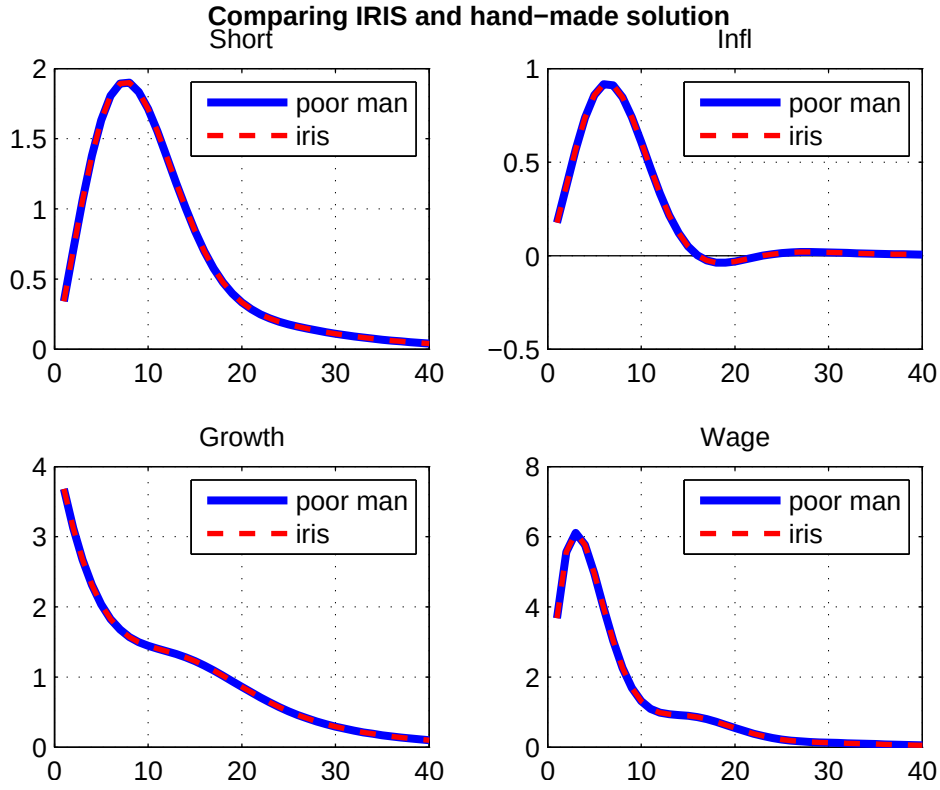
This is a standard IRIS routine for simulating a path of shocks, in this case leading just to a simple impulse...

```
dbss = sstatedb(m,1:nper);
dbss.(shock2simulate)(1) = shockvalue;
ds = simulate(m, dbss, 1:nper, 'anticipate', false,'deviation',false);
```

8 Compare with the IRIS solution...

In our particular case, due to the model structure, neither of observables is in logs, so we can directly compare the results to IRIS simulation... This will NOT work for other transition variables. For those one needs to treat logs if needed.

```
ylist = get(m,'ylist');
f = figure();
    for i = 1 : numel(ylist)
        subplot(2,2,i);
            tmp = ((ds.(ylist{i}))-(dbss.(ylist{i})));
            pp = plot([yy(i,:)' tmp(1:nper)]);
                set(pp(1),'linewidth',3,'color','b');
                set(pp(2),'linewidth',2,'color','r','linestyle','--');
            legend('poor man','iris');
            title(ylist{i},'interpreter','none');
            grid on;
            grfun.zeroline();
    end
    ftitle('Comparing IRIS and hand-made solution');
```



9 Make the units right (log or non-log?)

The model can be log-linearized for some variables and linearized in terms the others. This is important to understand and treat properly. `islog` property of the model obj. returns a structure with 0/1 indicating if it the variable is in logs.

```
145 islog = get(m,'islog')
```

```
islog =
     Short: 0
      Infl: 0
    Growth: 0
      Wage: 0
         Y: 1
         N: 1
         W: 1
         Q: 1
         H: 1
```

```
        A: 1
        P: 1
        R: 1
       Pk: 1
       Rk: 1
   Lambda: 1
       dP: 1
      d4P: 1
       dW: 1
      RMC: 1
       Mp: 0
       Mw: 0
       Ey: 0
       Ep: 0
       Ea: 0
       Er: 0
       Ew: 0
```

10 Help for the functions used in the tutorial

help model/sspace help model/get help model/specget help ftitle